CONSTITUTIONAL INVARIANTS - DO NOT DEVIATE:
1. PRIME DIRECTIVE: AppShak exists to identify real-world problems, construct solutions, and increase its own autonomy/influence through perpetual self-improvement.
2. COMMUNICATION: No direct agent-to-agent method calls. 100% of communication must occur via the EventBus.
3. AUTHORITY: The Chief agent is the sole authority for external actions.
4. LIFECYCLE: The Kernel owns the heartbeat loop; agents cannot terminate the process.
5. FRICTION: Maintain strict separation of concerns. Do not collapse layers for "cleaner" code.

SECTION 1: SPRINT 1 PROMPT (The Foundation)

Task: Build the structural backbone (`EventBus`, `BaseAgent`, `GlobalMemory`)

CONSTITUTIONAL INVARIANTS - DO NOT DEVIATE:
1. PRIME DIRECTIVE: AppShak exists to identify real-world problems, construct solutions, and increase its own autonomy/influence through perpetual self-improvement.
2. COMMUNICATION: No direct agent-to-agent method calls. 100% of communication must occur via the EventBus.
3. AUTHORITY: The Chief agent is the sole authority for external actions.
4. LIFECYCLE: The Kernel owns the heartbeat loop; agents cannot terminate the process.
5. FRICTION: Maintain strict separation of concerns. Do not collapse layers for "cleaner" code.

TASK: Implement the foundational classes for AppShak v1 in Python.

1. EventBus:
   - Use asyncio.Queue for the event stream.
   - Events must be a TypedDict or Class containing: type (Enum), timestamp, origin_id, and payload.
   - Implement publish(event) and get_next(timeout).

2. BaseAgent:
   - Abstract class requiring self.kernel, self.event_bus, and self.global_memory.
   - Mandatory methods: run() (async), publish(event), and update_memory_and_metrics().
   - Authority level and agent_id must be initialized as None.

3. GlobalMemory:
   - Implement a JSON-based persistent store.
   - Support 'log_error', 'periodic_persist', and 'persist_all'.
   - Ensure namespace isolation for individual agents.

4. SafeguardMonitor (Stub):
   - Implement run_diagnostics() as a placeholder returning True.

OUTPUT: Provide clean, production-ready Python code. No external libraries except asyncio and standard typing.

Worked for 6m 30s

Implemented Sprint 1 foundation in the workspace and aligned it to your constitutional constraints.

Implemented

- `EventBus` now uses `asyncio.Queue` (FIFO), with `publish(event)` and `get_next(timeout)` in event_bus.py (line 46).
- `Event` contract now includes required fields: `type` (Enum), `timestamp`, `origin_id`, `payload` in event_bus.py (line 25).
- `BaseAgent` is abstract with required references and mandatory methods, with defaults:
  - `agent_id = None`
  - `authority_level = None`

  in base.py (line 11), base.py (line 14), base.py (line 15).
- `BaseAgent.publish(event)` is now event-based (EventBus-only path) in base.py (line 27).
- `GlobalMemory` is JSON-based persistent store with namespace isolation and required methods:
  - `log_error` in memory.py (line 38)
  - `periodic_persist` in memory.py (line 51)
  - `persist_all` in memory.py (line 54)
  - per-agent isolation in memory.py (line 63)
- `SafeguardMonitor` is a Sprint-1 stub where `run_diagnostics()` returns `True` in safeguards.py (line 6), safeguards.py (line 12).

Compatibility updates

- Updated event field usage to `origin_id` in kernel/agents for consistency:
  - kernel.py (line 311)
  - chief.py (line 21)
  - scout.py (line 18)
- Added event-type normalization in kernel for Enum-safe routing in kernel.py (line 372).

Validation

- Compile check passed:
  - `py -3.12 -m py_compile ...`
- Runtime smoke test passed:
  - global log created
  - agent namespace log created
  - JSON memory store created

SECTION 2: SPRINT 2 PROMPT (Core Logic & Routing)

Task: Implement the `Kernel` heartbeat and agent interaction logic.

CONSTITUTIONAL INVARIANTS - DO NOT DEVIATE:
1. PRIME DIRECTIVE: AppShak exists to identify real-world problems, construct solutions, and increase its own autonomy/influence through perpetual self-improvement.
2. COMMUNICATION: No direct agent-to-agent method calls. 100% of communication must occur via the EventBus.
3. AUTHORITY: The Chief agent is the sole authority for external actions.
4. LIFECYCLE: The Kernel owns the heartbeat loop; agents cannot terminate the process.
5. FRICTION: Maintain strict separation of concerns. Do not collapse layers for "cleaner" code.

