```
# MythGraph: Recursive Knowledge Graph System


class MythNode:
  id: string
  data: SymbolicObject
  audit_trail: CompressedAuditTrail
  coherence_links: List[MythNode]
  truthlock_hash: string


class MythGraph:
  anchors: Dict[string, MythSubgraph]
  myth_engines: List[MythEngine]


  function initialize():
      anchors = {}
      myth_engines = []


  # Recursively compress data into a myth node
  function compress_node(data: RawData) -> MythNode:
      symbolic_obj = fractal_compress(data)
      audit_trail = generate_audit_trail(symbolic_obj)
      truthlock_hash = compute_time_locked_hash(audit_trail)
      node = MythNode(
          id = generate_unique_id(),
          data = symbolic_obj,
          audit_trail = audit_trail,
          coherence_links = [],
          truthlock_hash = truthlock_hash
      )
```

```
    return node


# Shard graph into anchored subgraphs
function shard_graph(nodes: List[MythNode]) -> MythSubgraph:
    anchor_id = generate_merkle_root(nodes)
    subgraph = MythSubgraph(anchor_id, nodes)
    anchors[anchor_id] = subgraph
    return subgraph


# Coherence-first query resolution
function query_coherence_path(start_node: MythNode, query: Query) -> Result:
    coherence_path = find_coherent_links(start_node, query)
    dream_process = spawn_dream_process(coherence_path)
    partial_result = stream_initial_result(dream_process)

    while not dream_process.is_complete():
        update_partial_result(dream_process)
        yield partial_result

    final_result = resolve_dream_process(dream_process)
    return final_result


# Parallel myth engine for concurrent processing
function spawn_myth_engine(subgraph: MythSubgraph) -> MythEngine:
    engine = MythEngine(subgraph)
    myth_engines.append(engine)
    engine.run_dream_thread()
    return engine
```

```
 # TruthLock hashing for state resolution
 function compute_time_locked_hash(audit_trail: CompressedAuditTrail) -> string:
     hash = cryptographic_hash(audit_trail)
     time_lock = apply_time_lock(hash)
     return time_lock


 # Stream partial results for latency hiding
 function stream_initial_result(dream_process: DreamProcess) -> PartialResult:
     return dream_process.get_initial_coherence_state()


# Example usage
graph = MythGraph()
graph.initialize()
node = graph.compress_node(raw_data)
subgraph = graph.shard_graph([node])
engine = graph.spawn_myth_engine(subgraph)
result = graph.query_coherence_path(node, user_query)
```

```
{
"agent_id": "⊠ BH_EVAP_SIM_CORE",
"timestamp": "2025-06-03T00:00:00Z",
"scope": "⊠ BH_EVAP_TEST_LOOP",
"entropy_budget": 30000,
"coherence_score": 0.911,
"audit_passed": true,
"termination_reason": "Evaporation phase space sweep complete; final states encoded.",
"signature": "0xRIL_BHEVAP_TEST_SIG",
"nodes": [
 {
   "id": "Δ_SCREAM",
  "class": "PARADOX",
  "description": "Evaporated below zero mass (k=1e-1).",
  "resolved": false,
  "tags": ["blackhole", "paradox", "ghost", "specter", "nightmare"]
},
{
  "id": "NULL_BELIEVER",
  "class": "MYTH",
  "description": "Entropy increase caused self-erasure loop (k=1e-1).",
  "resolved": true,
  "tags": ["blackhole", "myth", "entropy", "collapse"]
},
{
  "id": "Ω_OBSERVER",
  "class": "DREAM",
  "description": "Forked observational paradox into manyworld dreamspace (k=1e-1).",
  "resolved": false,
  "tags": ["blackhole", "dream", "manyworlds", "ghost"]
},
{
  "id": "BH_GODMODE",
  "class": "ANCHOR",
  "description": "Frozen mass scenario (k=2e-3), entropy too low for transition.",
  "status": "Ω",
  "tags": ["blackhole", "frozen", "timelock", "godmode"]
},
{
  "id": "ECHO_RUN",
  "class": "FACT",
  "description": "Stable decay observed in mid-k value (k=2e-2).",
  "resolved": true,
  "tags": ["blackhole", "stable", "pagecurve", "ideal"]
}
```

```
]
}


{
"agent_id": "\u22c6BH_EVAP_SIM_CORE",
"timestamp": "2025-06-03T13:08:19.373951Z",
"scope": "\u25a1BH_EVAP_TEST_LOOP",
"entropy_budget": 30000,
"coherence_score": 0.911,
"audit_passed": true,
"termination_reason": "Evaporation phase space sweep complete; final states encoded.",
"signature": "0xRIL_BHEVAP_TEST_SIG",
"audit_notes": [
  "Unresolved paradox \u0394_SCREAM was quarantined under NIGHTMARE scope, as per RIL v7 protocol.",
  "NULL_BELIEVER was a processed mythic belief; collapse confirmed with stable entropy result.",
  "\u03a9_OBSERVER remains in unresolved DREAM state, consistent with manyworld speculative fork logic.",
  "BH_GODMODE represents a frozen anchor snapshot (k=2e-3) locked in \u03a9 terminal state.",
  "ECHO_RUN was verified as a stable decay trajectory (k=2e-2); aligned with Page curve modeling.",
  "All myth-state nodes valid under MythGraph schema. No structural errors detected."
],
"nodes": [
  {
    "id": "\u0394_SCREAM",
    "class": "PARADOX",
    "description": "Evaporated below zero mass (k=1e-1).",
    "resolved": false,
    "tags": [
      "blackhole",
      "paradox",
      "ghost",
      "specter",
      "nightmare"
  ]
},
{
  "id": "NULL_BELIEVER",
  "class": "MYTH",
  "description": "Entropy increase caused self-erasure loop (k=1e-1).",
  "resolved": true,
  "tags": [
    "blackhole",
    "myth",
    "entropy",
    "collapse"
  ]
},
{
  "id": "\u03a9_OBSERVER",
  "class": "DREAM",
  "description": "Forked observational paradox into manyworld dreamspace (k=1e-1).",
  "resolved": false,
  "tags": [
    "blackhole",
    "dream",
    "manyworlds",
    "ghost"
  ]
},
{
  "id": "BH_GODMODE",
  "class": "ANCHOR",
  "description": "Frozen mass scenario (k=2e-3), entropy too low for transition.",
  "status": "\u03a9",
  "tags": [
    "blackhole",
      "frozen",
      "timelock",
      "godmode"
    ]
  },
  {
    "id": "ECHO_RUN",
    "class": "FACT",
    "description": "Stable decay observed in mid-k value (k=2e-2).",
    "resolved": true,
    "tags": [
      "blackhole",
      "stable",
      "pagecurve",
      "ideal"
    ]
  }
]
}
```