Real-world use cases (Root first once, then leaf→up excluding Root)

Pattern summary

1. Evaluate Root (depth 0) alone for a global/preemptive transition.
2. If none, scan from current leaf upward stopping before Root (so Root logic does not run twice).
3. First match wins (leaf-first specificity preserved after global check).

Why do this

* Global safety / E-STOP: Root can immediately divert to a safe state without waiting for deep states.
* System mode gate: Root enforces high-level mode (e.g., MAINTENANCE, LOW\_POWER) before feature-level transitions.
* Central resource arbitration: Root checks system-wide budget (power, thermal, memory) and triggers throttling transitions.
* Licensing / authorization: Root prevents unauthorized functional transitions early.
* Global scheduling window: Root can suspend or resume whole machine (e.g., timed blackout) while leaving local logic untouched otherwise.
* Fault aggregation: Root maps any latched global fault flags to a transition irrespective of current deep state.
* Startup / shutdown sequencing: Root can override deep initialization chatter with a higher-level transition (e.g., ABORT\_STARTUP).

Advantages over pure root-first

* Keeps conventional UML leaf-first behavior for ordinary transitions.
* Minimizes repeated evaluation of expensive root guards (once per cycle).
* Reduces risk of root shadowing nuanced local recovery transitions unless truly necessary.

When not to use

* If multiple ancestor layers (not just root) need preemptive priority—then use a configurable depth window.
* If root transitions depend on side-effects of deep exit/entry (those won’t have happened yet).
* If global and local transitions must be arbitrated by priority values rather than fixed order.

Design cautions

* Ensure root global conditions are stable for the whole cycle (avoid flapping causing starvation).
* If a root-triggered target lies inside current branch below root, clarify whether partial exits occur or full re-entry.
* Log which layer produced the transition for diagnostics.

Key published / de‑facto references for hierarchical state machine (HSM) event / transition dispatch behavior:

Primary theoretical / standards sources

* David Harel, “Statecharts: A Visual Formalism for Complex Systems,” Science of Computer Programming, 1987. (Original semantics: event offered to innermost (leaf) active state; bubbles up if unhandled.)
* OMG UML Specification (e.g., UML 2.5 / 2.6 draft) – Run‑to‑completion event dispatch: trigger searched from deepest active state toward root; first matching transition (guard true) fires.
* W3C SCXML Recommendation – Algorithm 3 (event step): transitions are selected with candidate evaluation starting at innermost states, then ancestors.

Industry / framework implementations (document similar leaf‑first, plus patterns for global/root overrides)

* Quantum Leaps QP Framework (Miro Samek) – Active object HSM; explicit description of hierarchical event dispatch (leaf-first) and “top state” global catch / safety transitions.
* Boost.Statechart (C++ Boost library) – Event dispatch travels from most-derived (deepest) state outward; allows outer states for global reactions.
* SMC (State Machine Compiler) HSM mode & Yakindu Statechart Tools – UML-conformant leaf-first dispatch; supports “top-level” transitions for global modes.
* MATLAB/Simulink Stateflow – Follows state hierarchy; leaf-first for events; recommends top-level transitions for global mode / fault / emergency.
* AUTOSAR SW-Cs using HSM patterns (e.g., BSW mode management) – Often implement a root-level mode arbiter with preemptive transitions (design guidelines, not a single spec line).
* SCION / scxml libraries – Mirror SCXML spec; root-level transitions used for global cancellation / abort.

Common pattern justification for “Root-first (single pass) then leaf-up (excluding root)”

* Safety / emergency stop (root transition preempts).
* Global mode gating (POWER\_OFF, MAINTENANCE) before local progression.
* System-wide fault aggregation (root maps aggregated flags to fail-safe).
* Resource / thermal governor at root before local transitions incur cost.
* Licensing / authorization gate (root denies disallowed feature transitions).
* Coordinated shutdown / reset (root override regardless of deep state).

Terminology sometimes used

* Preemptive (or priority) superstate transitions.
* Global transition layer.
* Top-level guard / arbiter.
* Supervisor state.

Design notes aligning with references

* Matches UML leaf-first principle after (optional) singular root preemption check (does not violate UML if root transition considered just another enabled transition evaluated earlier by implementation-defined priority).
* Similar to QP “top state” pattern where a global emergency transition can be taken before standard bubbling.

Search keywords (to locate detailed rationale)

* “UML state machine event dispatch innermost state”
* “Harel statechart event handling algorithm”
* “QP framework top state emergency stop”
* “Stateflow global transition fault handling”
* “SCXML algorithm transition selection hierarchy”

Concise citation pointers (publicly accessible)

* UML 2.5 Spec: Section “Behavior of State Machines” (event dispatch).
* W3C SCXML: Section “Algorithm for SCXML Interpretation.”
* Harel 1987 Paper: Sections describing event propagation / superstate hierarchy.
* Quantum Leaps Docs: “Hierarchical Event Processing” chapter.

