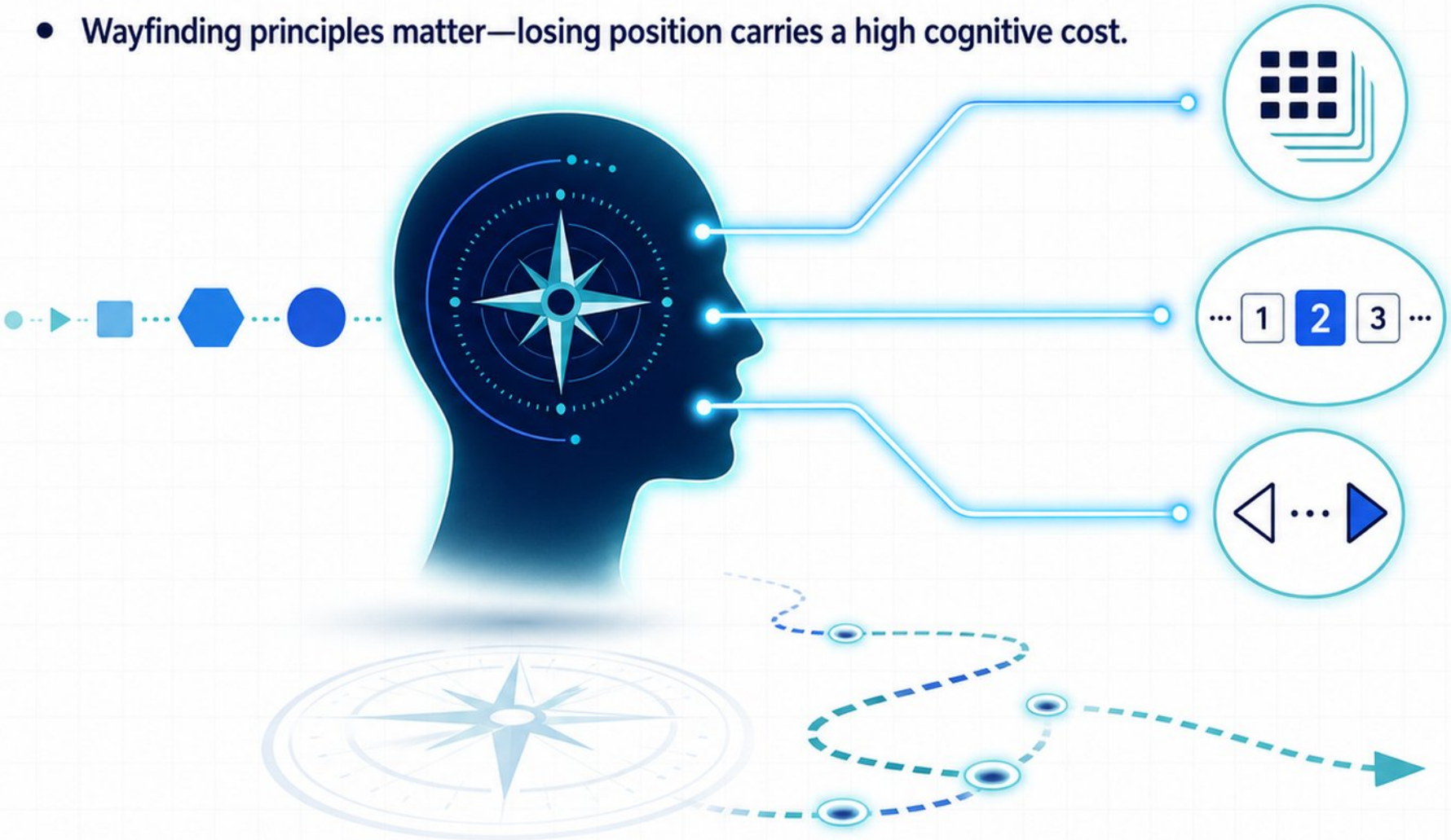
Introduction to Designing Pagination

- Pagination breaks large datasets into numbered pages for manageable navigation.
- Users need to understand collection length, current position, and reachable pages.
- Choose pagination for finding and comparing; infinite scroll for browsing; load-more for controlled exploration.
- This training covers mental models, control patterns, implementation trade-offs, and accessibility.

The Core Mental Model

- Translate user questions into three signals: total items, current page, and reachable adjacent pages.
- Users estimate scope from cues like "Page 2 of 24" or "1–50 of 13,241 results."
- Orientation answers: "Where am I now, and where can I go next?"
- Wayfinding principles matter—losing position carries a high cognitive cost.

