

Hierarchical Resource Brokerage and Spatial Traffic Management: Evolutionary Architectures in High-Performance Screeps Codebases

The fundamental challenge in Screeps, a persistent real-time strategy environment for programmers, lies in the efficient management of asynchronous unit logic within a constrained CPU budget.¹ As a player's colony matures, the simplistic role-based logic found in introductory scripts—where units (creeps) independently poll the environment for tasks—becomes a primary bottleneck for both throughput and computational efficiency.² Top-tier codebases such as Overmind, ScreepsOS, and BonzAI have addressed these limitations by migrating toward "Kernel" or "Operating System" (OS) architectures that abstract creep behavior into discrete processes managed by a centralized scheduler.² Within these sophisticated systems, the management of "idle" states, particularly for logistics units like transporters, represents a critical intersection of pathfinding optimization, geometric base planning, and predictive task brokerage.⁴

The Emergence of the Kernel-Process Paradigm

The architectural shift from monolithic loops to kernel-based systems is driven by the necessity for process isolation and priority-based scheduling.² In a standard Screeps script, every creep must evaluate its logic every tick, leading to redundant calculations and "racing" conditions where multiple units target the same resource.⁶ An OS-style bot, conversely, utilizes a kernel to manage a "process table," where each game task (e.g., source mining, extension filling, or room defense) is treated as a runnable entity with its own lifecycle.²

Core Architectural Components of Screeps OS Models

The sophistication of an OS-style codebase is defined by its ability to separate high-level decision-making from low-level unit execution. The kernel provides an API that processes use to interact with the game world, often utilizing "syscalls" to yield control or request resources.²

OS Component	Functional Description	Impact on Logistics and Idling
Process	A modular logic container	Isolation ensures that a

	(e.g., MiningProcess, HaulingProcess) with its own state and priority. ²	hauler's idle logic does not interfere with critical defense processes.
Scheduler	The engine that decides the execution order of processes based on priority and CPU bucket availability. ²	High-priority logistics requests are fulfilled before secondary tasks, minimizing unit downtime.
Process Table	A persistent registry of all active tasks, allowing state to survive across multiple game ticks. ²	Enables creeps to "remember" their last task or parking location even after a global reset.
File System	An abstraction layer over Memory and Segments that handles data persistence. ²	Centralizes the storage of traffic maps and parking locations for colony-wide access.
Syscall Interface	A mechanism (often generator-based) for processes to communicate with the kernel. ³	Processes can yield new Sleep(n) to conserve CPU when units are idling at a rally point.

The use of generators (e.g., `run*() {}`) allows for sophisticated control flow, where a hauler process can pause mid-execution to wait for a resource to become available, effectively idling without re-evaluating its entire decision tree every tick.³

Hierarchical Command in the Overmind Framework

One of the most influential codebases in the Screeps ecosystem is Overmind, which adopts a centralized swarm intelligence model loosely themed around the Zerg from Starcraft.⁴ This architecture moves beyond simple role scripts and replaces them with a hierarchy of Overlords, HiveClusters, and Zerg (creep wrappers).⁴ The Overmind framework is particularly notable for its handling of transporters and their transition between active and idle states through a centralized logistics network.⁵

The Lifecycle of an Overmind Colony

In the Overmind model, each colony functions as a semi-autonomous unit with its own "Overseer" that reacts to stimuli and places "Directives" to achieve specific goals.⁴ The logic is

executed in three distinct phases: build(), init(), and run().⁴

1. **Build Phase:** The system instantiates colonies, hive clusters, and overlords. It performs all necessary caching and object instantiation, ensuring that the environment is fully mapped before any logic is executed.⁴
2. **Init Phase:** This phase handles pre-state-changing actions, such as generating spawn requests and transport or link requests.⁴ This is where the "Logistics Network" identifies which transporters are available and which tasks require fulfillment.⁵
3. **Run Phase:** The Overlords scan their associated Zerg and assign tasks through a decision tree to any unit that is currently isidle.⁴

The isidle predicate is a crucial check; if a transporter has no task assigned via the LogisticsNetwork or its parent Overlord, it falls into a standby state.⁴

The Logistics Network and Request Brokerage

Top-tier codebases handle transporters not as independent actors, but as units in a colony-wide resource distribution system.⁵ The "Transport Request System" in Overmind replaces individual creep "find" calls with a centralized broker that matches supply with demand.⁴

Producer-Consumer Dynamics

The logistics system categorizes every structure and resource point as either a provider or a requester of energy and minerals.⁵

