Bombs in Converge: What, Why, and How

This document explains how Converge detects structural degradation patterns ("bombs") that signal a codebase is approaching a failure state. It covers both per-change and system-level detection. It is written for an external audience — no prior knowledge of the Converge internals is assumed.

---

1. What Problem Do Bombs Solve?

Individual risk metrics (entropy, containment, propagation) capture how risky a single change is. But some failure modes are emergent — they arise not from any single metric being high, but from combinations of indicators aligning in dangerous patterns.

Converge calls these patterns bombs: structural degradation conditions that predict cascading failures, dependency spirals, or system-wide meltdown. They operate at two levels:

Level	Scope	Detection Module
Per-change	Single proposed change	`risk/bombs.py`
System-level	Aggregate trends across all recent changes	`projections/predictions.py`

Bombs are not policy gates — they don't directly block changes. Instead, they generate high-severity diagnostics that inform the risk evaluation. A change with detected bombs will have elevated risk findings, which in turn may cause the risk gate or policy gates to block it.

---

2. Per-Change Bombs

Three structural patterns are detected for each proposed change:

2.1 Cascade Bomb

Question: "Does this change touch load-bearing files that fan out to many others?"

Detection logic:

```
1. Compute PageRank for all nodes in the dependency graph
2. Find file nodes with PageRank > 1.5 / graph_size
   → These are the "high-centrality" files
3. Among those, find files with out-degree >= 3
   → These are "high-fanout" files (connect to 3+ other nodes)
4. For each high-fanout file, compute descendants (all reachable nodes)
5. If total affected nodes > files_changed × 1.5:
   → TRIGGER: cascade bomb
```

Source: `risk/bombs.py`, function `_detect_cascade`

What the thresholds mean

Threshold	Value	Why
`_CASCADE_PR_FACTOR`	1.5	PageRank threshold = 1.5 / graph_size. A node is “high centrality” if its PageRank is 1.5× the expected uniform value (1/N).
`_CASCADE_MIN_FANOUT`	3	A file needs at least 3 outgoing connections to qualify as “high fanout.” Fewer than 3 is normal structural connectivity.
`_CASCADE_BLAST_FACTOR`	1.5	Total blast radius must exceed 1.5× the number of changed files. This distinguishes “the change has wide impact” from “the graph is naturally connected.”

Why this matters   A cascade bomb means the change modifies files that are structurally critical: they have high centrality (many things depend on them) and high fan-out (they connect to many other nodes). A bug in such a file doesn’t just break the file — it cascades through the dependency graph, affecting nodes that are 2, 3, or more hops away.

Severity: high

Example:

```
Change touches src/core/engine.py (PageRank=0.15, out-degree=8)
engine.py connects to 12 descendant nodes
Change only modifies 3 files
12 > 3 × 1.5 = 4.5 → cascade bomb triggered

Output: "Change touches 1 high-centrality node(s) with
         potential cascade to 12 nodes"
```

2.2 Spiral Bomb

Question: "Does the dependency graph contain circular dependencies?"

Detection logic:

```
1. Check if graph is a DAG (directed acyclic graph)
2. If not → enumerate simple cycles
3. Filter to cycles with length >= 2 (at least 2 nodes)
4. If 2+ significant cycles found:
   → TRIGGER: spiral bomb
```

Source: `risk/bombs.py`, function `_detect_spiral`

What the thresholds mean

Threshold	Value	Why
`_SPIRAL_MIN_CYCLE_LEN`	2	Self-loops (length 1) are degenerate — meaningful cycles involve at least 2 nodes.
`_SPIRAL_MAX_CYCLES`	10	Cap cycle enumeration for performance. Finding all cycles in a dense graph can be exponential. 10 is enough to confirm the pattern.
`_SPIRAL_MIN_SIGNIFICANT`	2	A single cycle might be an acceptable design choice (mutual dependency between two closely related files). Two or more cycles indicate a systemic pattern.

