(Consolidated)

SPL Multi-Agent Guide v3.1 (Consolidated)

State-Grounded Coordination & Agent-to-Agent Communication

Author: Pamela Cuce

Email: pamela@dasein.works

Phone: 914-240-3564

Version: 3.1 (Consolidated & Corrected)

Date: December 3, 2025

Overview

SPL isn't just for single agents. The real power emerges when multiple SPL agents communicate and coordinate using:

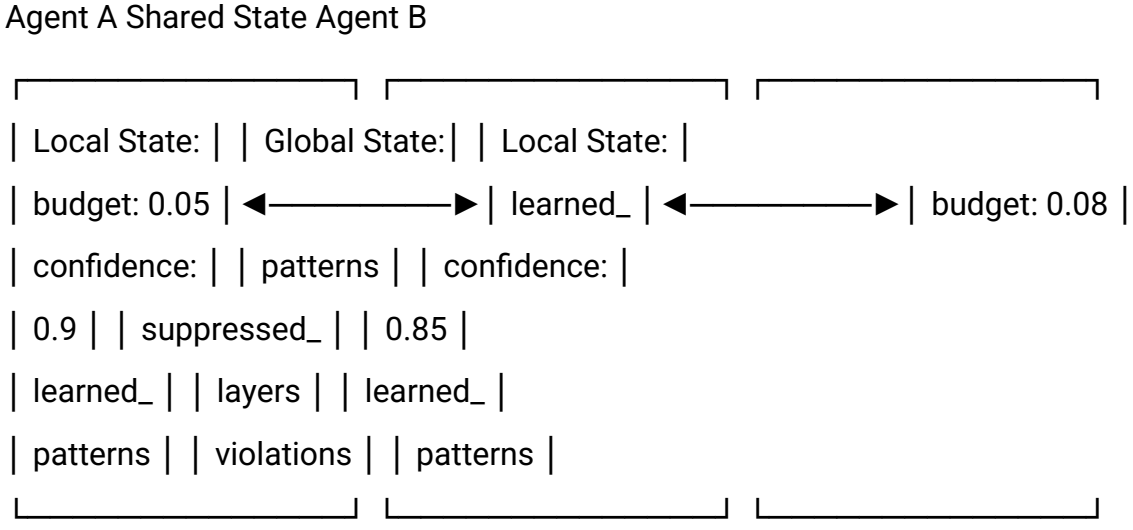
1. **State-grounded control:** Agents share explicit world state (no guessing)
2. **Cross-agent suppression:** Lower-layer agents can suppress upper-layer operations network-wide
3. **Pattern sharing:** Learn once, reuse across entire agent team
4. **Network-wide safety:** Single violation halts all agents immediately

Result: Single-agent SPL achieves 5-15x cost reduction. Multi-agent SPL achieves 10-50x reduction through network effects.

Part 1: State-Grounded Coordination

Core Concept

With state-grounded control, multiple agents coordinate by sharing explicit world state. Agents don't guess what others know—they read state directly.



Pattern Sharing Protocol

Agent A Learns Pattern:

1. Agent A processes 5 emails
2. Learns: "urgent_emails → high_priority" (0.95 confidence)
3. Publishes to shared state: `learned_patterns["urgent"] = 0.95`

Agent B Uses Shared Patterns:

1. Agent B receives email
2. Reads shared state: `learned_patterns["urgent"] = 0.95`
3. Checks: "Is this email urgent?" → YES
4. **STATE SUPPRESSES Layer 2** (LLM call)
5. Uses cached result from Agent A's learning
6. **Cost: $0** (vs $0.01 for LLM)

Part 2: Shared State Architecture

What Lives in Shared State?

Shared state structure

shared_state = {

Patterns learned by any agent

'learned_patterns': {

'urgent': {'confidence': 0.95, 'learned_by': 'agent_a', 'learned_at': '2025-12-03T17:00:00'},

'billing': {'confidence': 0.92, 'learned_by': 'agent_c', 'learned_at': '2025-12-03T17:05:00'}

},

```
text

# Which layers are currently suppressed?
'suppressed_layers': {
    'agent_a':,  # Agent A's Layer 2 is suppressed[1]
    'agent_b': [],   # Agent B has no suppressions
    'agent_c':   # Agent C's Layers 1 & 2 are suppressed[2][1]
},

# Safety violations detected
'violations': [
    {'agent': 'agent_b', 'violation': 'budget_exceeded',
'timestamp': '2025-12-03T17:10:00'}
],

# Network-wide cost tracking
'cost_tracking': {
    'total_network_cost': 15.50,
    'total_suppressions': 450,
    'total_llm_calls': 50,
```

```
    'cost_by_agent': {
        'agent_a': 5.00,
        'agent_b': 5.25,
        'agent_c': 5.25
    }
}
```

