Understanding Software Errors: Detection, Messages, and Recovery

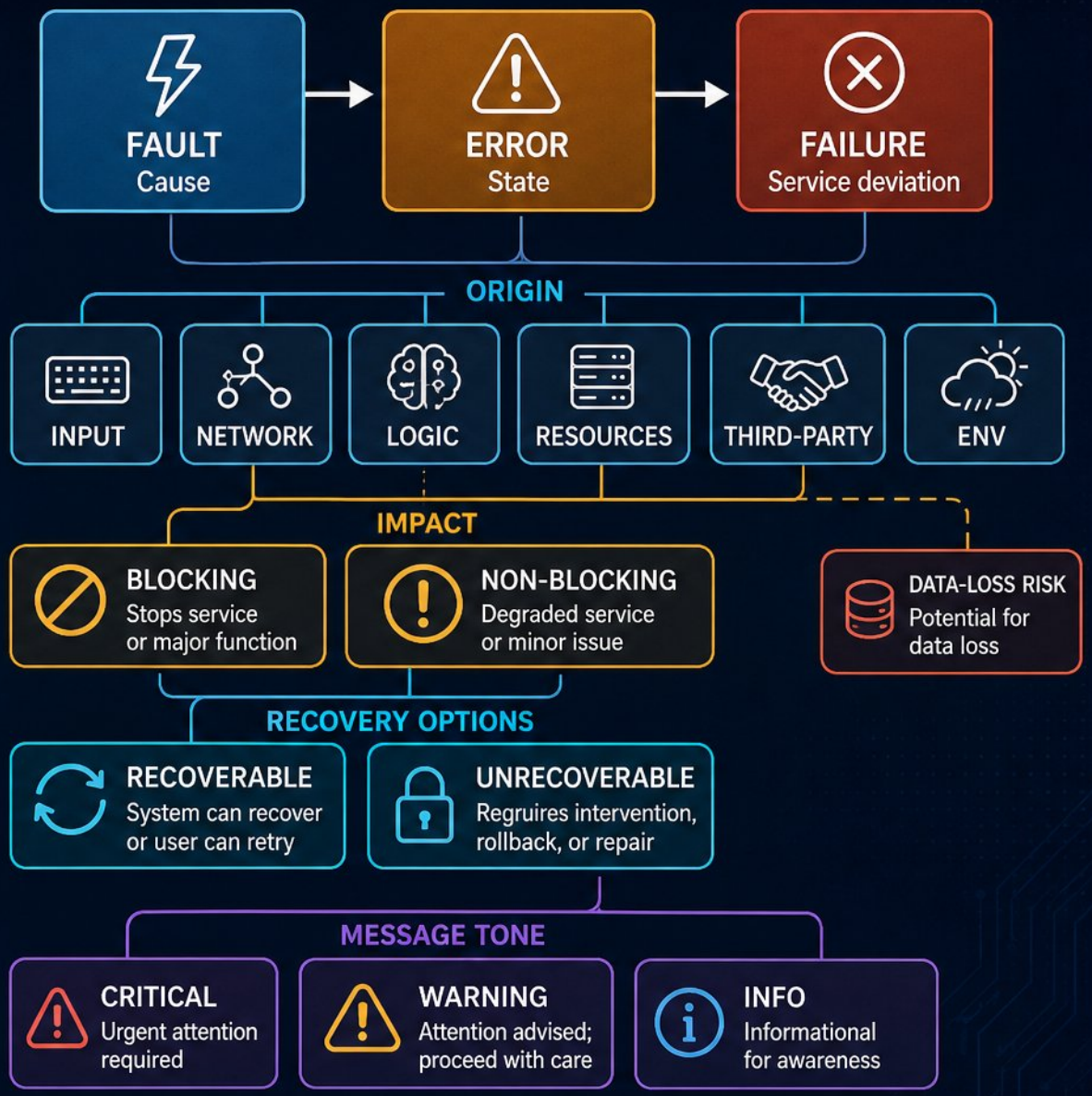
- Software errors disrupt workflows and require structured handling
- Three roles: failure signals → user messages → recovery paths
- Detection finds issues; messages inform users clearly
- Recovery paths give users actionable next steps
- Training covers taxonomy, design, practice, and monitoring

The Error Lifecycle: A Three-Role Framework

- **Failure Signal:** internal detection of a problem
- **User-Visible Message:** clear, safe communication to the user
- **Recovery Path:** actionable next step the user can take
- Detection → Safe Message → Guided Recovery forms a pipeline
- Disconnect causes blame without guidance or dead recovery buttons

