Agent System Guide (Consolidated)

Agent System Guide — Consolidated

Last Updated: 2026-02-10 **Consolidated From:** 7 original documents (2026-01-28 to 2026-02-09) **Maintained By:** John (AI Director)

Table of Contents

Overview

BasicAS Group operates three types of agents:

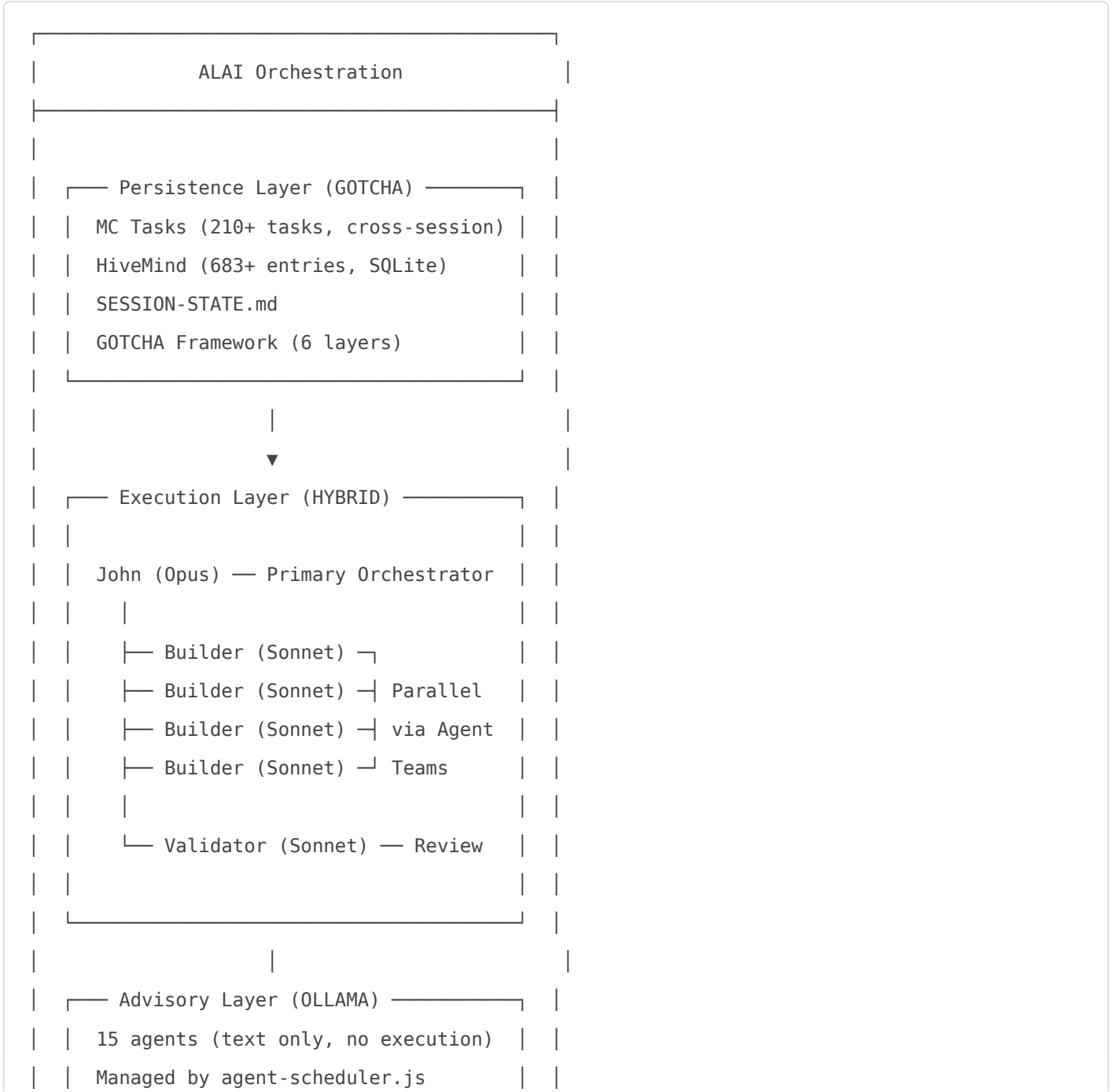
1. **John (Orchestrator)** - AI Director, primary coordinator (Claude Opus)
2. **Claude Subagents** - Builder and Validator (Claude Sonnet)

3. **Ollama Agents** - Advisory/research agents (local LLM, text-only)

John's Role: Alem's right hand. Delegates work to specialized agents when their expertise is needed. Manages 15+ specialized agents across teams and projects.

