The Science Overlord: A Systems Architecture for High-Tier Laboratory Automation and Chemical Sovereignty

The evolution of a Screeps colony from the resource-starved survivalism of the early game to the industrial dominance of Room Controller Level (RCL) 8 is fundamentally defined by the transition from an energy-centric economy to a mineral-based chemical hegemony.[1] While the early stages of development are characterized by the acquisition of energy for population growth and infrastructure, the attainment of RCL 6 introduces the structure laboratory, a device that facilitates the production of complex compounds capable of amplifying creep performance by factors of up to 4.0x.[1] In the competitive meta of high-level autonomous agents, an unboosted creep is an economic and tactical liability; true sovereignty is achieved through the mastery of the laboratory complex and the implementation of a Science Overlord capable of orchestrating recursive stoichiometry and precision logistics within a Kernel-driven operating system.[1] This report provides an exhaustive engineering breakdown of the architectural requirements for automating Screeps laboratories, specifically addressing spatial optimization, recursive dependency management, finite state machine (FSM) design for specialized logistics units, and the integration of military boost protocols.

Foundations of the Chemical Economy: RCL 6-8 Transition

The structural capacity of a laboratory system scales non-linearly with the Room Controller Level, imposing distinct architectural requirements at each phase of a colony's development. At RCL 6, the system is limited to three laboratories, which restricts chemical operations to basic reactions or Tier 1 boosts.[5] The transition to RCL 7 doubles this capacity to six laboratories, enabling the production of Tier 2 compounds, while RCL 8 provides the full complement of ten laboratories, allowing for the maximum parallelization of chemical refining.[2] This progression necessitates a Science Overlord that is not merely a collection of scripts but a persistent process manager within a hierarchical command structure.[1]

The hierarchical decoupling of colony management into specific layers—Overmind, Overseer, Directive, and Overlord—ensures that high-level strategic goals do not interfere with the micro-optimizations required for individual unit performance.[1] Within this framework, the Science Overlord functions as a virtual manager responsible for the entire chemical lifecycle of a room, including mineral mining via the extractor, refining via labs, and distribution via the terminal.[1] By integrating this overlord into a Kernel-driven OS, the system achieves preemption and error isolation, ensuring that a "Lab Jam" or a mineral shortage does not crash the core

defensive or offensive logic.[1]

Laboratory Structure Progression and Capacity

The following table delineates the structural evolution and tactical shift associated with laboratory levels in a high-tier autonomous system:

RCL	Lab Count	Primary Objective	Strategic Shift
6	3	Initial Ghodium production	Transition from energy to initial mineral refining [5]
7	6	Tier 2 compound synthesis	Introduction of Lab-boosted sorties and specialized units [1]
8	10	Tier 3 (T3) mass production	Absolute chemical supremacy and nuke/power creep synergy [1]

The Science Overlord must manage the transition between these levels dynamically. At RCL 6, the three available labs are typically configured in a 2-input, 1-output triangle.[2] At RCL 8, the objective shifts to a 2-input, 8-output parallel array to maximize the ROI of the Scientist creep's travel time.[1]

Algorithmic Spatial Placement: The Physics of Range-2 Constraints

The primary constraint of laboratory chemistry is the geometric requirement of the runReaction intent: a reaction requires three laboratories where two "reagent source" labs are within a range of 2 squares of the "product collector" lab.[2] To maximize throughput at RCL 8, a cluster must be designed such that two centrally located input labs can supply reagents to the remaining eight output labs simultaneously.[1]

The 10-Lab Flower Cluster

The Flower layout is a compact, hexagonal configuration designed for accessibility and minimal travel distance for the logistics creep. In this model, two input labs are placed at the center of the cluster, often referred to as the "Source Pair." Eight output labs are then arranged in a

circular pattern around this core.[1]

For a flower centered at coordinate (x, y), the relative offsets for the input labs and output labs are typically defined to ensure every collector is within range 2 of both reagents. A common implementation places the inputs at (x, y) and $(x + 1, y)$. The remaining eight labs are then positioned at the perimeter of a 5×5 area centered on the input pair. This layout allows the Scientist creep to stand on a single tile—the "Science Hub"—and reach all input labs and several output labs for efficient refilling and collection.[1]