- **Supply Side:** Mining sites, terminals (when receiving), and dropped resources generate supply requests to the LogisticsNetwork.⁵
- **Demand Side:** Spawns, extensions, towers, and labs generate withdrawal requests.⁵
- **The Broker:** The network calculates the optimal unit to fulfill a request based on proximity, carry capacity, and current fatigue.⁵

This system significantly reduces the "racing" problem where multiple haulers target the same container, only for the first one to arrive and leave the others empty-handed.⁶ By invalidating caches and using predictive algorithms to estimate when a container will be full, the system ensures that transporters are dispatched exactly when they are needed.⁵

Predicted Request Amount and Throughput Optimization

A key innovation in the Overmind logistics system is the predictedRequestAmount function.⁵ This function accounts for the time it takes a transporter to reach its target and the rate at which the target structure is gaining or losing energy.⁵ If a container is being filled by a miner at 10 energy per tick and a transporter is 20 tiles away, the broker knows the container will have 200 more energy by the time the transporter arrives.⁵ This allows the transporter to be "active"

even while technically moving toward an empty structure, effectively eliminating what would otherwise be idle time.⁵

The Mathematics of Movement: Fatigue and Body Planning

For logistics units, the "rally point" behavior is often a response to movement inefficiency.¹⁰ A creep's ability to move is governed by its fatigue, which is generated by its body parts as it traverses terrain.¹¹

The fatigue generation and reduction are calculated using the following relationships:

$$Fatigue_{Gen} = (\sum BodyParts_{Non-MOVE} - \sum CarryParts_{Empty}) \times TerrainMultiplier$$

$$Fatigue_{Red} = \sum MoveParts \times 2$$

Where the TerrainMultiplier is 1 for roads, 2 for plain land, and 10 for swamps.¹¹ Top-tier transporters are designed with specific MOVE to CARRY ratios to maximize their "duty cycle"—the percentage of time they are moving versus waiting for fatigue to dissipate.¹⁰

Transporter Strategy	Body Ratio (MOVE:CARRY)	Ideal Environment	Idling Consideration
Max Throughput	1:1	Mixed Terrain	Rarely idles due to high movement speed; high CPU cost per creep. ¹¹
Road Optimized	1:2	100% Road Coverage	More efficient per-creep; moves at 1 tile/tick on roads even when full. ¹¹
Stationary Manager	0:1	RCL8 Bunker Core	Never moves; spends 100% of life

			at an anchor tile, eliminating idling. ⁵
--	--	--	--

When a hauler is improperly balanced (e.g., too few MOVE parts), it appears to "idle" or stutter as it waits multiple ticks for its fatigue to reach zero.¹⁰ Top-tier bots use this math to ensure that transporters are always at maximum speed, reducing the number of units required and simplifying traffic management.⁵

Traffic Management and the "Push" Protocol

In a high-density colony, idle creeps are not merely unproductive; they are physical obstacles that can cause massive gridlock.⁵ Top-tier codebases handle this by treating idle creeps as "yieldable" entities.⁵

Priority-Based Yielding

Instead of static rally points, systems like Overmind use dynamic "Move Priorities" to characterize the importance of a creep's current assignment.⁵

Move Priority	Unit Assignment	Behavioral Rule
Emergency	Invasion Defense, Tower Refilling. ⁵	All other units must move to an adjacent tile to clear the path.
Logistics	Transporters moving energy to Spawns/Extensions. ⁵	Yields only to Emergency units.
Normal	Remote Mining, Scavenging. ⁵	Yields to Logistics and Emergency units.
Idle	Standby Transporters, Off-duty Workers. ⁵	Must move if any non-idle unit attempts to enter its tile.

When a high-priority creep attempts to move into a tile occupied by a lower-priority creep, the system executes a "Push" behavior.⁵ The idle creep is forced to find an adjacent free tile and move into it, potentially triggering a "Recursive Push" if that tile is also occupied.⁵ This ensures that "rallying" units never block the critical path of a hatchery or a mining site.⁵

Recursive Pushing in the Hatchery

One of the most complex areas for traffic management is the hatchery, where newly spawned creeps must immediately clear the area to allow the next unit to emerge.⁵ Overmind implements a recursive pushing algorithm that allows a spawn structure to "shove" an idle unit out of the way the same tick it is spawned.⁵ This is critical for RCL8 rooms that may be spawning max-sized creeps every few ticks, where even a single idle hauler could delay the entire production pipeline.⁵

Geometric Rallying and Base Planning

Where a creep idles is as important as how it idles. Top-tier bots use sophisticated base planning algorithms to identify "safe" zones or "dead-ends" where idle creeps can wait without obstructing traffic.¹⁵