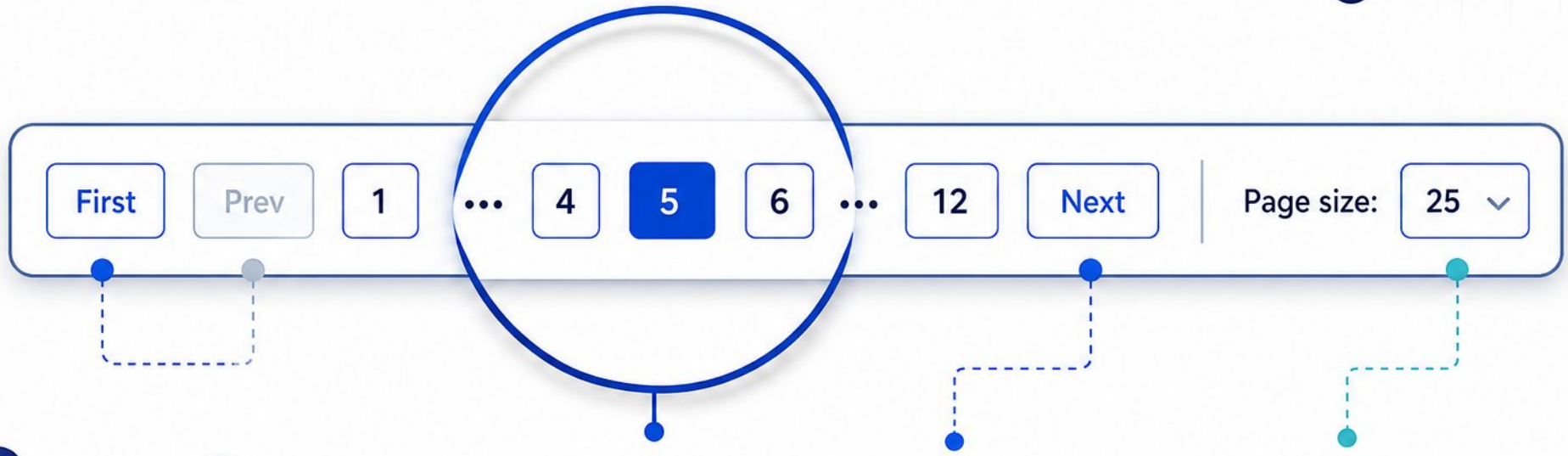


Choosing the Right Content-Loading Pattern

- **Pagination:** best for search and comparison tasks needing stable URLs and position recall.
- **Infinite scroll:** ideal for discovery feeds but hides footers and loses position.
- **Load-more:** a hybrid giving user control with scrolling momentum.

Match the pattern to user intent—browsing vs. finding specific items.