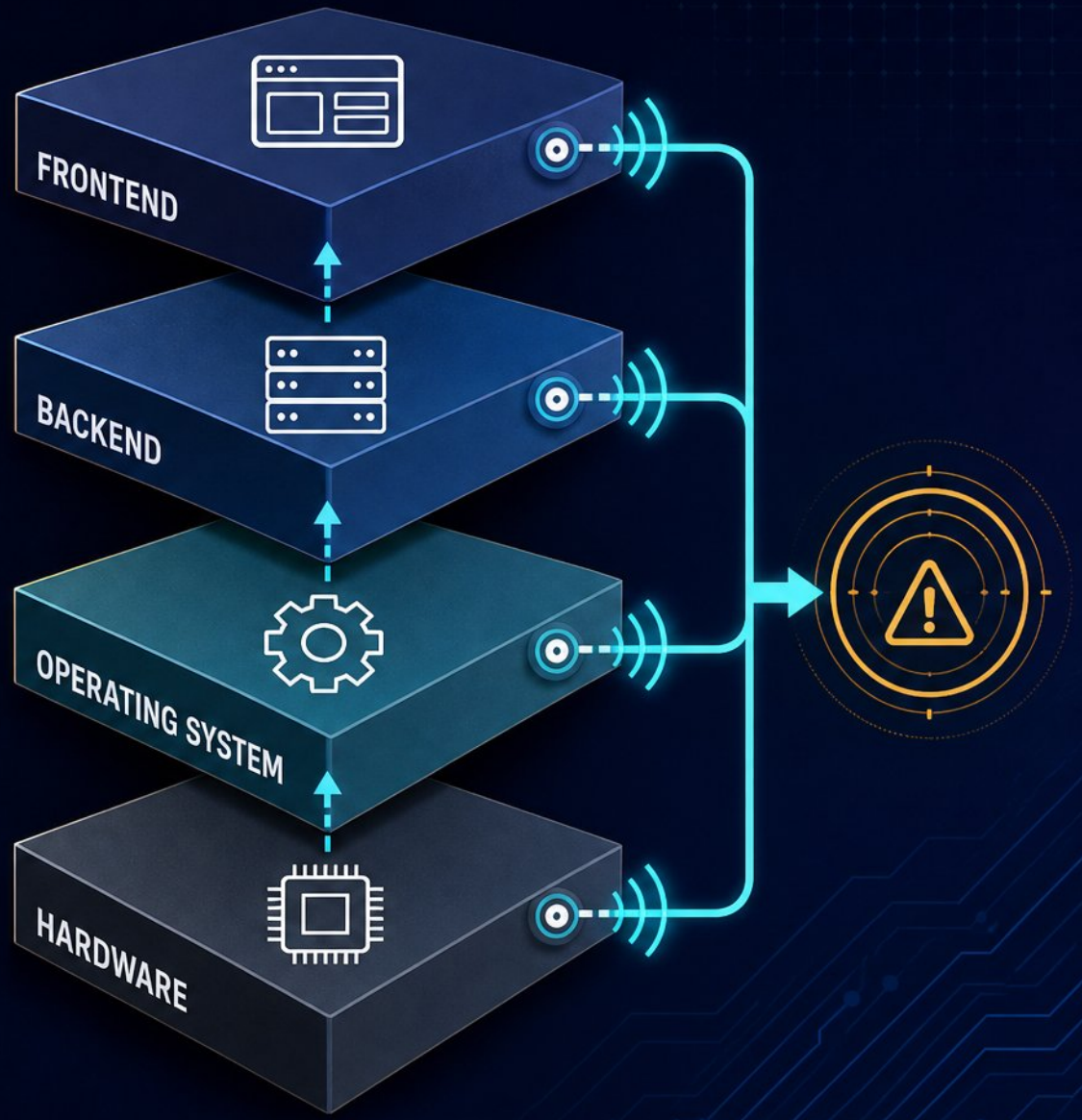
Taxonomy of Software Errors

- **Fault, Error, Failure:** cause, state, and service deviation
- **Origin:** input, network, logic, resources, third-party, env
- **Impact:** blocking vs. non-blocking and data-loss risk
- **Recoverable** vs. unrecoverable shapes recovery options
- Match message tone to impact: critical, warning, or info