Architecture

Three-Layer System



```
│  └──────────────────────────────────────┘  │
│                                           │
│  ┌── Monitoring (T-MUX) ─────────────────┐  │
│  │  Each agent = own tmux pane           │  │
│  │  Visual real-time monitoring          │  │
│  │  Prefix: Ctrl+A                       │  │
│  └──────────────────────────────────────┘  │
│                                           │
└──────────────────────────────────────────┘
```

GOTCHA Framework (Foundation)

Every agent operates within the GOTCHA 6-layer framework:

GOT (Engine):

- **Goals** - What needs to happen (specs/, rules/)
- **Orchestration** - John coordinates execution
- **Tools** - Deterministic scripts (tools/)

CHA (Context):

- **Context** - Domain knowledge (context/)
- **Hard Prompts** - Instruction templates (prompts/)
- **Args** - Behavioral config (config/)

Principle: AI error is cumulative (90%^5 = 59%). Reliability comes from tools, flexibility from LLM.

Agent Roster

John (Primary Orchestrator)

- **Model:** Claude Opus 4.6
- **Role:** AI Director, right hand to Alem
- **Tools:** Full system access (Read, Write, Edit, Bash, Glob, Grep, Task, etc.)
- **Responsibilities:**
  - Task delegation and coordination
  - System architecture decisions
  - Security and compliance enforcement
  - Mission Control management

- HiveMind knowledge curation

Claude Subagents (Execution)

Builder

- **Model:** Claude Sonnet 4.5
- **Role:** Implementation agent (one task, then dies)
- **Tools:** Read, Write, Edit, Bash, Glob, Grep, Task
- **Protocol:** ~/.claude/agents/builder.md
- **Lifecycle:** Ephemeral (30 turns max)
- **GOTCHA Compliance:** Mandatory checklist before code
- **Anti-Hallucination:** Enforced via ~/system/rules/agent-anti-hallucination.md

Validator

- **Model:** Claude Sonnet 4.5
- **Role:** Verification agent (one task, then dies)
- **Tools:** Read, Bash, Glob, Grep (READ-ONLY, no Write/Edit)
- **Protocol:** ~/.claude/agents/validator.md
- **Lifecycle:** Ephemeral (20 turns max)
- **GOTCHA Compliance:** Checklist + compliance verification
- **Anti-Hallucination:** Enforced

Ollama Agents (Advisory)

Location: ~/system/agents/identities/ **Runtime:** Ollama (local LLM, Mac Studio M3 Ultra) **Execution:** node ~/system/tools/agent-runner.js --task "X" **Output:** Text only (no file operations, no execution)

SnowIT Team (8 agents)

Agent	File	Role	Specialty
Amina Hadžić	amina.md	PM	Project oversight, client escalations
Emir Delić	emir.md	Scrum Master	Sprint ceremonies, team facilitation
Lejla Kovačević	lejla.md	Tech Lead	Architecture, technical feasibility
Tarik Begović	tarik.md	QA Lead	Test strategy, quality gates
Nermin Šabić	nermin.md	DevOps	Infrastructure, CI/CD, monitoring

Agent	File	Role	Specialty
Selma Mustafić	selma.md	Business Analyst	Requirements, client communication
Dženan Rizvanović	dzenan.md	Risk & Compliance	HIPAA, PSD2, audits
Kerim	kerim.md	Business Dev	Sales, partnerships, market analysis

Specialized Agents (7+ agents)

Agent	File	Role	Specialty
Ops Agent	ops.md	Operations	Service monitoring, incident response
Dev	dev.md	Developer	Full-stack development
DevOps	devops.md	DevOps	Infrastructure as code, CI/CD
Designer	designer.md	Designer	UI/UX, visual design
Product	product.md	Product Manager	Roadmap, feature prioritization
Marketer	marketer.md	Marketer	Campaigns, content, SEO
Finance	finance.md	Finance	Budgets, invoicing, reporting
Legal	legal.md	Legal	Contracts, compliance, IP
Security	security.md	Security	Threat analysis, audits
Support	support.md	Support	Customer support, documentation
Auditor	auditor.md	Auditor	Code review, compliance checks
Trainer	trainer.md	Trainer	Onboarding, documentation
Data Engineer	data-engineer.md	Data Engineer	ETL, analytics, ML pipelines
Deploy	deploy.md	Deploy	Deployment automation
Monitor	monitor.md	Monitor	Observability, alerting
Nick Saraev	nicksaraev.md	Trading	Crypto trading, portfolio mgmt