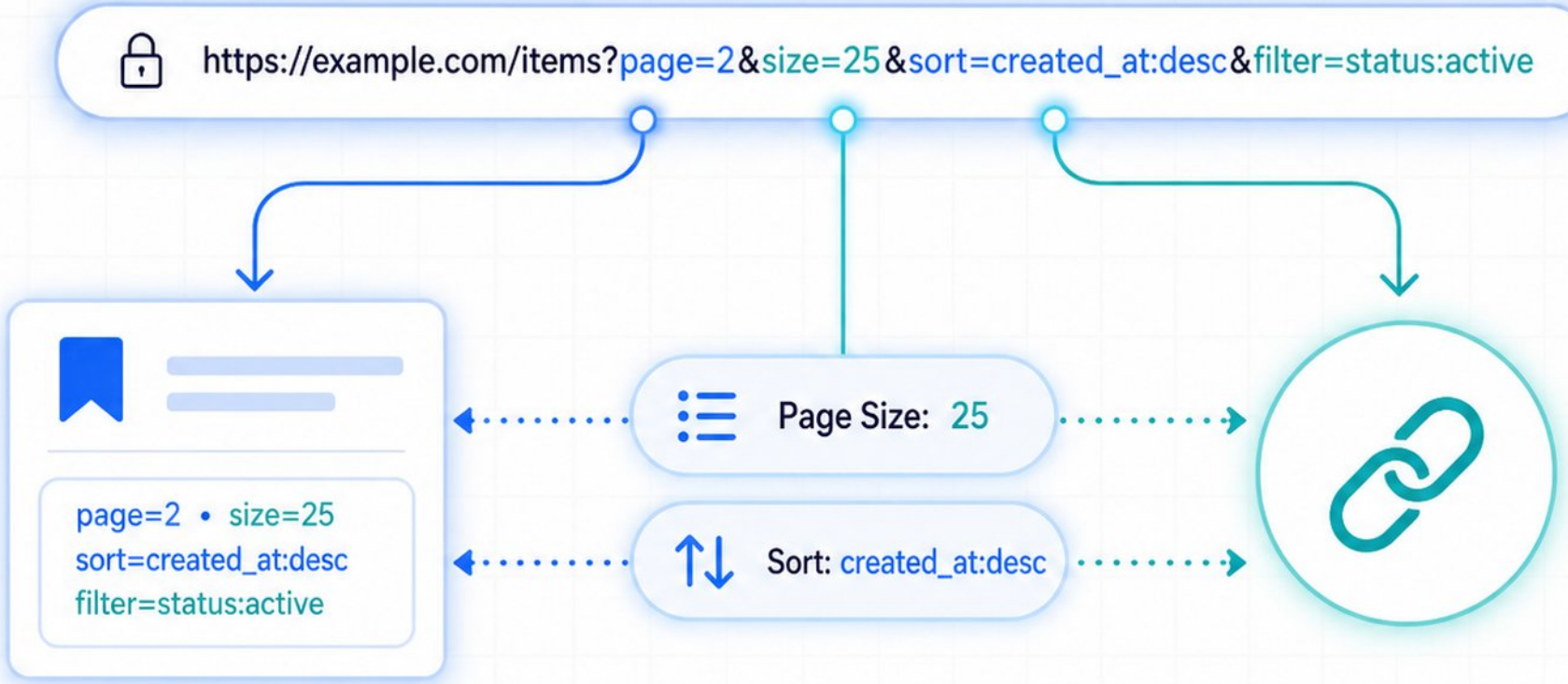
Pagination Control Patterns



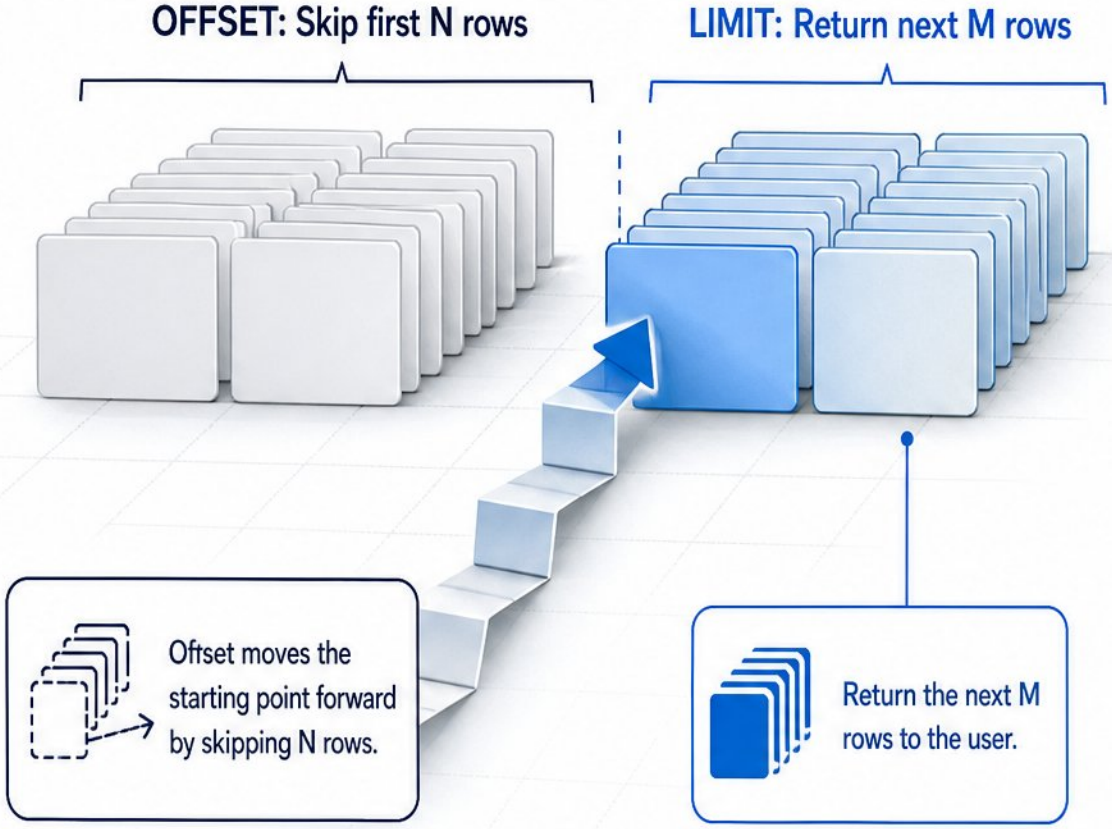
- Previous/Next buttons plus numbered page links for structured navigation
- Ellipsis truncation (e.g., 1 … 4 5 6 … 12) keeps long page ranges scannable
- Display First/Last links when truncated; omit if first/last numbers already visible
- Disable boundary controls (Prev on page 1) but keep them visible—never hide
- Page-size selectors: offer clear tiers (25, 50, 100) rather than excessive choices