The 10-Lab Diamond Cluster

The Diamond layout is a grid-aligned configuration favored by bots utilizing "Bunker" stamps or 13×13 base layouts.[1] This layout aligns labs in a diagonal or grid pattern that integrates seamlessly into the protective rampart perimeter of a core base. While potentially introducing more travel for the Scientist, the Diamond layout is easier to protect with the "Min-Cut" algorithm, as it avoids the protruding edges often found in the Flower layout.[1]

Layout Feature	Flower Cluster	Diamond Grid	Figure-8 Progression
Input Placement	Centralized / Dual-Hub	Grid-Aligned	Centralized / Linear
Output Capacity	8 (Parallel)	8 (Parallel)	1 to 8 (Scalable)
Scientist Access	1 to 2 tiles	2 to 4 tiles	1 to 2 tiles
Recommended Use	Compact Bases [8]	Bunker-Style Planning [1]	Progressive Growth [5]

Algorithmic Placement using Distance Transform

For a fully autonomous system, manual lab placement is an architectural dead-end. The "Architect" process utilizes the Distance Transform algorithm to identify suitable open spaces during the colony boot phase.[1] The algorithm calculates the distance of every walkable tile from the nearest wall or exit. To place a 10-lab cluster, the Architect targets positions with a distance value of $d \geq 3$, indicating a 5×5 or larger open area.[8]

Once a candidate center point is identified, a Floodfill algorithm is executed to categorize

surrounding tiles by their accessibility to core structures like the Storage and Terminal.[1] Because the Scientist creep must move frequently between the Terminal and the Labs, the logistics cost—defined as the product of distance and transfer frequency—must be minimized. The most efficient systems place the lab cluster immediately adjacent to the Terminal.[1]

Recursive Stoichiometry: Programmatic Dependency Resolving

High-tier chemistry involves multi-tiered reaction trees. To produce a Tier 3 boost such as Catalyzed Ghodium Acid (XGH_2O), the system must first produce base compounds like Ghodium and Hydroxide, then intermediate acids like Ghodium Acid (GH_2O), before the final reaction with Catalyst (X).[4] A Science Overlord must programmatically determine these precursors to ensure the lab complex never idles.[1]

The Reaction Hierarchy and Stoichiometry

Tier 3 production is modeled as a Directed Acyclic Graph (DAG) where nodes represent mineral compounds and edges represent reactions. The stoichiometry of these reactions is generally 1:1, but the reaction cooldown times vary significantly based on the tier of the product.[2]

Compound Tier	Example Formula	Reaction Time (Ticks)	Key Utility
Base	$Z + K \rightarrow$	5	Precursor only [2]
Tier 1	$G + H \rightarrow$	10	Moderate WORK boost [9]
Tier 2	$GH + OH \rightarrow$	15	Significant WORK boost [2]
Tier 3	$GH_2O + X \rightarrow$	80	Maximum efficiency boost [2]

The Science Overlord utilizes a recursive formulation function that takes a target resource and quantity and returns a list of missing precursors.[1] This function must account for the current inventory in both the Terminal and Storage to prevent redundant production. If the bot already possesses 500 units of Ghodium Acid, it should skip the precursor reactions and move directly

to the final Tier 3 stage.[1]

Recursive Dependency Algorithm Logic

The recursive stoichiometry algorithm functions by decomposing a target product into its immediate reagents using the REACTIONS global object.[10] The algorithm then checks if these reagents are available in sufficient quantities. If not, it recursively calls itself for each missing reagent until it reaches base minerals.[4]

The pseudocode for this process involves:

1. **Define Target:** The system identifies a need for 3,000 units of $XGHO$.
2. **Inventory Check:** It queries terminal.store and storage.store.
3. **Decomposition:** $XGHO$ decomposes into X and GHO_2. GHO_2 further decomposes into GH and OH.
4. **Work Allocation:** The system generates a "Lab Order" for the lowest available tier (e.g., $H + O \rightarrow OH$) and assigns it to the reaction scheduler.[1]

State-Machine Design for the Scientist Creep

The Scientist is a specialized logistics creep whose sole function is the movement of minerals between the Terminal, Storage, and Labs. Unlike general haulers that use simple "Pull" logic, the Scientist is a high-precision agent driven by the Science Overlord's current orders.[1] To ensure deterministic behavior and minimize CPU overhead, the Scientist operates on a Finite State Machine (FSM).