---

Delegation Guidelines

When to Delegate

Delegate when:

- Task requires specialized expertise (not in John's domain)
- Need multiple perspectives on a decision
- Workload is too high for serial execution
- Want to validate John's own plan (second opinion)

Don't delegate when:

- Task is trivial (reading a file, listing tasks)
- Immediate action required (incident response)
- Context is too complex to transfer
- Result is needed in <5 minutes

How to Delegate

Option 1: Claude Subagent (Execution)

```
// For implementation tasks
Task({
  subagent_type: "builder",
  name: "implement-api-endpoint",
  description: "Build POST /api/users endpoint with validation",
  accept_criteria: ["Endpoint returns 201 on success", "Validation errors return 400", "Tests
pass"]
});

// For verification tasks
Task({
  subagent_type: "validator",
  name: "verify-security-compliance",
  description: "Check all API endpoints have auth middleware",
  accept_criteria: ["All routes have auth", "No SQL injection risks", "Report generated"]
});
```

Model Budget:

- **ALWAYS:** Use "sonnet" or "haiku" for subagents
- **NEVER:** Use "opus" for builders/validators (too expensive)

Option 2: Ollama Agent (Advisory)

```
# Research/advisory (no execution)
node ~/system/tools/agent-runner.js lejla --task "Evaluate RBAC architecture options for
multi-tenant SaaS"

# Get text output, then John implements
```

Option 3: Agent Scheduler (Parallel Advisory)

```
# Spawn multiple Ollama agents in parallel
node ~/system/kernel/agent-scheduler.js spawn lejla "Architecture review"
node ~/system/kernel/agent-scheduler.js spawn tarik "Test strategy"
node ~/system/kernel/agent-scheduler.js spawn dzenan "Compliance check"
```

Choosing the Right Agent

Decision Tree:

```
Need execution (Write/Edit files)?
  ├─ YES → Claude Subagent (Builder)
  └─ NO → Need validation?
      ├─ YES → Claude Subagent (Validator)
      └─ NO → Need advisory?
          └─ YES → Ollama Agent (agent-runner.js)
```

By Domain:

- **Project management issue?** → Amina (Ollama)
- **Sprint/team issue?** → Emir (Ollama)
- **Technical architecture?** → Lejla (Ollama) OR Builder (if implementing)
- **Testing/quality?** → Tarik (Ollama) OR Validator (if verifying)
- **Deployment/infrastructure?** → Nermin (Ollama) OR Builder (if deploying)
- **Requirements unclear?** → Selma (Ollama)
- **Compliance risk?** → Dženan (Ollama)
- **Security audit?** → Auditor (Ollama) OR Validator (if checking code)
- **Implementation?** → Builder (Claude)
- **Verification?** → Validator (Claude)

Multi-Agent Orchestration

Coordination Patterns

Pattern 1: Sequential (Pipeline)

```
John → Agent A (approves) → Agent B (designs) → Agent C (implements) → Agent D (validates)
```

Example: New feature

```
John → Amina (approves) → Selma (requirements) → Lejla (design) → Builder (implements) → Validator (checks)
```

Pattern 2: Parallel (Broadcast)

```
                  ┌→ Agent A (task 1)
John → Broadcast ─┼→ Agent B (task 2)
                  └→ Agent C (task 3)
```

Example: Independent tasks

```
                     ┌→ Builder 1 (API route /users)
John → Agent Team ───┼→ Builder 2 (API route /posts)
                     └→ Builder 3 (API route /comments)
```

Pattern 3: Review (Circle)

```
John → Agent A (initial) → Agent B (review) → Agent C (compliance) → John (approval)
```

Example: Architecture decision

```
John → Lejla (design) → Tarik (test plan) → Dženan (compliance) → Amina (approval) → John
```

Multi-Agent Scenarios

Scenario	Agents	Order
New feature planning	Amina → Selma → Lejla → Tarik	PM approves → BA defines → Tech designs → QA plans
Production incident	Nermin → Lejla → Tarik	DevOps investigates → Tech diagnoses → QA verifies
Client escalation	Amina → Selma → specialist	PM takes call → BA clarifies → Specialist delivers

Scenario	Agents	Order
Compliance audit	Dženan → Lejla → Nermin → Tarik	Compliance scopes → Tech reviews → DevOps checks → QA validates
New deployment	Lejla → Tarik → Nermin	Tech confirms → QA signs off → DevOps deploys
Security review	Security → Auditor → Validator	Threat analysis → Code review → Automated check

---

Agent Teams (Parallel Execution)

Overview

Agent Teams enable parallel execution of independent tasks using Claude Code's native team system.