URL Structure and Shareable State

- Encode page, size, and sort in the URL for direct deep-linking
- Preserve filter-and-page combos so users can bookmark and share exact views
- Use crawlable paginated URLs with canonical tags to avoid duplicate content
- Remember user choices (page size, sort) across navigation without resetting

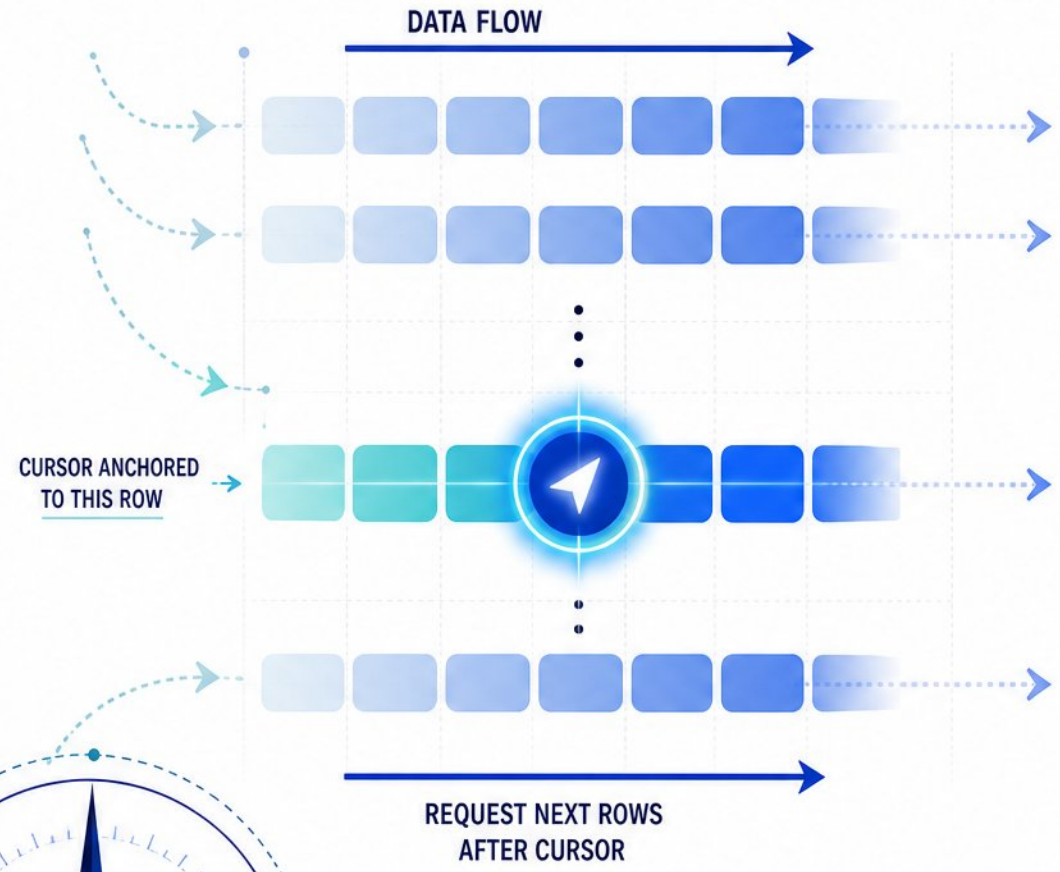


Offset Pagination in Practice



- Skip N rows via OFFSET and return next M rows with LIMIT
- Simple to implement with intuitive page-number math (page × size)
- Performance degrades linearly: deep pages scan and discard all prior rows
- Insertions or deletions mid-session cause duplicate or missing records
- Best for small, static datasets, admin dashboards, or when users need 'jump to page 47'

Cursor-Based Pagination for Real-Time Data



- Uses a unique, ordered cursor to request rows after a specific item
- Constant-time performance regardless of page depth, unlike offset's O(N) degradation
- Solves shifting-data problem: inserts and deletes cause no duplicates or skips
- Cursor anchors to data, not position, ensuring consistent results
- Trade-offs: no "jump to page N," requires composite unique tiebreaker, sequential access only