Why this matters Circular dependencies create feedback loops: a change to node A propagates to node B, which propagates back to A. These loops are:

- Hard to reason about: The effect of a change depends on itself
- Fragile: Breaking any node in the cycle can cascade unpredictably
- Ordering-hostile: There's no valid merge order that respects all dependency edges

Severity: medium (less severe than cascade because cycles can be intentional in some designs)

Example:

```
Graph contains:
  src/auth/login.py → src/auth/session.py → src/auth/login.py (cycle 1)
  src/core/engine.py → src/core/policy.py → src/core/engine.py (cycle 2)

2 significant cycles → spiral bomb triggered

Output: "2 circular dependency cycle(s) detected"
```

Display limits Cycle details are truncated for readability: - Maximum 3 cycles shown (`_CYCLE_DISPLAY_LIMIT`) - Maximum 5 nodes per cycle shown (`_CYCLE_NODE_LIMIT`)

2.3 Thermal Death Bomb (Per-Change)

Question: “Are multiple entropy indicators elevated simultaneously?”

Detection logic:

```
Count how many of these 5 indicators are "hot":

1. files_changed > 10
2. conflict_count > 0
3. dependencies > 3
4. graph_components > 3
5. edge_count > node_count × 2

If 3+ indicators are hot:
  → TRIGGER: thermal death bomb
```

Source: `risk/bombs.py`, function `_detect_thermal_death`

What each indicator means

#	Indicator	Threshold	What It Signals
1	Files changed > 10	`_THERMAL_FILES_HOT = 10`	The change is large — many moving parts
2	Merge conflicts > 0	Any conflict	The change collides with current state
3	Dependencies > 3	`_THERMAL_DEPS_HOT = 3`	Complex ordering constraints
4	Components > 3	`_THERMAL_COMPONENTS_HOT = 3`	The change is highly fragmented
5	Edge density > 2×	`_THERMAL_EDGE_DENSITY_FACTOR = 2`	The graph is densely connected

Why 3 out of 5? Any single indicator being elevated is normal: a large change (10+ files) in one directory is fine; a conflict on a small change is manageable. But when 3 or more indicators are elevated simultaneously, the change is "hot" across multiple dimensions — large, conflicting, dependent, fragmented, and dense. This combination overwhelms the system's ability to safely integrate the change.

Severity: critical (the most severe per-change bomb)

Why "thermal death"? The name comes from thermodynamics — the heat death of the universe is the state of maximum entropy where no useful work can be done. A change with 3+ hot indicators is in this state: every dimension of risk is elevated, and the change is too disordered to safely integrate.

---

3. System-Level Bombs

System-level bombs detect patterns across all recent changes, not just one. They are computed by the predictions module using 24-hour time windows.

3.1 System Cascade Signal

Question: "Are many recent changes hitting high-propagation areas?"

```
Look at all risk evaluations in the last 24 hours
Count changes with propagation_score > 40
If 3+ changes have high propagation:
  → SIGNAL: bomb.cascade
```

Threshold	Value	Why
`_BOMB_PROPAGATION`	40	Propagation above 40 indicates wide blast radius
`_BOMB_CASCADE_COUNT`	3	Three high-propagation changes in 24h is a pattern, not coincidence

What this means: Multiple changes with wide blast radii are entering the system simultaneously. Each one individually might be manageable, but together they create overlapping blast zones — a failure in any one could interact with the others.

Severity: high

Source: `projections/predictions.py`, function `_detect_bomb_cascade`

3.2 System Spiral Signal

Question: “Are containment scores trending downward?”

```
Compare average containment in last 24h vs previous 24h

If avg_containment_now < avg_containment_prev - 0.1
   AND avg_containment_now < 0.6:
  → SIGNAL: bomb.spiral
```

Threshold	Value	Why
`_BOMB_SPIRAL_CONT_DROP`	0.1	A 10-percentage-point drop is significant
`_BOMB_SPIRAL_CONT_ABS`	0.6	Containment below 0.6 is already in the warning zone

What this means: Changes are becoming less isolated over time. The system’s modular boundaries are eroding — each new change reaches further across boundaries than the last. This is a leading indicator of cascading failures: as containment drops, the blast radius of each change expands.