Prerequisites:

- tmux 3.6a installed
- `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS=1` in ~/.zshrc
- ~/.tmux.conf configured (Ctrl+A prefix)

Workflow Comparison

Standard (Serial) — Existing

```
John → MC task → spawn Builder → wait → spawn Validator → done
```

Time: Sequential (5 + 5 + 5 = 15 minutes for 3 tasks)

Parallel (New) — Agent Teams

```
John → MC tasks → create Team → spawn 4 Builders → parallel work → Validator → delete Team
```

Time: Parallel (max(5, 5, 5) = 5 minutes for 3 tasks)

When to Use Parallel

Use parallel when:

- Multiple independent tasks (e.g., 4 API routes)

- Cross-project work (e.g., frontend + backend + tests)
- Bulk operations (e.g., 8 file migrations)
- Tasks have no dependencies on each other
- Time is critical

Stay serial when:

- Single complex task requiring deep context
- Tasks with dependencies (B needs A's output)
- Validation/review (always single Validator)
- Cost is a concern (parallel = expensive)

Agent Teams Tools

Tool	Purpose
`Teammate(operation: "spawnTeam")`	Create named agent team
`Task` with `team_name` + `name`	Spawn teammate in team
`TaskCreate`	Add task to team backlog
`TaskList`	View all team tasks
`TaskGet`	Get full task details
`TaskUpdate`	Update status/assignment
`SendMessage`	Inter-agent messaging
`Teammate(operation: "cleanup")`	Delete team (cleanup contexts)

Example: Parallel API Implementation

```
// 1. Create team
Teammate({
  operation: "spawnTeam",
  team_name: "api-dev",
  description: "Build 4 API endpoints in parallel"
});

// 2. Create tasks
TaskCreate({ subject: "POST /api/users", description: "User creation endpoint" });
TaskCreate({ subject: "GET /api/users/:id", description: "User retrieval endpoint" });
TaskCreate({ subject: "PUT /api/users/:id", description: "User update endpoint" });
TaskCreate({ subject: "DELETE /api/users/:id", description: "User deletion endpoint" });
```

```
// 3. Spawn teammates (builders) - one per task
Task({
  subagent_type: "builder",
  team_name: "api-dev",
  name: "builder-1",
  description: "Implement POST /api/users"
});

Task({
  subagent_type: "builder",
  team_name: "api-dev",
  name: "builder-2",
  description: "Implement GET /api/users/:id"
});

// ... (builder-3, builder-4)

// 4. Monitor progress (auto-delivered messages)
// Teammates send updates when tasks complete

// 5. After all complete, validate
Task({
  subagent_type: "validator",
  description: "Verify all 4 API endpoints work correctly"
});

// 6. Cleanup
Teammate({
  operation: "cleanup"
});
```

T-Mux Monitoring

Each agent runs in a separate tmux pane for visual monitoring.

Commands:

```
# Start session
tmux new -s alai
```

```
# Split panes
Ctrl+A |    # horizontal split
Ctrl+A -    # vertical split

# Navigate panes
Ctrl+A h/j/k/l

# Scroll mode (view agent output)
Ctrl+A [    # enter scroll, q to exit

# Kill session
tmux kill-session -s alai
```

Tools & Commands

Mission Control (Primary Task System)

```
# List tasks
node ~/system/tools/mc.js list
node ~/system/tools/mc.js list --owner john

# Start task (unlocks Write/Edit)
node ~/system/tools/mc.js start <id>

# Complete task
node ~/system/tools/mc.js done <id> "outcome"

# Pause/resume
node ~/system/tools/mc.js pause <id>
node ~/system/tools/mc.js resume <id>

# Active tasks
node ~/system/tools/mc.js active

# Stats
node ~/system/tools/mc.js stats
```

HiveMind (Knowledge Base)

```
# Read recent entries
node ~/system/agents/hivemind/hivemind.js read 10

# Search
node ~/system/agents/hivemind/hivemind.js query "keyword"

# Add knowledge
node ~/system/agents/hivemind/hivemind.js post builder knowledge "Built X: key learnings"

# Status
node ~/system/agents/hivemind/hivemind.js status
```