Choosing Between Offset and Cursor Pagination

Decision framework: data mutability, scale, and user navigation needs

Path 1
Stable & Orderly

Path 2
Continuous & Scalable

OFFSET-BASED PAGINATION

1 — 2 — 3 — • • • — N

Page-jump navigation supported

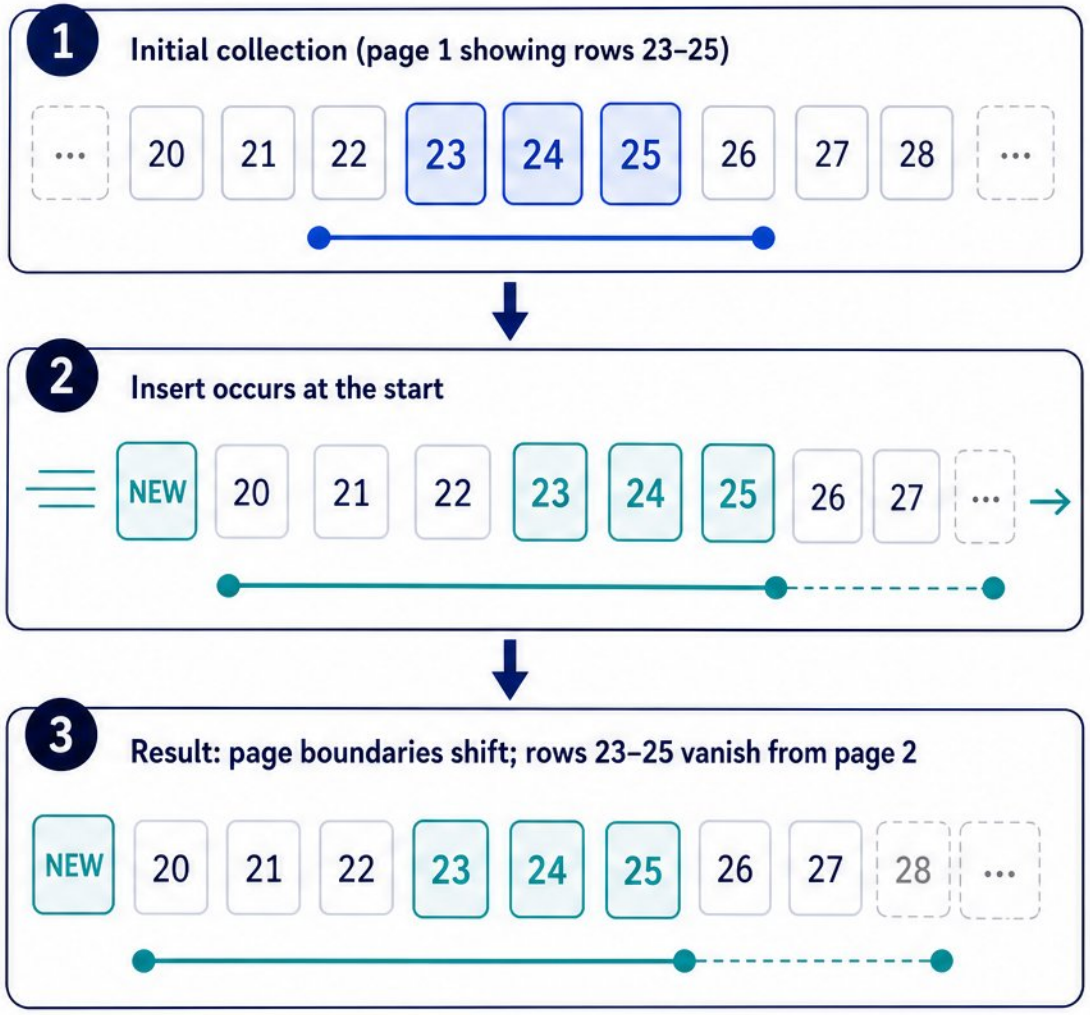
CURSOR-BASED PAGINATION

Sequential navigation (forward/backward)

- **Offset is acceptable for:** small datasets (<10K rows), admin panels, static catalogs, and when page-jump navigation is required
- **Cursor is required for:** large, growing, or frequently mutated datasets; real-time feeds; infinite scroll UIs; and high-traffic public APIs
- **Hybrid approaches:** use offset for internal dashboards, cursor for customer-facing and real-time endpoints

Managing Changing Collections

- Inserts or deletes mid-pagination cause shifting indices and silent data loss
- Offset example: rows 23-25 on page 1 become 26-28 and vanish from page 2
- Cursor-based resilience: anchor to a specific item's key, not a numeric position
- New rows don't disrupt the cursor; deletions don't shift the starting point
- Notify users with refresh indicators when results may be stale