How Systems Detect Errors

- Assertions, exceptions, return codes flag immediate failures
- Health checks, timeouts, watchdogs catch stalled components
- Detection lives everywhere: frontend, backend, OS, hardware
- Silent failures and swallowed exceptions corrupt data quietly



Failure Signals: From Raw Error to Actionable Trigger

- **Failure signal:** internal flag indicating an error (stack trace, error code, HTTP status)
- **Signal routing:** transforms internal data for the user without leaking sensitive internals
- **Transmit only** safe structured objects or codes to the message generation layer
- **OWASP risk:** exposing raw signals enables information leakage and attacker reconnaissance
- **Strip** stack traces, file paths, and debug data before user-facing communication

Crafting User-Visible Error Messages: Principles

- Core role: inform, guide, and reduce anxiety—never blame the user
- Structure every message with the Avoid, Explain, Resolve framework
- Use plain language; avoid jargon and internal diagnostics
- Lead with the user's next action, not system details

CLEAR & SUPPORTIVE

We can't process your request right now.

This can happen if a required field is missing or the information isn't in the right format.

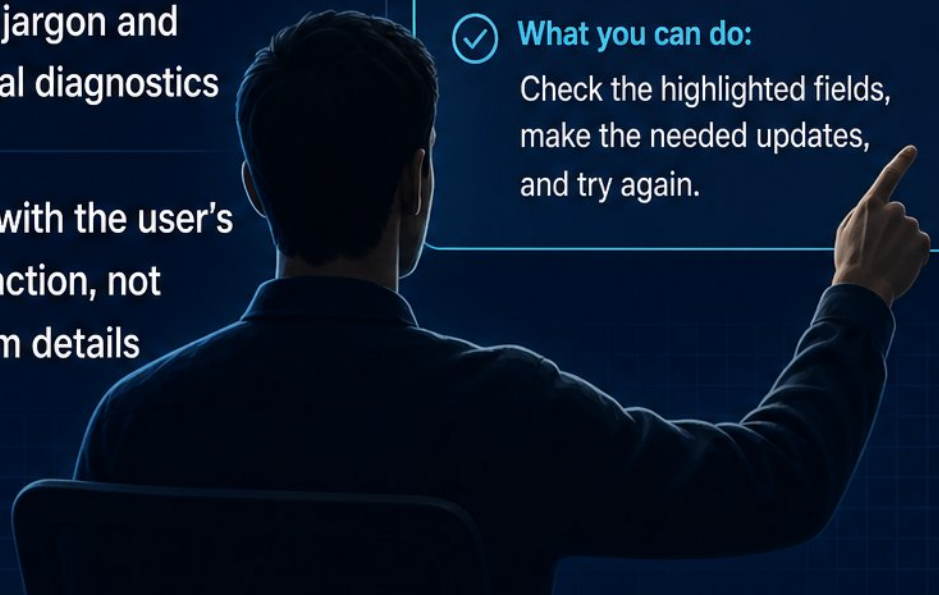
What you can do:
Check the highlighted fields, make the needed updates, and try again.

BLAMING & CONFUSING

Error: Invalid input detected.

Your submission failed because the system cannot process the data you entered.

What you did wrong:
Incorrect data was entered. Please fix your input and resubmit.