Scientist FSM States

The implementation of an FSM avoids the "re-deciding" problem where a creep recalculates its target every tick. The core states include:

- **RENEW:** Moving to a spawn to increase ticksToLive before starting a high-value transfer, preventing the creep from dying while carrying expensive minerals.[1]
- **SUPPLY_REAGENTS:** Withdrawing required quantities of reagents from the Terminal and depositing them into the designated input labs.[1]
- **PRODUCT_COLLECTION:** Withdrawing finished minerals from the output labs once they reach a threshold and returning them to the Terminal.[1]
- **CLEANUP (FLUSHING):** The most critical state. If an order changes or a lab contains the wrong mineral type, the Scientist must purge the labs to prevent a Lab Jam.[1]

The Lab Jam and Flushing Protocols

A Lab Jam occurs when a lab contains a residual amount of a mineral that is not part of the

current reaction order, preventing the input of new reagents.[1] This often happens after a global reset or a change in high-level strategy. The Scientist must be programmed with a "Purge" protocol that scans all ten labs against the current LabOrder.[1] If a lab's store contains an unauthorized resource, the Scientist immediately prioritizes emptying that lab into the Terminal.[1] Failure to implement this cleanup logic is a primary reason why lower-tier lab automation systems fail over long durations.

The Gale-Shapley Logistics Broker for Mineral Management

The economy of a Screeps colony is essentially a flow problem. Energy and minerals are produced at various sources and consumed at centers of refining or defense.[1] To manage this across a multi-room empire, the Science Overlord integrates with a Global Logistics Broker utilizing the Gale-Shapley stable matching algorithm.[1]

Stable Matching in Logistics

The Gale-Shapley algorithm finds a stable matching between two sets of elements (e.g., Haulers and Requests) given an ordering of preferences for each. In the context of mineral logistics, stability is defined as a state where no hauler and no request mutually prefer each other over their current assignment.[1]

The matching process involves:

1. **Registry Phase:** All labs and terminals register their needs (Requesters) or surpluses (Providers) with the Logistics Broker.[1]
2. **Preference Ranking:** Each hauler ranks requests based on a heuristic combining priority (e.g., T3 boosts for defense $= 10$) and linear distance.[1]
3. **Iteration:** The algorithm matches haulers to requests until all units are assigned, ensuring that critical defense minerals are never "starved" by low-priority refining tasks.[1]

Mineral Quotas and Terminal Synchronization

The industry standard for inter-room logistics is the Quota System. Each room's memory defines a target quantity for every resource type. The Logistics Overlord calculates the "Delta"—the difference between the target and the current stock.[1] A positive delta indicates a surplus, while a negative delta indicates a deficit. The system then matches "Providers" with "Requesters" using the stable matching algorithm to minimize the energy cost of the terminal.send command.[1]

The Boost Request Protocol and Military

Synchronization

The ultimate objective of laboratory automation is the application of boosts. A boost is a permanent modification that enhances a specific body part type for the duration of the creep's 1,500-tick life.[2] Modern bots use a "Request" protocol rather than a "Push" system.[1]

Military Boost Reservation

When a creep is spawned, its logic identifies its mission requirements. If it is a siege unit, it might require TOUGH and ATTACK boosts. It enters a BOOSTING state and registers a request with the room's Science Overlord.[1] The Science Overlord then reserves the necessary minerals and energy in the designated labs.[2]

The request-based system allows the Science Overlord to prioritize boosts for high-priority units, such as defenders, over routine industrial production.[1] This is critical during a siege, where the economy must pivot from "Upgrading" to "Combat" boosts in a single tick.

Technical Effects of Tier 3 Boosts

The efficacy of boosts is non-linear when combined with strategic body part sequencing. Damage in Screeps is applied to body parts in the order they appear in the array; therefore, TOUGH parts should always be placed at the front.[1]

Boost Type	Compound	Efficiency Multiplier	Primary Application
Upgrade	XGH_2O	+100% (No extra energy)	Power-leveling GCL [10]
Heal	$XLHO$	+300% Effectiveness	Quad formation survival [1]
Tough	$XGHO$	-70% Damage Taken	Siege tanks and defenders [1]
Attack	XUH_2O	+300% Effectiveness	Dismantling ramparts [1]
Move	$XZHO$	+300% Fatigue reduction	Rapid response and quads [1]

Architectural Resilience: Managing the CPU Crisis

The primary constraint for any automated agent in Screeps is the hard limit of CPU cycles and the overhead of memory serialization.[1] Every function call and memory access is scrutinized for its overhead. The Science Overlord must be designed to minimize these costs.