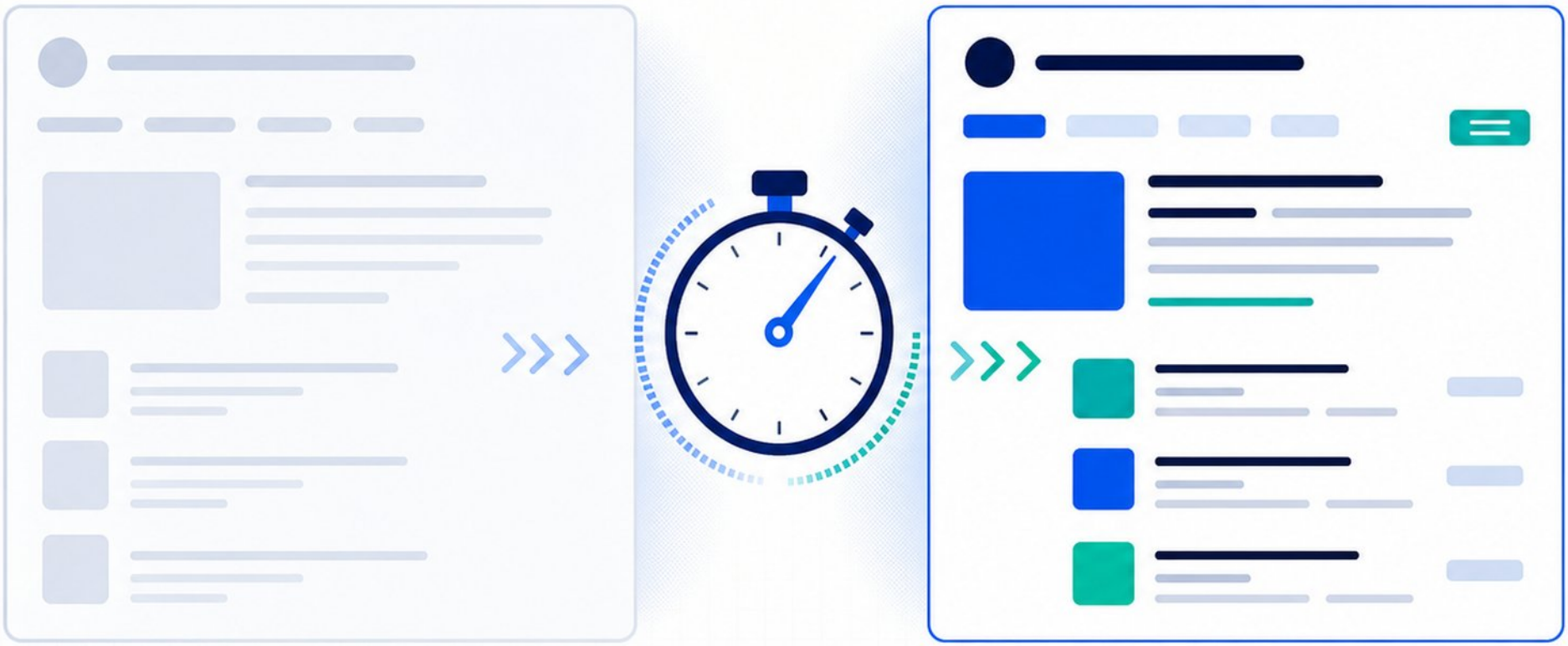


Loading States and Perceived Performance

- Match skeleton screens to >300ms waits; use spinners for sub-300ms transitions
- Prefetch adjacent pages via link rel=prefetch or service workers for instant feels
- Reflect page-size and filter changes immediately with inline result count updates