Summary Leaf-first (deepest to root) is the normative (UML / Harel / SCXML) dispatch order. Adding a single preliminary root check is a pragmatic extension widely used for safety, global mode, and resource arbitration without undermining standard hierarchical specificity.

Below is a concise requirements / expectation set for adopting the Root‑first (single check) then Leaf→Root(exclusive) processing style, mapped to conventional HSM semantics (Harel / UML run‑to‑completion / QP / SCXML), and applied to two concerns: (A) continuous (loop/do) state actions and (B) event inputs.

1. General dispatch model

* Canonical (UML / Harel): event (or implicit “output” transition scan) searches from deepest active state upward; first enabled transition fires; then run-to-completion (RTC) finishes before next event.
* Your Root-first variant: add a single, early, preemptive evaluation point at Root (depth 0) before normal leaf-up bubbling (excluding Root on second pass) to host global overrides (fault, mode, kill switch) without double evaluation.

1. Output / spontaneous transition phase (your \_StateMachine\_ProcBranchRoot) Requirements / expectations:

* Root transition evaluation happens at most once per cycle (no duplicate call later in the same cycle).
* If Root yields a transition, deeper states’ transitions (and their loop/do actions for that cycle) are skipped (preemptive semantics).
* If Root does not transition, traverse leaf→parent until (but not including) Root.
* stopLevel=1 path guarantees Root’s TransitionOfOutput code is not re-entered inadvertently.
* Must ensure state depth metadata remains consistent; Root-first logic assumes a single well-formed chain.

1. Continuous actions (loop / do behavior) Decide and document order; recommended invariant set:

Option A (common in safety-critical preemption): Phase 1: Evaluate transitions (Root preemptive, then leaf→up). Phase 2: If no transition taken, execute continuous actions. Ordering inside Phase 2: a) Either Root→Leaf (broad to specific) or b) Leaf→Root (specific first). Recommendation: Root→Leaf if higher layers provide supervisory gating values used by lower loops in same cycle; Leaf→Root if lower levels produce data aggregated upward. Pick one and keep consistent.

Option B (less safe for preemption): run loop actions first, then transitions. Avoid if Root is for emergency stop; could delay reaction by one cycle.

Explicit requirements:

* A state’s loop/do action MUST NOT run in a cycle where that state is exited (UML expectation).
* After a transition, new state entry actions run before any loop actions of the newly active leaf path (run-to-completion).
* Root loop action executes at most once per cycle, even with Root-first enabled.
* Provide a flag (e.g., rootEvaluatedThisCycle) so generic branch processors don’t re-run root logic.

1. Interaction between Root and subordinate loops

* If Root maintains global aggregates (fault latch, power budget), subordinate loop guards can rely on stable values within the cycle (Root loop executes before subordinate loops if using Root→Leaf ordering).
* If loops may raise internal triggers (convert conditions into deferred synthetic inputs), they should enqueue them; do not directly call transition traversal mid-loop (preserves RTC determinism).

1. Event inputs (synchronous vs asynchronous) Current code: synchronous mask (bitfield) + async single pending pointer. Requirements for Root-first input processing:

* For each input event (id,value):
  1. Root input transition attempt (State\_TransitionOfInput\_AsTop).
  2. If no Root transition, traverse leaf→up excluding Root.
* Consumption semantics: If a transition is taken anywhere, event is considered consumed (do not re-offer upwards).
* If Root takes a transition, do not attempt subordinate processing for that event.
* For synchronous batch (\_StateMachine\_ProcBranchSyncInput): Ensure ordering (e.g., lowest id first) is stable; Root-first policy must be applied uniformly per event.
* For asynchronous (buffered) next-state (p\_SyncNextSubState): Apply Root-first policy again prior to committing buffered leaf transition; if Root preempts, discard or revalidate the buffered target (document the policy).
* Starvation prevention: If Root frequently preempts, low-level events may starve. Mitigation options: a) Count consecutive Root preemptions; after N cycles, log or escalate. b) Allow certain input classes to bypass Root preemption (whitelist). c) Time-slice: every M cycles skip Root preemption unless critical flag set.

1. Guard / side-effect purity

* Root transition functions should be (ideally) free of irreversible side-effects until a transition decision is made (pure guard style). This avoids inconsistent state if a later design adds additional preemptive layers.
* Loop actions should not directly mutate guard conditions retroactively evaluated earlier in the same cycle (define evaluation-before-action boundary).