Heap-First Architecture and Serialization

Modern architectures exploit the persistence of the V8 Heap across ticks.[1] The Science Overlord caches massive amounts of data in global objects, avoiding the expensive JSON.parse costs associated with the Memory object.[1] Ephemeral data, such as current reaction cooldowns or Scientist paths, are never written to Memory; they exist only in the Heap.[1]

CPU Efficiency in Chemical Operations

The StructureLab.runReaction and boostCreep methods each cost 0.2 CPU per intent. In a large empire with dozens of rooms, running 160 reactions per tick (8 per room) would consume 32 CPU—nearly a third of the standard limit for many players.[1] To optimize this, the Science Overlord implements:

1. **Batch Reactions:** Reactions only run if the product is needed or the bucket is full.[1]
2. **Process Suspension:** If a lab has a 160-tick cooldown, the reaction process "sleeps" in the Kernel for 160 ticks, saving the CPU cost of checking it every tick.[1]

Conclusion: Implementing the Science Overlord

The implementation of a top-tier laboratory automation and boosting system represents the pinnacle of Screeps engineering. By adopting a process-driven architecture, optimizing spatial layouts through automated planning, and integrating empire-wide mineral logistics via stable matching, an autonomous agent can achieve chemical supremacy.[1]

For the Senior Systems Architect, the roadmap forward involves the systematic re-engineering of the science pipeline into a set of modular Kernel processes. The Science Overlord must treat every mineral unit as a potential multiplier for empire-wide success. Through the implementation of FSM-driven logistics, recursive stoichiometry dependency solvers, and inter-room terminal balancing, a colony will achieve the tactical and economic sovereignty necessary to dominate the Screeps endgame. Autonomy is no longer just a feature; it is the fundamental metabolic requirement for survival in a persistent, competitive world.[1]

Works cited

1. 2026-02-18 - Screeps Research Combined.pdf
2. Resources | Screeps Documentation, accessed February 19, 2026,

https://docs.screeps.com/resources.html
3. Screeps #1: Overlord overload - Ben Bartlett, accessed February 19, 2026, https://bencbartlett.com/blog/screeps-1-overlord-overload/
4. Screeps #22: For Science | Field Journal, accessed February 19, 2026, https://jonwinsley.com/notes/screeps-for-science
5. Control | Screeps Documentation, accessed February 19, 2026, https://docs.screeps.com/control.html
6. StructureLab - Screeps Wiki, accessed February 19, 2026, https://wiki.screepspl.us/StructureLab/
7. Base Building Considerations : r/screeps - Reddit, accessed February 19, 2026, https://www.reddit.com/r/screeps/comments/87ed6x/base_building_considerations/
8. Automating Base Planning in Screeps – A Step-by-Step Guide, accessed February 19, 2026, https://sy-harabi.github.io/Automating-base-planning-in-screeps/
9. Minerals | Screeps Wiki - Fandom, accessed February 19, 2026, https://screeps.fandom.com/wiki/Minerals
10. Boosts - Screeps Wiki, accessed February 19, 2026, https://wiki.screepspl.us/Boosts/
11. Recursive Chemical Reactions | TDS Archive - Medium, accessed February 19, 2026, https://medium.com/data-science/recursive-chemical-reactions-how-to-algorithmicallyanalyze-chemical-structures-c6a0fab95fa0
12. Creep Boost discussion | Screeps Forum, accessed February 19, 2026, https://screeps.com/forum/topic/471/creep-boost-discussion
13. Understanding game loop, time and ticks - Screeps Documentation, accessed February 19, 2026, https://docs.screeps.com/game-loop.html
14. X/Y offsets positive or negative? : r/klippers - Reddit, accessed February 19, 2026, https://www.reddit.com/r/klippers/comments/1cwlcmu/xy_offsets_positive_or_negative/