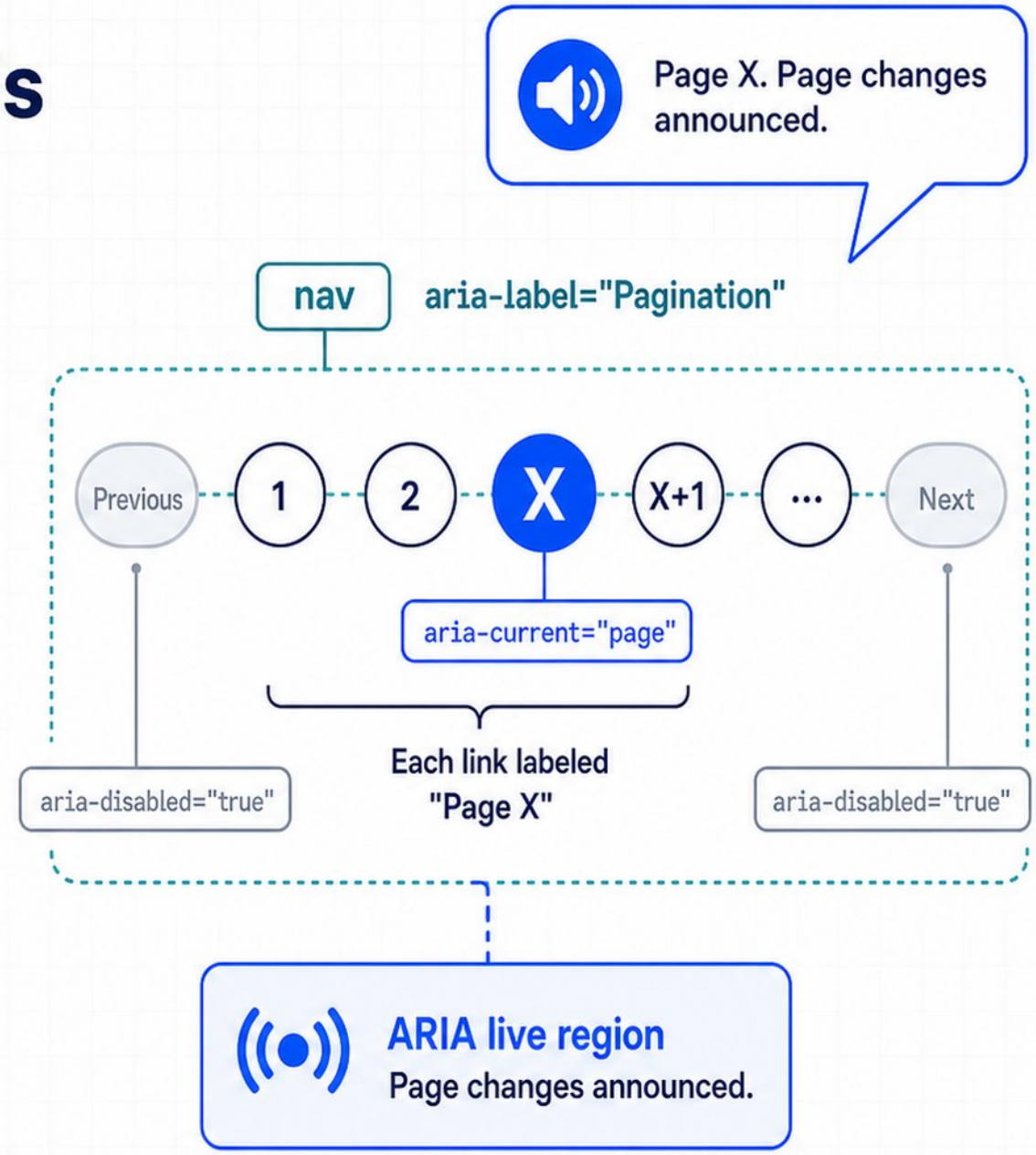
---

Target: initiate change <100ms, load start <200ms, full render <1 second



Accessibility for Pagination Controls

1. Wrap controls in a <nav> with aria-label="Pagination" to create a distinguishable landmark.
2. Use an unordered list structure so screen readers announce the number of available pages.
3. Mark the current page with aria-current="page" and label each link with "Page X".
4. Keep Previous/Next controls visible but use aria-disabled="true" at boundary pages.
5. For dynamic content loads, use an ARIA live region to announce page changes.

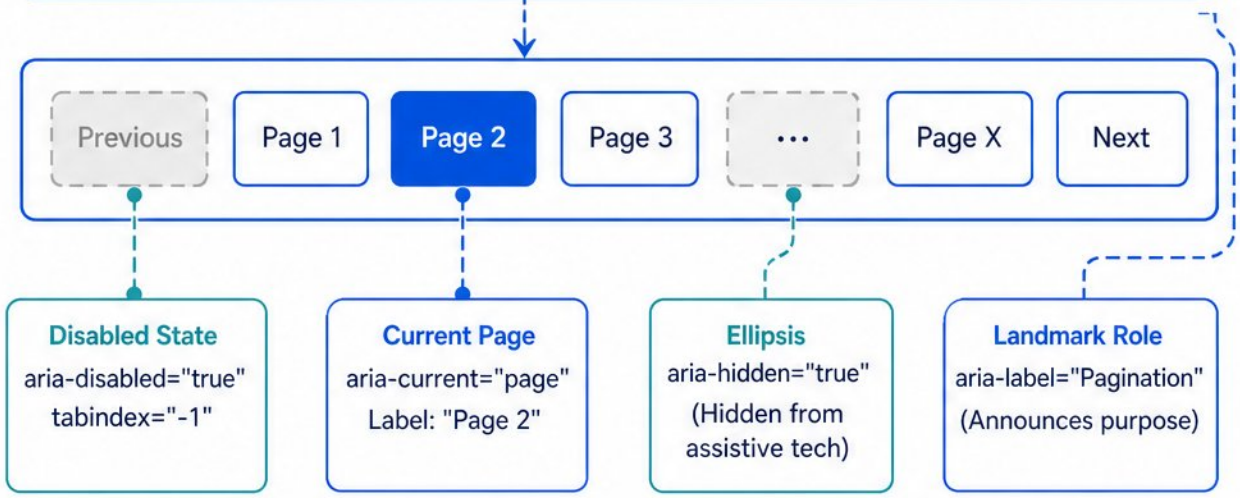


Accessible Pagination: Structure and Best Practices

- Wrap page links in an unordered list so screen readers announce item count.
- Use aria-current="page" on the active link; label links as "Page X".
- Handle disabled states with aria-disabled="true" and tabindex="-1".
- Hide ellipsis from assistive technology using aria-hidden="true".
- Tested patterns from USWDS, Vanilla, and CMS Design System.