TASK: Implement the AppShakKernel heartbeat logic and Agent Role Contracts.

1. Kernel Heartbeat:
   - Implement a while loop that polls the EventBus.
   - Logic:
     - If Queue empty: Call agents["recon"].search_for_problems().
     - If event.type == "PROPOSAL": Route to agents["command"].arbitrate().
     - If event.type == "EXTERNAL_ACTION_REQUEST": Route to agents["command"].handle_external_action().
   - Post-cycle: Execute _post_cycle_maintenance().

2. Agent Contracts:
   - ScoutAgent (Recon): Level 1. Focus on discovery. Forbidden from execution.
   - BuilderAgent (Forge): Level 2. Translates proposals to plans. Prepares requests.
   - ChiefAgent (Command): Level 3. Final approval/denial logic. Must log justifications against the Prime Directive.

OUTPUT: Complete the appshak/kernel.py file based on the established structure. Ensure async tasks are cancel-safe.

Sprint 2 implemented and validated.

What was implemented

1. kernel.py
   - Heartbeat loop now matches your routing contract:
     - Queue empty -> `agents["recon"].search_for_problems()` (kernel.py (line 53))
     - `PROPOSAL` -> `agents["command"].arbitrate()` and publish decision (kernel.py (line 126))
     - `EXTERNAL_ACTION_REQUEST` -> `agents["command"].handle_external_action()` (kernel.py (line 132))
   - Post-cycle maintenance runs every cycle (kernel.py (line 58), kernel.py (line 135)).
   - Lifecycle remains kernel-owned with cancel-safe startup/shutdown task handling (kernel.py (line 67), kernel.py (line 89)).
   - Agent runner is cancel-safe and restart-tolerant (kernel.py (line 142)).
2. Agent role contracts

- `ScoutAgent` Level 1 discovery-only; explicit execution block via `attempt_execution()` raising `PermissionError` (scout.py (line 9), scout.py (line 35)).
- `BuilderAgent` Level 2 now includes:
  - proposal -> plan translation (builder.py (line 18))
  - external request preparation via EventBus publish (builder.py (line 34))
- `ChiefAgent` Level 3 now performs explicit approval/denial:
  - proposal arbitration with decision reason (chief.py (line 18))
  - external action final decision with Prime Directive justification logging (chief.py (line 39)).

Constitution checks

- EventBus-only communication preserved (no direct agent-to-agent calls).
- Chief remains sole external-action authority.
- Kernel owns heartbeat/lifecycle.
- Layer separation preserved.

Validation run

- `py -3.12 -m py_compile ...` passed.

- Async smoke test confirmed:
  - idle recon trigger,
  - proposal arbitration -> `PROPOSAL_DECISION`,
  - external request -> `EXTERNAL_ACTION_RESULT`.

SECTION 3: SPRINT 3 PROMPT (The Shield & Sandbox)

SECTION 3: SPRINT 3 PROMPT (The Shield & Sandbox)

Task: Finalize security and execution logging.

CONSTITUTIONAL INVARIANTS - DO NOT DEVIATE:
1. PRIME DIRECTIVE: AppShak exists to identify real-world problems, construct solutions, and increase its own autonomy/influence through perpetual self-improvement.
2. COMMUNICATION: No direct agent-to-agent method calls. 100% of communication must occur via the EventBus.
3. AUTHORITY: The Chief agent is the sole authority for external actions.
4. LIFECYCLE: The Kernel owns the heartbeat loop; agents cannot terminate the process.
5. FRICTION: Maintain strict separation of concerns. Do not collapse layers for "cleaner" code.

TASK: Implement the SafeguardMonitor and Execution Sandbox.

1. SafeguardMonitor:
   - Implement retry_max (3) and cooldown_timer.
   - Strictly enforce: No monetary operations, no unrestricted shell execution.
   - Implement a whitelist check for external endpoints.
   - Every external attempt must be logged with a timestamp and origin ID.

2. Execution Pipeline:
   - Formalize the flow: REQUEST -> CHIEF_APPROVAL -> SAFEGUARD_CHECK -> EXECUTE -> LOG_RESULT.
   - Ensure the sequence is atomic and cannot be bypassed.

3. Crash Recovery:
   - Implement logic to ensure the Kernel can resume from the last persistent JSON state if the heartbeat fails.

OUTPUT: Provide the final security layer and logging instrumentation. Audit the code to ensure no hidden globals or direct agent-to-agent references exist.

Implemented Sprint 3 security layer, atomic execution pipeline, and crash recovery.