1. Ordering contract (proposed canonical cycle with Root-first) Cycle steps (no transition taken):
   1. Apply pending async transition (once).
   2. Process synchronous inputs: For each pending input: a) Root input transition check. b) If none, leaf→up (excl. Root). c) If transition taken: perform exit/entry sequence; cease further processing of that event; continue with remaining events on new active branch (RTC extension).
   3. Output (spontaneous) transitions: a) Root-first single check. b) If none, leaf→up (excl. Root).
   4. If still no transition: execute loop/do actions (ordered per design choice).
   5. Clear per-cycle flags.

If any transition occurs at steps 2 or 3: complete exit/entry sequence, then (optionally) run post-transition entry actions and skip loop actions for states exited.

1. Diagnostics & traceability

* Log which layer (Root vs depth N) produced transition to aid debugging of preemptive overrides.
* Provide counters: rootTransitionsTaken, suppressedLeafTransitions, starvedEvents.

1. Performance considerations

* Root-first adds only one extra call per cycle when no Root transition occurs (O(1)).
* Avoid rebuilding the full ancestor path twice; Root-first pass can cache leaf pointer or depth if needed.
* Ensure Root path evaluation does not allocate or traverse deep structures unnecessarily (keep O(1)).

1. Extensibility

* If later adding “priority ancestors” (e.g., depth 0..K preemptive window), generalize Root-first to a bounded top segment.
* Document invariants now so extension preserves run-to-completion determinism.

1. Conformance to references

* UML / Harel / SCXML principle (leaf bubbling) preserved after an implementation-defined priority override at Root (still valid; spec allows tool-defined prioritization as long as single transition selected per RTC step).
* Mirrors QP practice where a “top state” can host emergency transitions outside normal bubbling, yet ordinary events still follow inner-first logic.

1. Minimal code contract adjustments (conceptual, not full patch) Add per-cycle flags: bool rootOutputChecked; bool rootInputCheckedThisEvent; Reset at start of cycle; set after Root-phase to prevent accidental re-evaluation.
2. Testing scenarios

* Root preemption of local transition (ensure local guard true but not taken).
* Leaf transition when Root inactive.
* Transition triggered by input vs spontaneous; no double root check.
* Loop actions suppressed on exit.
* Buffered async state overridden by Root in same cycle.
* Starvation test: repeated Root transition; verify counters/logging.

Summary Root-first single-pass adds a deterministic, globally preemptive layer without discarding conventional leaf-first specificity. Clear separation of (1) transition selection phases and (2) loop action execution, plus strict single-evaluation of Root per cycle, ensures predictable run-to-completion semantics aligned with established HSM references. Define and enforce: one transition per RTC cycle; no loop action for exiting states; event consumption upon transition; root preemption cannot silently starve subordinate event handling without instrumentation.

**Root-first + leaf-up (exclusive) compact requirements**

1. Order per cycle  
   a) Apply pending async transition (if any)  
   b) Process input events (Root check once per event, else leaf→up excl. Root)  
   c) Spontaneous/output transitions (Root once, else leaf→up excl. Root)  
   d) If no transition taken: run continuous (loop/do) actions
2. Single evaluation
   * Root transition logic evaluated at most once per phase (input event / spontaneous) per cycle.
   * Exclude Root from second (leaf-up) pass.
3. Transition semantics
   * First enabled transition fires; run-to-completion; only one transition per event step.
   * Loop/do actions skipped for any state exited that cycle.
4. Continuous actions ordering (choose & document)
   * Recommended: Root→Leaf for supervisory data first OR Leaf→Root for bottom-up aggregation; remain consistent.
   * No loop action for states just entered until after entry actions complete.
5. Input handling
   * Event consumed where transition taken (no further bubbling).
   * Root preemption can starve leaf logic; track counters or add policy if needed.
6. Guards & actions
   * Root guards side-effect free until decision.
   * Loop actions must not alter earlier guard decisions in same cycle.
7. Data/flags
   * Per-cycle flags: rootOutputChecked, rootInputCheckedThisEvent.
   * Reset at cycle start.
8. Diagnostics
   * Log source depth of transition.
   * Counters: rootTransitions, suppressedLeafTransitions, starvedEvents.
9. Testing essentials
   * Root preempts valid leaf transition.
   * Leaf transition when Root idle.
   * No double Root evaluation.
   * Loop skipped on exit; executed when stable.
   * Async pending overridden by Root correctly.
10. Extensibility

* Can generalize Root-first to top-K depths later without breaking invariants.

Summary: Root-first adds a single global preemptive check while preserving standard leaf-first specificity, with strict one-pass evaluation and clear separation of transition vs continuous action phases.