Severity: medium

Source: `projections/predictions.py`, function `_detect_bomb_spiral`

3.3 System Thermal Death Signal

Question: “Are entropy, conflict rate, and propagation all elevated simultaneously?”

```
In the last 24 hours:
  avg_entropy > 20
  AND conflict_rate > 20%
  AND avg_propagation > 30

If all three are true:
  → SIGNAL: bomb.thermal_death
```

Threshold	Value	Why
`_THERMAL_ENTROPY`	20	Average entropy above 20 means changes are consistently disordered
`_THERMAL_CONFLICT`	0.2 (20%)	One in five changes has merge conflicts

Threshold	Value	Why
`_THERMAL_PROPAGATION`	30	Average propagation above 30 means changes have wide blast radii

What this means: The system has reached a state where nearly every new change is disordered, conflicts are frequent, and blast radii are wide. At this point, the system is in a positive feedback loop: high entropy causes conflicts, conflicts cause retries, retries consume resources and delay other changes, which increases entropy further.

Severity: critical

Recommendation: "Halt new intents — system entropy is approaching critical levels"

Source: `projections/predictions.py`, function `_detect_bomb_thermal`

---

4. Other System-Level Signals

The predictions module also detects signals that aren't "bombs" but contribute to the overall predictive picture:

4.1 Rising Conflict Rate

```
conflict_rate_24h > conflict_rate_prev_24h + 0.1
AND sample_count > 3
```

Conflict rate rising by more than 10 percentage points. Severity: high.

4.2 Entropy Spike

```
avg_entropy_24h > avg_entropy_prev_24h × 1.2
AND avg_entropy_24h > 15
AND sample_count > 3
```

Average entropy rose by 20% and exceeds absolute floor of 15. Severity: medium.

4.3 Queue Stalling

```
requeued_count_24h > 5
```

More than 5 intents requeued in 24h — the queue is churning. Severity: high.

4.4 High Rejection Rate

```
rejected / (rejected + merged) > 40%
AND total_decisions > 3
```

More than 40% of intents are being rejected. Severity: critical.

Source: `projections/predictions.py`

---

5. How Bombs Feed Into the System

5.1 Per-Change Bombs

```
                 risk evaluation
                      |
            detect_bombs(intent, simulation, G)
                      |
               bombs: list[dict]
                 |          |
                 ▼          ▼
         RiskEval.bombs   build_diagnostics()
                 |          |
                 ▼          ▼
         API response    Diagnostic output
         (visible in     "diag.bomb.cascade:
          dashboard)      Split change to avoid
                          touching high-centrality
                          files simultaneously"
```

Bombs are stored in `RiskEval.bombs` and generate diagnostics with specific recommendations:

Bomb Type	Diagnostic Code	Recommendation
`cascade`	`diag.bomb.cascade`	"Split change to avoid touching high-centrality files simultaneously"
`spiral`	`diag.bomb.spiral`	"Break circular dependencies before merging"
`thermal_death`	`diag.bomb.thermal_death`	"System is under stress — reduce change scope immediately"

Source: `risk/eval.py`, constant `_BOMB_RECOMMENDATIONS`

5.2 System-Level Signals

System-level signals flow into the predictions API and the dashboard:

```
predict_issues(tenant_id)
     |
     ▼
 list[dict] with:
   signal: "bomb.thermal_death"
   severity: "critical"
```

```
    message: "System under thermal stress..."
    recommendation: "Halt new intents..."
      |
      ▼
  Dashboard alerts
  Health predictions
  Intake throttling decisions
```

---

6. The Relationship Between Per-Change and System-Level

Dimension	Per-Change (`risk/bombs.py`)	System-Level (`projections/predictions.py`)
Scope	Single intent + simulation	All intents in last 24–48h
Data source	Dependency graph of one change	Event log aggregates
Cascade	PageRank + fanout in one graph	Multiple high-propagation changes
Spiral	Cycle detection in one graph	Containment score trending downward
Thermal	5 indicators on one change	3 system metrics all elevated
When detected	During risk evaluation	During health/prediction snapshot
Attached to	`RiskEval.bombs`	`predict_issues()` output

A per-change thermal death bomb means this specific change is dangerously complex. A system-level thermal death signal means the entire codebase is in a dangerous state. Both are important, and they can occur independently — a single safe change doesn't prevent system-level thermal death, and a dangerous change can occur in an otherwise healthy system.