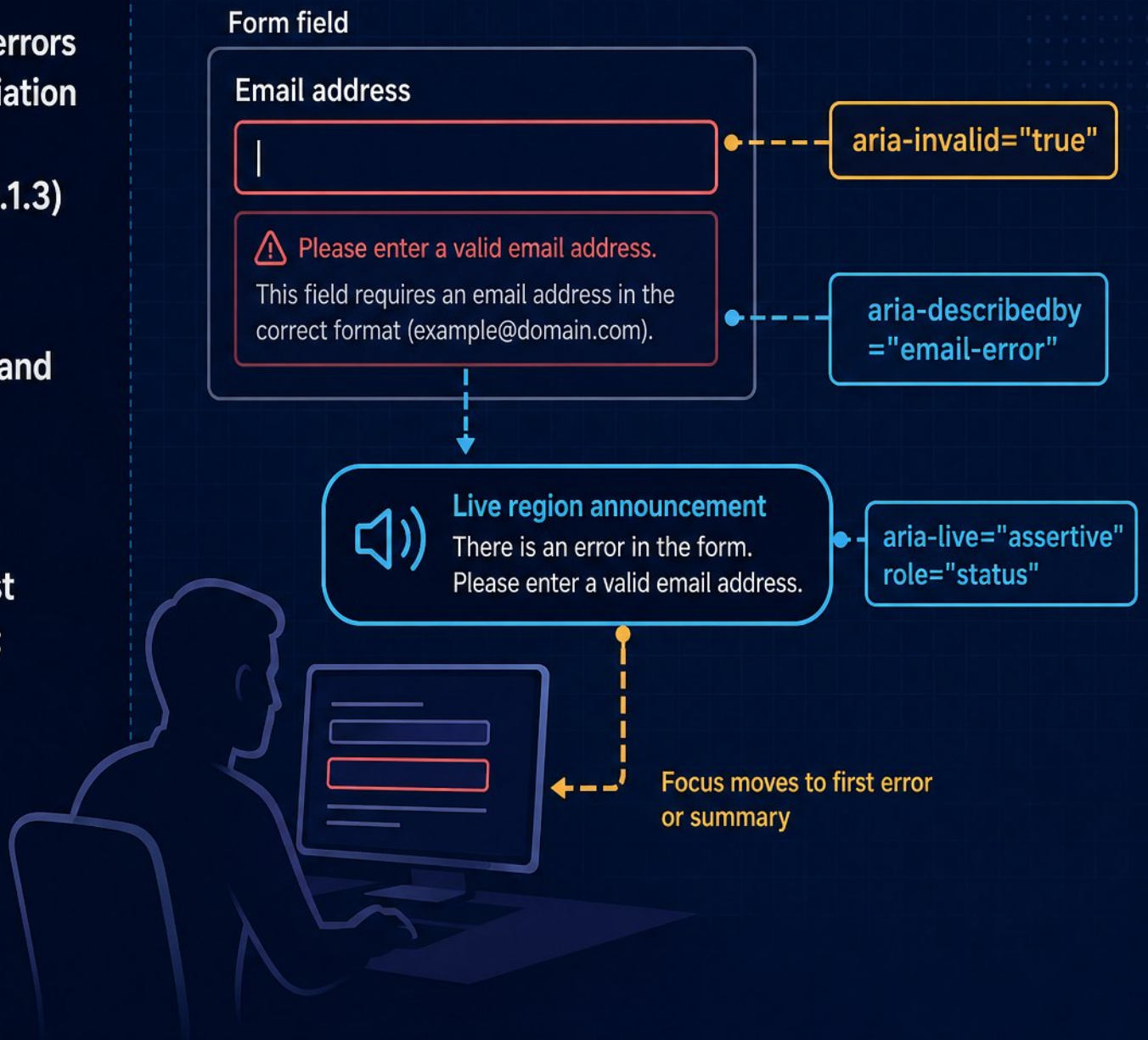


Crafting User-Visible Error Messages: Practice

- Transform 'Error 403' into *'You don't have access to this report'*
- Replace 'Invalid input' with *'Please enter a valid email address'*
- Network failure: *'Check your connection and try again'*
- Permission issue: *'Contact your admin to request access'*
- Validate inline and on-submit; avoid premature on-blur errors

Accessibility in Error Communication

- Meet WCAG: text errors (3.3.1), field association (4.1.2), status announcements (4.1.3)
- Use aria-invalid, aria-describedby, and live regions for screen readers
- Move focus to first error or summary; never rely on color alone