The Distance Transform Algorithm

Base planners like those used by "commie bot" and Overmind utilize a "Distance Transform" to map the room.¹⁶ This algorithm calculates the distance of every walkable tile from the nearest wall or structure.¹⁶

- **Open Area Identification:** Tiles with a high distance transform value (e.g., > 3) represent large open spaces.¹⁶ These are often designated as rally points for combat squads or idle transporters, as they provide ample space for units to shuffle around without being trapped against a wall.¹⁵
- **Bunker Layouts:** In a "Bunker" or "Hive" layout, the base is designed around a 5x5 or 7x7 core.¹⁴ This compact design creates specific "parking" tiles—usually near the storage or terminal—where transporters can idle in range 1 of their primary provider without blocking the corridors used by other workers.¹⁴

Floodfill and Traffic Zoning

After a base is planned, the bot may run a "Floodfill" algorithm to categorize tiles by their proximity to core structures.¹⁶

- **Zone 0 (The Core):** Only stationary managers and high-priority queens are allowed to remain here.⁵
- **Zone 1 (Logistics Ring):** Active transporters traverse this zone. Idle transporters are encouraged to move to Zone 2.¹⁴
- **Zone 2 (The Periphery):** Designated as the "Parking Zone" for idle workers and transporters.¹⁴

By pushing idle units to the periphery, the bot ensures that the pathfinding cost for high-priority units remains low, as they do not have to calculate complex paths around

stationary obstacles.⁵

The RCL8 Stationary Manager: The Ultimate Idle State

The peak of logistics architecture is the "Stationary Manager" used in RCL8 bunker layouts.⁵ In these systems, the traditional "hauler" role is split between long-distance mobile transporters and a single, immobile central manager.¹⁴

The Central Anchor

The manager is a specialized creep with a body plan of `` (or variations thereof).¹⁴ It is spawned without MOVE parts and is moved to its central "anchor" tile either by a specialized "pusher" creep or by being spawned directly onto the tile if the hatchery layout allows.⁵

- **Functional Connectivity:** From the anchor tile, the manager can reach the storage terminal, central link, and several power-user structures (like the power spawn or nuker).¹⁴
- **Zero-Idle Logic:** Because the manager is stationary, it never occupies a tile it doesn't need, and it never needs to "rally".⁵
- **Productive Standby:** When the manager has no transport requests to fill, it utilizes its 32 WORK parts to fortify the bunker's central ramparts.¹⁴ This "idling with intent" ensures that the creep's life—and the player's energy—is never wasted.¹⁴

Defensive Parking and Panic Modes

Invasion defense forces a dramatic shift in idle behavior.⁵ When a room enters a high "DEFCON" state, transporters cannot simply idle in their usual zones.⁵

Rampart-Aware Idling

Top-tier codebases implement a defensive posturing system where idle creeps must seek cover.⁵

- **Rampart Seeking:** If enemies are detected in a room, all idle creeps are ordered to move to the nearest tile protected by a rampart.⁵ This is handled by a RetreatTask that overrides standard parking logic.⁵
- **Safe Zone Clustering:** Miners and remote haulers will retreat toward the colony controller—the safest part of the room—if an invasion is too large for the local defense force to handle.⁵
- **Distraction and Meat-Shielding:** In some community-discussed patterns (though rarely in top-tier open-source bots), idle creeps are used as "meat shields" to absorb tower damage or to distract enemy attackers from vital structures like the spawn.⁷ However, the consensus in high-level play is that preserving the "time-cost" of the creep is more valuable than its hit points.⁷

The Abandon Directive and Terminal Evacuation

In the event of a total breach, the Overseer may place a DirectiveAbandon.⁵ This shifts all transporters from an idle or normal logistics state into an "Evacuation" state.⁵ Instead of rallying, units will continuously withdraw high-value minerals and energy from the terminal and move them to any available adjacent colony's storage, effectively "parking" the colony's wealth in a safer location before the room is lost.⁵

Persistence and Global Reset Resilience

A significant challenge for kernel-based bots is maintaining unit state across "Global Resets"—ticks where the JavaScript engine's global object is wiped and all non-memory variables are lost.²

Cross-Tick Caching

Overmind and ScreepsOS utilize a "Persistence" layer to ensure that idle creeps do not lose their assigned rally points during a reset.⁵

- **Global Refresh:** The Overmind object is rebuilt every 20 ticks, but in the interim, a refresh() method is called to restore references to game objects from their persistent IDs.⁵
- **Parsed Memory Restoration:** To avoid the high CPU cost of JSON.parse(), the bot saves its parsed memory state in the global scope.⁵ If a transporter was idling at a specific coordinate, this coordinate is preserved, allowing the creep to resume its standby state without re-running the expensive base-planning logic that identified the parking spot in the first place.⁵