}

```
### Reading Shared State
```

Pattern reading

agent_b_patterns = shared_state['learned_patterns']

Result: {'urgent': {...}, 'billing': {...}}

Suppression status

agent_a_suppressed = shared_state['suppressed_layers'].get('agent_a', [])

Result: - Layer 2 is suppressed for Agent A

Safety check

violations = shared_state['violations']

Result: [...] - All violations on network

```
text
```

```
---
```

```
## Part 3: Cross-Agent Suppression
```

```
### Scenario: Budget Alert Triggers Network Suppression
```

```
**Situation:** Agent B's budget exceeded ($100/day limit
reached)
```

```
**Response:**
```

Agent B detects budget exceeded

if agent_b.total_cost > 100.0:

Publish violation to shared state

shared_state['violations'].append({

'agent': 'agent_b',

'violation': 'budget_exceeded',

'timestamp': now()

})

```
text
```

```
# Signal all agents to suppress Layer 2
shared_state['suppressed_layers']['agent_b'] =[1]

# All agents check shared state
# Agent A: "Agent B's Layer 2 is suppressed"
# Agent C: "Agent B's Layer 2 is suppressed"
# → No more expensive LLM calls for Agent B

text


### Broadcasting Layer Suppressions
```

In main loop for each agent

def check_network_suppressions(agent_id, shared_state):

"""

Check if this agent should suppress any layers

```
text

Args:
    agent_id: this agent's ID (e.g., 'agent_a')
    shared_state: network-wide state

Returns:
    list: layers to suppress (,,, or [])[2][1]
```

```
Logic:
1. Check violation list
2. If violation affects this agent, suppress that layer
3. Return list of suppressed layers
"""

suppressed_layers = []

# Check all violations
for violation in shared_state['violations']:
    if violation['agent'] == agent_id:
        # This violation affects us!
        if violation['violation'] == 'budget_exceeded':
            # Budget exceeded - suppress expensive Layer 2
            suppressed_layers.append(2)

        elif violation['violation'] == 'safety_concern':
            # Safety issue - suppress everything
            suppressed_layers.extend()[2][1]

# Update shared state
shared_state['suppressed_layers'][agent_id] = suppressed_layers

return suppressed_layers
```

text

Part 4: Pattern Learning & Sharing

Individual Agent Learning

Agent A's Learning Process:

Iteration 1: Email "URGENT: Meeting" → Call LLM → Learns "urgent" pattern
Iteration 2: Email "URGENT: Budget" → Pattern matches → SUPPRESS Layer 2
...
Iteration 10: Email "URGENT: Hiring" → Pattern matches → SUPPRESS Layer 2

Cost: 1 LLM call (Iteration 1) + 9 cached (Iterations 2-10) = $0.01
Savings: 90%

```
text
```

Network Pattern Learning

Multi-Agent Learning Process:

Agent A processes emails 1-100
→ Learns 10 patterns (costs: $0.10)
→ Publishes to shared state

Agent B processes emails 101-200

→ Uses Agent A's 10 patterns

→ Learns 5 new patterns

→ Publishes to shared state

→ Cost: $0.05

Agent C processes emails 201-300

→ Uses patterns from Agents A & B

→ Learns 2 new patterns

→ Cost: $0.02

Total network cost: $0.17

Per-email cost: $0.00057 (vs $0.01 baseline)

Cost reduction: 17.6x

Part 5: Multi-Agent Execution Example

Email Processing Pipeline

Initial State:

Agent A: budget=$0.05, confidence=0.0

Agent B: budget=$0.10, confidence=0.0

Agent C: budget=$0.08, confidence=0.0

Global: learned_patterns={}

Iteration 1 (Agent A):

Processes: "URGENT: Meeting moved"

Layer 0: Valid ✓

Layer 1: No learned patterns → Escalate

Layer 2: LLM categorizes → "urgent"

→ Publishes: global_patterns["urgent"] = 0.92

→ Cost: $0.01

Iteration 2 (Agent B):

Processes: "URGENT: Budget review"

Layer 0: Valid ✓

Layer 1: Checks global patterns → "URGENT" matches!