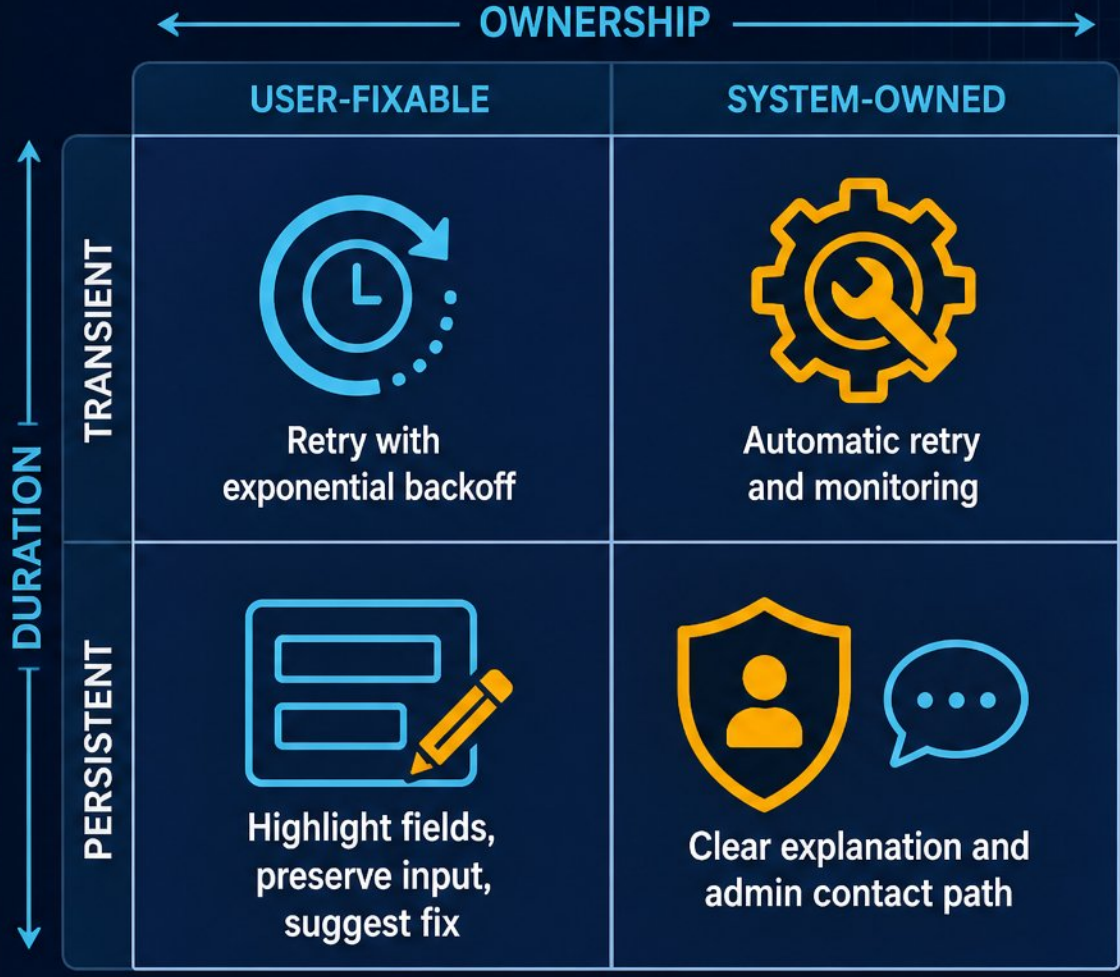


Recovery Paths:
Giving Users a Way Forward

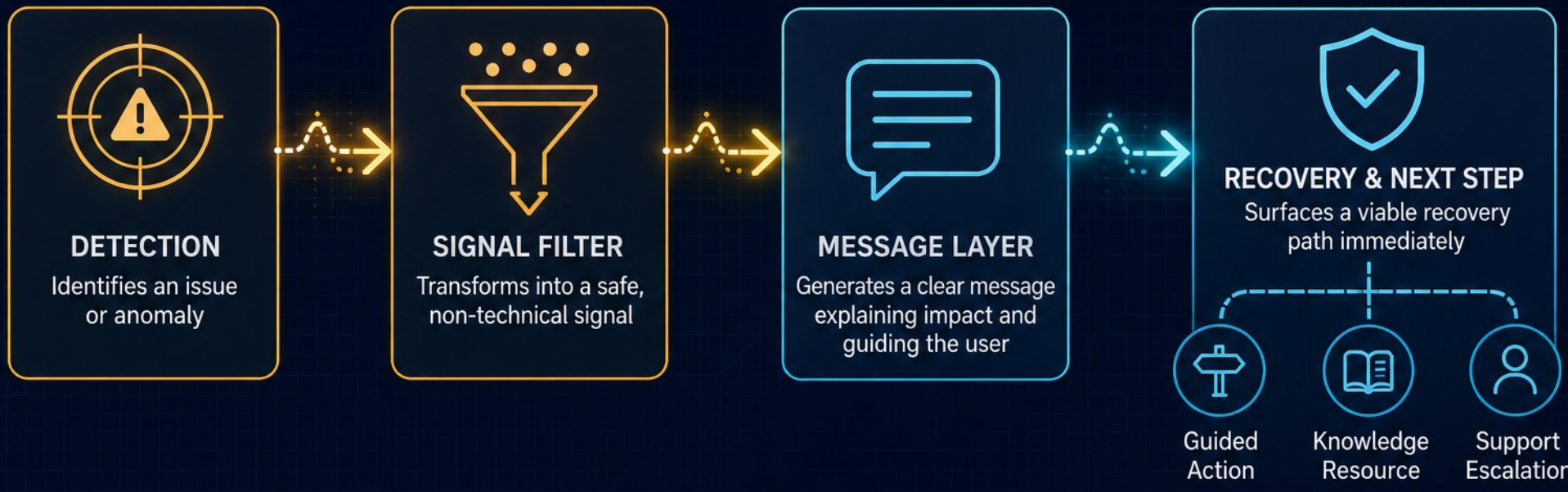
- Direct action after error: retry, undo, fallback, workaround, or contact support
- Use idempotent retries, graceful degradation, and partial success patterns
- Always preserve user input — recovery should never mean starting over
- Missing recovery paths lead to churn, support overload, and data loss