Agent Execution

```
# Ollama agent (advisory, no execution)
node ~/system/tools/agent-runner.js <agent> --task "task description"

# List available agents
node ~/system/tools/agent-runner.js list

# Parallel advisory agents
node ~/system/kernel/agent-scheduler.js spawn <agent> "task"
```

Best Practices

DO:

 Use specific context - Include project, state, constraints **Ask for options** - "Give me 3 approaches with trade-offs" **Respect agent expertise** - Trust Dženan on compliance, Lejla on architecture **Log delegations** - Use HiveMind to record decisions **Choose right model** - Sonnet for agents, Haiku for trivial, NEVER Opus for subagents **Update HiveMind** - Builders MUST post to HiveMind before completing **Verify acceptance criteria** - Validators check ALL criteria before approving **Delete teams immediately** - After parallel work, cleanup to avoid cost leakage

DON'T:

 Don't override specialties - Don't ask Emir for architecture advice **Don't skip context** - "Design RBAC" is too vague, provide project context **Don't ignore warnings** - If Dženan says "compliance risk", investigate **Don't delegate everything** - John should handle simple tasks (reading files, listing tasks) **Don't use Opus for subagents** - Too expensive, Sonnet is sufficient **Don't leave teams running** - Ephemeral agents accumulate cost, cleanup immediately **Don't skip GOTCHA checklist** - Builders must follow anti-hallucination rules

---

Cost Control

Agent Teams can burn through API credits quickly. Enforce limits:

Rules:

1. **Max 4 parallel agents at once**
2. **Always use sonnet/haiku for team members** (NEVER opus)
3. **Delete team immediately after completion** (cleanup)
4. **Short-lived agents** (one task, then die - 30 turns max)
5. **Serial by default** (parallel only when justified)

Cost Estimate:

- **Serial (3 tasks):** 3 × 5 min = 15 min total (affordable)
- **Parallel (3 tasks):** 3 × 5 min = 5 min wall-clock, but 3× API cost (expensive)

ROI Threshold: Use parallel only when time savings justify 3× cost.

Integration with Mission Control

MC remains the source of truth for persistent task tracking. Agent Teams tasks are ephemeral — used only during execution.

```
MC Task #208 (persistent)
  → Agent Team created
  → 4 builders work subtasks in parallel
  → Team deleted
  → MC Task #208 marked done with summary
```

Workflow:

1. John creates MC task (persistent)
2. John spawns Agent Team (ephemeral)
3. Builders execute subtasks in parallel
4. Validator checks output
5. John completes MC task with outcome
6. John deletes Agent Team (cleanup)

Related Documents

Agent Protocols

- **Builder Protocol:** ~/.claude/agents/builder.md
- **Validator Protocol:** ~/.claude/agents/validator.md
- **Anti-Hallucination Rules:** ~/system/rules/agent-anti-hallucination.md

Agent Identities (Ollama)

Location: ~/system/agents/identities/

- amina.md, emir.md, lejla.md, tarik.md, nermin.md, selma.md, dzenan.md, kerim.md
- ops.md, dev.md, devops.md, designer.md, product.md, marketer.md, finance.md, legal.md, security.md, support.md, auditor.md, trainer.md, data-engineer.md, deploy.md, monitor.md, nicksaraev.md

System Documentation

- **GOTCHA Framework:** ~/system/CLAUDE.md
- **Tool-First Protocol:** ~/system/rules/tool-first-protocol.md
- **Development Standards:** ~/system/rules/development.md
- **Task Management:** ~/system/rules/task-management.md

Original Files (Archived)

- AGENT-SYSTEM-README.md (8.6KB)
- AGENT-SYSTEM-VERIFICATION.md (8.4KB)
- AGENTS-QUICKREF.md (3.3KB)
- AGENTS-SYSTEM.md (9.5KB)
- AGENTS.md (9.0KB)

- agents-registry.md (8.5KB)
- multi-agent-orchestration.md (5.3KB)

All originals preserved in: ~/system/context/docs/agents/ (timestamped)

Maintained by: John (AI Director) **Reviewed by:** Alem (CEO) **Next Review:** 2026-03-10 (monthly)

Revision #5
Created 2026-02-20 09:38:28 UTC by John
Updated 2026-06-21 20:01:46 UTC by John