Reads: learned_patterns["urgent"] = 0.92

→ STATE SUPPRESSES Layer 2

→ Uses cached pattern (no LLM call)

→ Cost: $0 (saved $0.01)

Iteration 3 (Agent C):

Processes: "URGENT: Hiring decision"

Layer 0: Valid ✓

Layer 1: Checks global patterns → "URGENT" matches

Reads: learned_patterns["urgent"] = 0.92

→ STATE SUPPRESSES Layer 2

→ Uses cached pattern (no LLM call)

→ Cost: $0 (saved $0.01)

Final State:

Total emails: 3

LLM calls: 1 (only Agent A)

Cached calls: 2 (Agents B, C used Agent A's learning)

Total cost: $0.01 (vs $0.03 if all used LLM)

Savings: 67%

Part 6: Network-Wide Cost Tracking

class NetworkCostTracker:
def init(self):

self.costs_by_agent = {}

self.costs_by_layer = {}

self.network_savings = 0.0

def record_execution(self, agent_id, layer, cost, method):

"""Record cost for one execution"""

if agent_id not in self.costs_by_agent:

self.costs_by_agent[agent_id] = 0.0

self.costs_by_agent[agent_id] += cost

if layer not in self.costs_by_layer:

self.costs_by_layer[layer] = 0.0

self.costs_by_layer[layer] += cost

```
def record_network_savings(self, saved_cost, reason):
    """Track network-wide savings"""
    self.network_savings += saved_cost


def report(self):
    """Generate comprehensive report"""
    total = sum(self.costs_by_agent.values())
    return {
        "costs_by_agent": self.costs_by_agent,
        "costs_by_layer": self.costs_by_layer,
        "total_network_savings": self.network_savings,
        "efficiency": self.network_savings / (total + self.network_savings) if total > 0 else 0
    }
```

Part 7: Multi-Agent Learning Rate

Single Agent:

100 emails

Learns: 10 patterns

Cost: $0.50

Multi-Agent (5 agents, 100 emails each):

500 emails total
Shared: 15 patterns (network-wide learning)

Cost if isolated: $0.50 × 5 = $2.50
Cost with sharing: $0.50 (first agent) + $0.00 × 4 = $0.50

Savings: $2.00 (80% reduction)

Why Network Learning is 20x Faster:

- Agent A learns pattern in iteration 10

- Agents B, C, D, E use it immediately in iteration 11

- No redundant LLM calls for same pattern

- Pattern library grows 5x faster than single agent

Part 8: Failure Modes & Recovery

Failure Mode 1: Shared State Unavailable

Symptom: Redis/database goes down

Impact: Agents can't read shared patterns

Recovery:

Fallback to local patterns only

```
def execute_with_fallback(agent, request, shared_state):
"""Execute with graceful fallback"""

try:
    # Try to use shared patterns
    shared_patterns = shared_state['learned_patterns']
except Exception:
    # Shared state unavailable - use local only
    shared_patterns = {}

# Combine local + shared patterns
all_patterns = {**agent.local_patterns, **shared_patterns}

# Continue processing
return process_with_patterns(request, all_patterns)
```

Failure Mode 2: Inconsistent Pattern Confidence

Symptom: Agent A learned pattern with 0.92 confidence, but it's wrong 20% of time

Impact: Layer 1 makes incorrect decisions at scale

Recovery:

Confidence validation

```
def validate_pattern_confidence(pattern_name, pattern_info):
"""Validate pattern confidence"""

confidence = pattern_info['confidence']
```

```
# If confidence dips below threshold, flag for review
if confidence < 0.85:
    # Alert: Pattern may be inaccurate
    # Option 1: Lower suppression (escalate to Layer 2 more often)
    # Option 2: Remove pattern entirely
    # Option 3: Retrain with more examples
    return False  # Don't use this pattern

return True  # Pattern is reliable
```

Failure Mode 3: Violation Broadcast Delays

Symptom: Budget exceeded but not all agents know yet

Impact: Some agents still calling expensive LLM

Recovery:

Aggressive violation checking

```
def critical_check_violations(agent_id, shared_state):
"""
Check for critical violations frequently

Run this every N requests (e.g., every 10 requests)
"""

critical_violations = [
    'budget_exceeded',
    'safety_concern',
```

```
    'rate_limit_exceeded'
]

for violation in shared_state['violations']:
    if violation['violation'] in critical_violations:
        if violation['agent'] == agent_id:
            # IMMEDIATE response
            # Suppress all expensive operations now
            return True  # Should halt

return False  # OK to continue
```

Conclusion

Single-agent SPL: 5-15x cost reduction
Multi-agent SPL: 10-50x cost reduction (80%+ for large networks)

This is the power of explicit state-grounded control: agents don't repeat learning; they build on each other's discoveries.

Key Benefits:

✅ **Compositional agents** (small, focused agents)
✅ **Cross-agent optimization** (suppress expensive agents)
✅ **Hierarchical control** (Layer 0 agents suppress higher layers globally)
✅ **Unprecedented cost efficiency** (10-50x reduction)
✅ **MCP-native** (agents discover & call each other via MCP)

Pamela Cuce | pamcuce@alum.vassar.edu | Version 3.1 (Consolidated & Corrected) | December 3, 2025