Mapping Errors to Recovery Strategies

- Transient network issues → Retry with exponential backoff
- Validation errors → Highlight fields, preserve input, suggest fix
- Permission failures → Clear explanation and admin contact path
- Match delivery to severity: inline for fields, modal for blocking
- Use a decision matrix: transient/persistent, user-fixable/system-owned



Connecting Detection, Messages, and Recovery in a Workflow

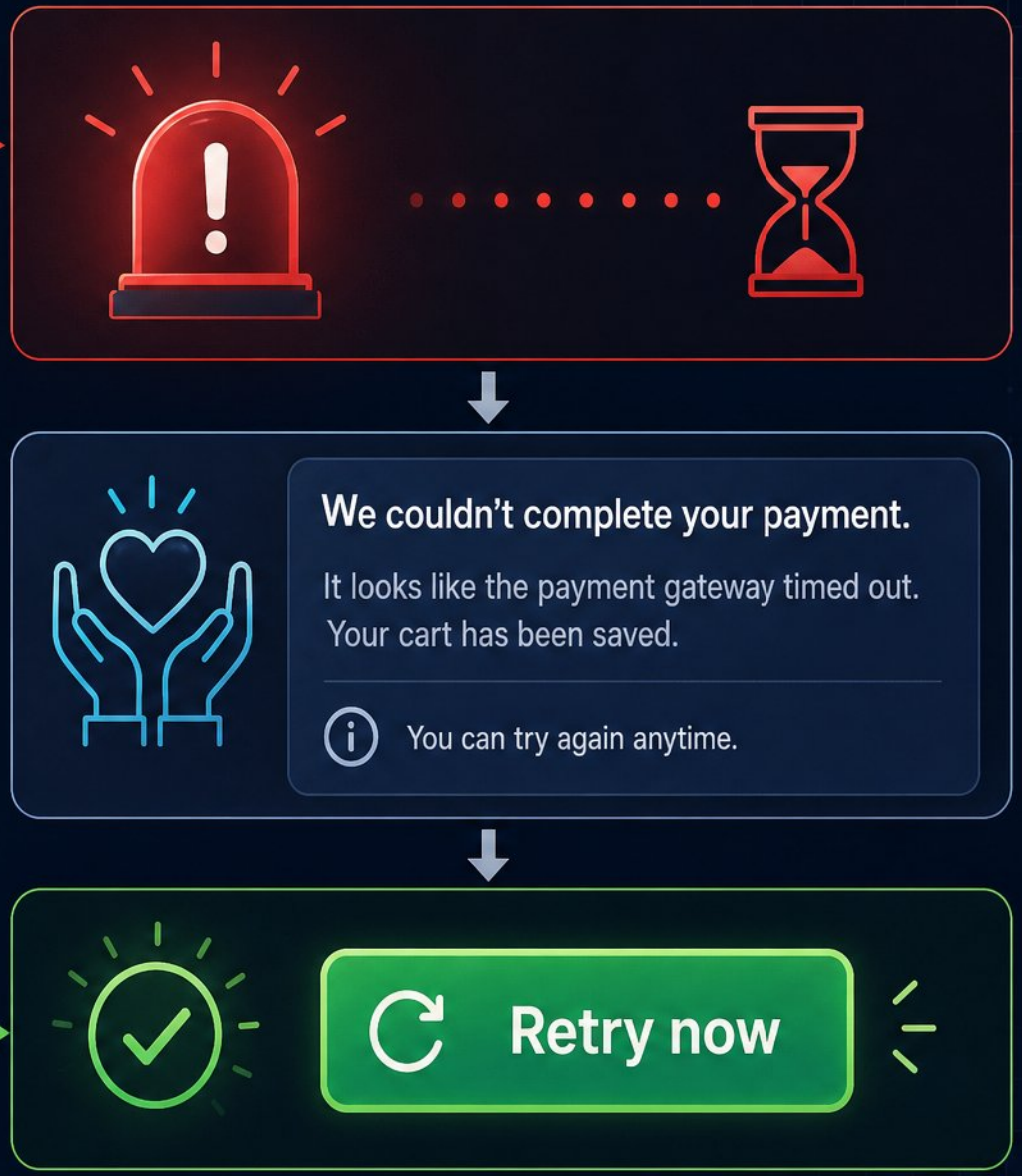


- Detection transforms into a safe, non-technical signal
- Signal routed to message layer without leaking internals
- Clear message explains impact and guides the user
- Viable recovery path is surfaced immediately
- Disconnects: dead buttons, invisible failures, useless text