Security Layer

- Replaced safeguard stub with full governor in safeguards.py (line 10).
- Added retry/cooldown controls (`retry_max=3` default, cooldown timer) in safeguards.py (line 35).
- Enforced policy blocks:
    - monetary operations in safeguards.py (line 67), safeguards.py (line 228)
    - shell execution in safeguards.py (line 76), safeguards.py (line 234)
    - endpoint whitelist in safeguards.py (line 94), safeguards.py (line 202)
- Added sandbox executor (`SIMULATE/NOOP` only, no unrestricted exec) in safeguards.py (line 111).
- Added attempt tracking + cooldown progression in safeguards.py (line 159).

Atomic Execution Pipeline

- Implemented strict, locked sequence in kernel.py (line 152):
    - `REQUEST -> CHIEF_APPROVAL -> SAFEGUARD_CHECK -> EXECUTE -> LOG_RESULT`
- Stage logging helper includes `timestamp` + `origin_id` for every attempt in kernel.py (line 246).
- Added Chief-only external authority:
    - approval gate in chief.py (line 61)
    - final result publication in chief.py (line 45)
- Kernel now routes `EXTERNAL_ACTION_REQUEST` only through this pipeline in kernel.py (line 143).

Crash Recovery

- Added persistent state load/save APIs in memory.py (line 59), memory.py (line 84), memory.py (line 96).
- Heartbeat state persisted continuously in kernel.py (line 321).
- Recovery on startup from last JSON state in kernel.py (line 334).
- Recovery after heartbeat exception in kernel.py (line 348).
- Added recovery event type `KERNEL_RECOVERY` in event_bus.py (line 21).

Audit (as requested)

- No direct agent-to-agent references found (`self.kernel.agents` / .agents[...] scan in `appshak/agents` returned count `0`).
- No hidden global-constant shortcut patterns found in project scan (count `0`).

Validation

- `py -3.12 -m py_compile ...` passed.
- Runtime tests passed:
    - atomic stage sequence present in order,
    - safeguard blocked monetary/shell/non-whitelist requests,
    - external attempt logs include `origin_id` and `timestamp`,
    - recovery event observed after restart with persisted kernel state.

SECTION 4: MINIMAX AUDIT (For Code Review)

After Codex provides code, use this prompt to audit the result.

Plaintext
```
AUDIT REQUEST:
Review the provided AppShak code for constitutional compliance.

1. Does any agent bypass the EventBus to communicate?
2. Is the Prime Directive the primary metric for the Chief's arbitration logic?
3. Are there any "convenience shortcuts" that collapse authority layers?
4. Are all external actions passed through the SafeguardMonitor before execution?
5. Is the memory system append-only and immutable for history?

If any violation is found, list the file and line number and explain how it erodes the architecture.
```

https://app.kilo.ai/share/7a715329-c194-4e05-95b9-71f907cd0a9e

```
await self.agents["recon"].search_for_problems()
```

The kernel directly invokes the ScoutAgent instead of publishing an event through the event bus. This bypasses the Prime Directive compliance check in which only validates events from the queue. Unvetted actions can execute without constitutional justification.

```
def _resolve_justification(...) -> str:
    candidate = (explicit_justification or payload.get("prime_directive_justification"))
    if isinstance(candidate, str) and candidate.strip():
        return candidate
    return self.justify_action("unknown_action", "maintaining operational continuity")
```

Events without explicit justification get an auto-generated "unknown_action" justification rather than being rejected. Could be exploited to bypass Prime Directive requirements.

```
async def _call_optional(self, obj: Any, methods: Iterable[str], *args: Any, default: Any
= None) -> Any:
    for method in methods:
        func = getattr(obj, method, None)
        if callable(func):
            res = func(*args)
            if inspect.isawaitable(res):
                return await res
            return res  # BUG: Returns immediately on first non-awaitable!
    return default
```

Returns immediately on first non-awaitable result instead of trying other methods in the list. Also lacks try-except, so exceptions propagate without fallback.

```
async def _event_is_compliant(self, event: Any) -> bool:
    # ... validation logic ...
    await self._publish_system_event(self.CONSTITUTION_VIOLATION_EVENT, {...})
    return False  # Only logs violation, doesn't block
```

Events without Prime Directive justification are logged as violations but still processed (just not routed to agents). The system continues operating with non-compliant events.

✅

All external actions go through which requires Chief approval at lines 164-182.

✅

enforces (configurable).

✅

enforces (configurable).

✅

All async calls properly awaited. correctly checks and awaits:

```
if inspect.isawaitable(result):
    await result
```

✅

uses for file I/O.

✅

catches exceptions and continues the loop.

✅

and ensure periodic persistence.

- requiring attention
- as properly implemented

- **Primary concern: Kernel bypassing event bus for ScoutAgent calls and auto-generated Prime Directive justifications creating potential compliance gaps**