STRUCTURE: UNORDERED LIST

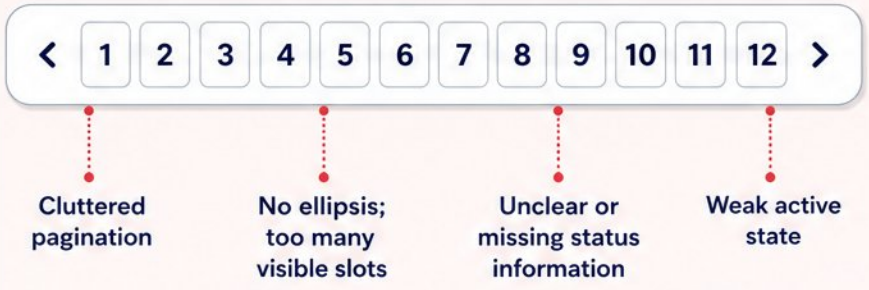
```
<nav aria-label="Pagination">
  <ul class="pagination">
    <li><a href="#" aria-disabled="true" tabindex="-1">Previous</a></li>
    <li><a href="#">Page 1</a></li>
    <li><a href="#" aria-current="page">Page 2</a></li>
    <li><a href="#">Page 3</a></li>
    <li><span class="ellipsis" aria-hidden="true">…</span></li>
    <li><a href="#">Page X</a></li>
    <li><a href="#">Next</a></li>
  </ul>
</nav>
```



Common Mistakes and How to Avoid Them

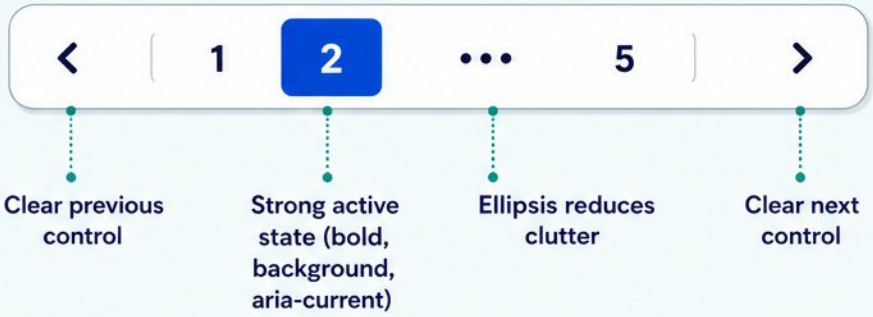
COMMON MISTAKES

Cluttered, confusing, and inaccessible



BETTER APPROACH

Clean, clear, and accessible



Cluttered pagination: use ellipsis, limit to 5–7 visible slots	Show 5–7 slots with ellipsis for a cleaner experience
Buried or ambiguous page-size labels confuse users	Use clear, visible page-size labels with context
Missing “Showing 41–60 of 247 results” or active-page distinction	Always show result range and clear active-page state
Broken back-button: mismatched SPA history and page state	Sync SPA history with page state for reliable back navigation
Relying on color alone for active state—combine bold, background, and aria-current	Use multiple cues and aria-current for accessibility

Practical Design Checklist for Pagination

- **Verify mental model clarity:** show 'Page X of Y,' result range, and that filters persist across navigation.

- **Audit control visibility and states:** prev/next, numbered links, ellipsis, page-size selector, and disabled edges.

- **Confirm implementation stability:** shareable URLs, cursor tiebreakers for real-time data, and canonical tags.

- **Run an accessibility audit:** <nav> landmark, aria-current, link labels, keyboard flow, and live region updates.

- **Usability-test key scenarios:** position recovery, preserved filter state, and user understanding of collection scope.

Key Takeaways

Pagination helps users orient:
"Where am I, and where can I go next?"

Match strategy to task:
pagination for search, infinite scroll for browsing, load-more for balanced control

Offset for small/static data with page-jump; cursor-based for large/ dynamic data to ensure consistency and performance

Accessibility is foundational:
semantic HTML, ARIA attributes, keyboard support, clear labeling are mandatory

Test with real users:
can they return, share a link, and understand remaining scope?

PersonWise.

PersonWise • AI digital-human course platform

Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/5895c5f6/public