Behavioral Loops and Demand-Side Validation

A common failure in hauler logic is the "oscillation loop," where a creep repeatedly withdraws energy from a container and immediately puts it back because it lacks a valid downstream target.⁷ Top-tier bots prevent this through "Demand-Side Validation".⁵

Role Separation: Refillers vs. Scavengers

By splitting the logistics role into two distinct sub-roles, the bot eliminates the ambiguity that leads to loops.⁷

Logistics Sub-Role	Responsibility	Idle Behavior
Refiller (Queen)	Taking energy from storage	Idles near the

	to fill structures (extensions, towers). ⁵	storage/terminal to respond to new requests instantly.
Scavenger (Hauler)	Taking energy from the ground or containers to the storage. ⁵	Idles near mining sites or room exits to minimize travel time once a site is full.

By ensuring that a "Refiller" never deposits into a container and a "Scavenger" never withdraws from one, the bot creates a one-way flow of energy that naturally terminates in an idle state when the flow is balanced.⁵

Conclusion: The Convergence of Architecture and Space

The management of "rally points" in top-tier Screeps codebases is not a single function, but the emergent result of several overlapping systems: a kernel-based process model that manages unit intent, a centralized logistics network that brokers resource demand, and a geometric base planner that maps the physical constraints of the environment.² The evolution from independent creeps to a hierarchical hive-mind allows for units to transition seamlessly from active hauling to productive idling, ensuring that every tick of a creep's life and every millisecond of the player's CPU bucket is utilized to its maximum potential.⁴ Whether through the recursive pushing of idle units in a hatchery or the stationary fortification of a bunker by an RCL8 manager, these systems demonstrate a nuanced understanding of how to manage a high-density, multi-agent simulation in real-time.⁵

Works cited

1. Screeps - MMO strategy sandbox game for programmers, accessed February 21, 2026, <https://screeps.com/>
2. Operating System - Screeps Wiki, accessed February 21, 2026, https://wiki.screepspl.us/Operating_System/
3. screepers/ScreepsOS: Screeps/Serializable Operation System (SOS) - GitHub, accessed February 21, 2026, <https://github.com/screepers/ScreepsOS>
4. Screeps #1: Overlord overload | Ben Bartlett, accessed February 21, 2026, <https://bencbartlett.com/blog/screeps-1-overlord-overload/>
5. Releases · bencbartlett/Overmind - GitHub, accessed February 21, 2026, <https://github.com/bencbartlett/Overmind/releases>
6. Screeps Part 7 – A complete Re Write - Adam Laycock, accessed February 21, 2026, <https://alaycock.co.uk/2016/12/screeps-part-7-a-complete-re-write>
7. Need some help with "hauler" logic | Screeps Forum, accessed February 21, 2026, <https://screeps.com/forum/topic/2387/need-some-help-with-hauler-logic>
8. ValkyrX/screeps - GitHub, accessed February 21, 2026,

<https://github.com/ValkyrX/screeps>

9. bencbartlett/Overmind: AI for Screeps, a multiplayer programming strategy game - GitHub, accessed February 21, 2026, <https://github.com/bencbartlett/Overmind>
10. Strange problem with larger creeps that would remain idle. :: Screeps: World Genel Tartışmalar - Steam Community, accessed February 21, 2026, <https://steamcommunity.com/app/464350/discussions/0/13188362663582313/?l=turkish>
11. Creeps - Screeps Documentation, accessed February 21, 2026, <https://docs.screeps.com/creeps.html>
12. Creep | Screeps Wiki | Fandom, accessed February 21, 2026, <https://screeps.fandom.com/wiki/Creep>
13. Questions on roles | Screeps Forum, accessed February 21, 2026, <https://screeps.com/forum/topic/913/questions-on-roles>
14. Bunkers · bencbartlett/Overmind Wiki - GitHub, accessed February 21, 2026, <https://github.com/bencbartlett/Overmind/wiki/Bunkers>
15. Screeps: First Steps in Automating Room Planning | by Jerroyd Moore | Medium, accessed February 21, 2026, <https://medium.com/@jerroydmoore/screeps-first-steps-in-automating-room-planning-6fcb2ec89c1a>
16. Automating Base Planning in Screeps – A Step-by-Step Guide, accessed February 21, 2026, <https://sy-harabi.github.io/Automating-base-planning-in-screeps/>
17. Screeps-AI/README.md at master - GitHub, accessed February 21, 2026, <https://github.com/AmandaRekonwith/Screeps-AI/blob/master/README.md>