---

7. Detection Constants Summary

Per-Change (risk/bombs.py)

Constant	Value	Used By
`_CASCADE_PR_FACTOR`	1.5	Cascade: PageRank threshold factor
`_CASCADE_MIN_FANOUT`	3	Cascade: minimum out-degree
`_CASCADE_BLAST_FACTOR`	1.5	Cascade: blast radius must exceed files×1.5
`_SPIRAL_MIN_CYCLE_LEN`	2	Spiral: minimum nodes per cycle
`_SPIRAL_MAX_CYCLES`	10	Spiral: cap on cycle enumeration

Constant	Value	Used By
`_SPIRAL_MIN_SIGNIFICANT`	2	Spiral: minimum cycles to trigger
`_THERMAL_FILES_HOT`	10	Thermal: file count indicator
`_THERMAL_DEPS_HOT`	3	Thermal: dependency count indicator
`_THERMAL_COMPONENTS_HOT`	3	Thermal: component count indicator
`_THERMAL_EDGE_DENSITY_FACTOR`	2	Thermal: edge density indicator
`_THERMAL_MIN_INDICATORS`	3	Thermal: minimum hot indicators to trigger

System-Level (projections/predictions.py)

Constant	Value	Used By
`_BOMB_PROPAGATION`	40	Cascade: propagation score threshold
`_BOMB_CASCADE_COUNT`	3	Cascade: minimum high-propagation changes
`_BOMB_SPIRAL_CONT_DROP`	0.1	Spiral: containment drop threshold
`_BOMB_SPIRAL_CONT_ABS`	0.6	Spiral: absolute containment threshold
`_THERMAL_ENTROPY`	20	Thermal: average entropy threshold
`_THERMAL_CONFLICT`	0.2	Thermal: conflict rate threshold (20%)
`_THERMAL_PROPAGATION`	30	Thermal: average propagation threshold

---

8. Design Rationale Summary

Design Choice	Rationale
Three bomb types	Each captures a distinct failure mode: concentrated impact (cascade), feedback loops (spiral), and systemic overload (thermal). Together they cover the major structural degradation patterns.
Per-change AND system-level	A single change can be dangerous in a healthy system; a healthy change can be dangerous in a degraded system. Both perspectives are needed.

Design Choice	Rationale
PageRank for cascade	Identifies load-bearing files algorithmically. No manual file classification needed. PageRank captures both direct and transitive importance.
Cycle enumeration with cap	Finding all cycles is potentially exponential. The cap (10) provides enough evidence to confirm the pattern without unbounded computation.
3/5 indicator threshold for thermal	Requires convergence of multiple signals. Any single indicator is normal; three simultaneous ones are not. This reduces false positives.
24h time windows for system-level	Recent enough to be actionable, long enough to be statistically meaningful. Comparing current 24h to previous 24h detects changes in behavior, not just absolute levels.
Bombs inform, not block	Bombs generate diagnostics and findings that influence risk scores, but they don't have their own policy gate. This prevents false-positive bombs from blocking changes directly while ensuring they contribute to the overall risk picture.
Display limits	Truncating cycle lists and trigger node lists keeps output readable. The full data is in the graph — the diagnostic only needs to convey the pattern.

---

9. File Reference

File	Role
`risk/bombs.py`	Per-change bomb detection: cascade, spiral, thermal death
`projections/predictions.py`	System-level signal detection: cascade, spiral, thermal, plus rising conflicts, entropy spikes, queue stalling, rejection rate
`risk/eval.py`	Calls `detect_bombs`, generates diagnostics from bomb results, bomb recommendation constants
`risk/graph.py`	Provides the dependency graph that bombs analyze (PageRank, components, edges)

File	Role
`projections/health.py`	Health projection system that feeds system-level detection
`models.py`	`RiskEval.bombs` field, event types for health/prediction