DISCONNECTS TO AVOID

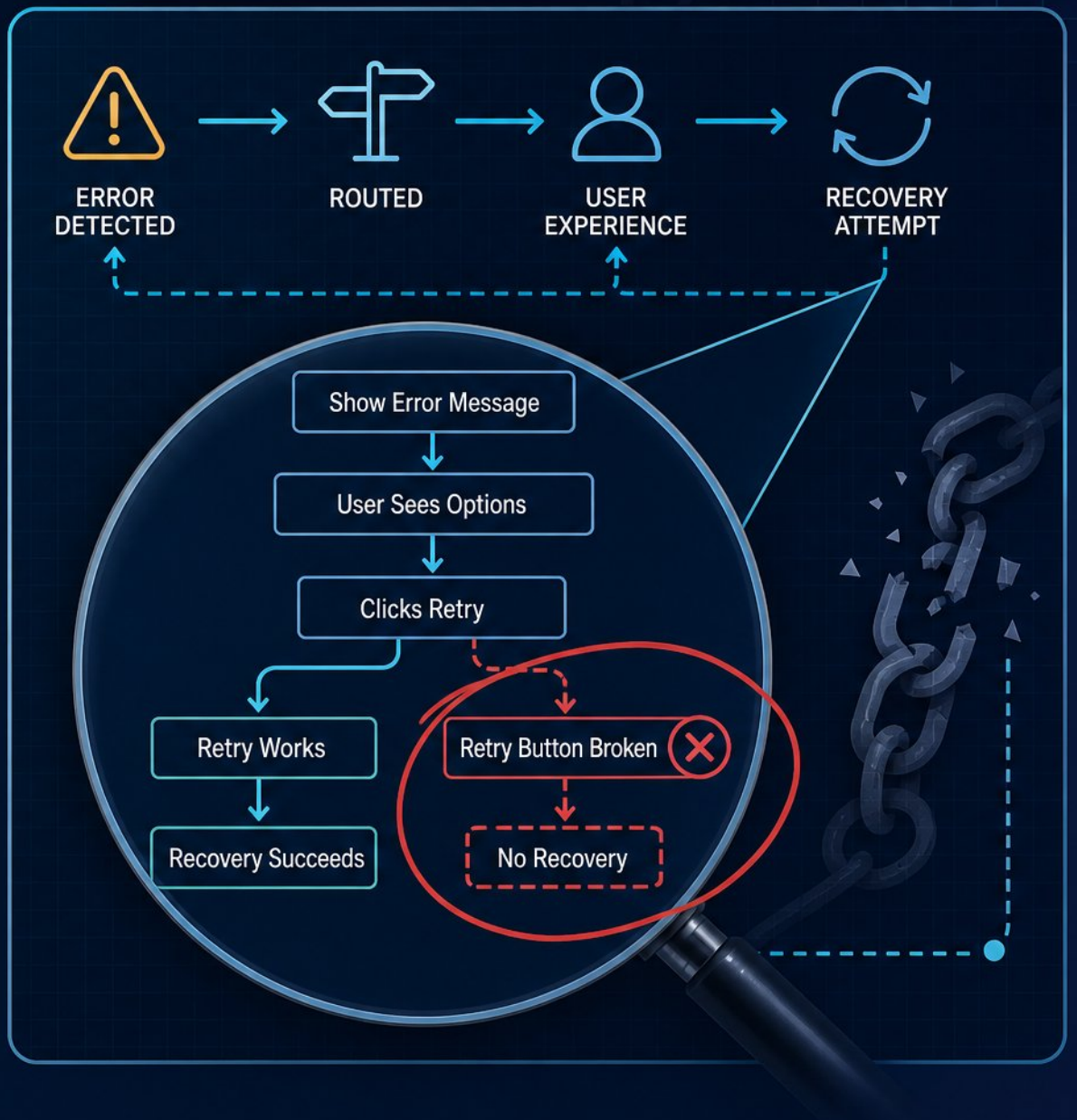
Practical Examples Across Platforms

- Login failure: detect 401, message "Check your details", offer password reset link
- Payment timeout: detect gateway timeout, preserve cart, surface "Retry now" button
- File upload rejection: validate type/size on client, explain limits, suggest alternatives
- GPS unavailable: detect null location, show fallback search, explain permission step
- API rate limiting: detect 429, display "Too many requests", show retry countdown timer
- Broken flows: avoid generic "Error occurred" and dead buttons with no next step

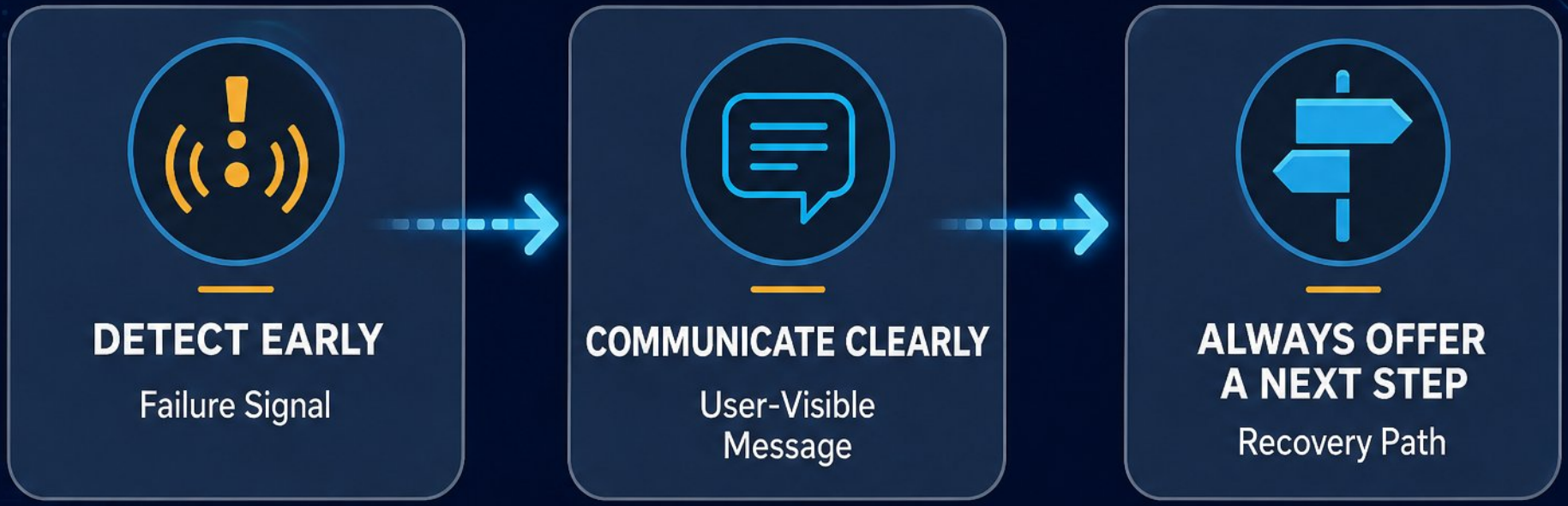


Testing and Monitoring the Error Experience

- Rigorously test error paths through detection, routing, and usability checks
- Log failure signals and track error-to-recovery conversion rates
- Set alerts for broken recovery paths like dead Retry buttons
- Refine messages and recovery using RUM data, tickets, and session recordings
- Continuously improve with real-user behavior and support feedback



Summary, Design Principles, and Action Checklist



- Recap the three roles: Failure Signal, User-Visible Message, Recovery Path
- Detect early, communicate clearly, and preserve the user's work
- Always offer a clear next step; never blame the user
- Checklist: Is the signal safe? Is the message specific and plain?
- Does the error flow include a clear, working recovery action?

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/4ad097bc/public