Context Engineering

Stop Agents from Choking on Their Own Data

Tres fallas que desperdician tokens — y las soluciones

Elizabeth Fuentes Leone

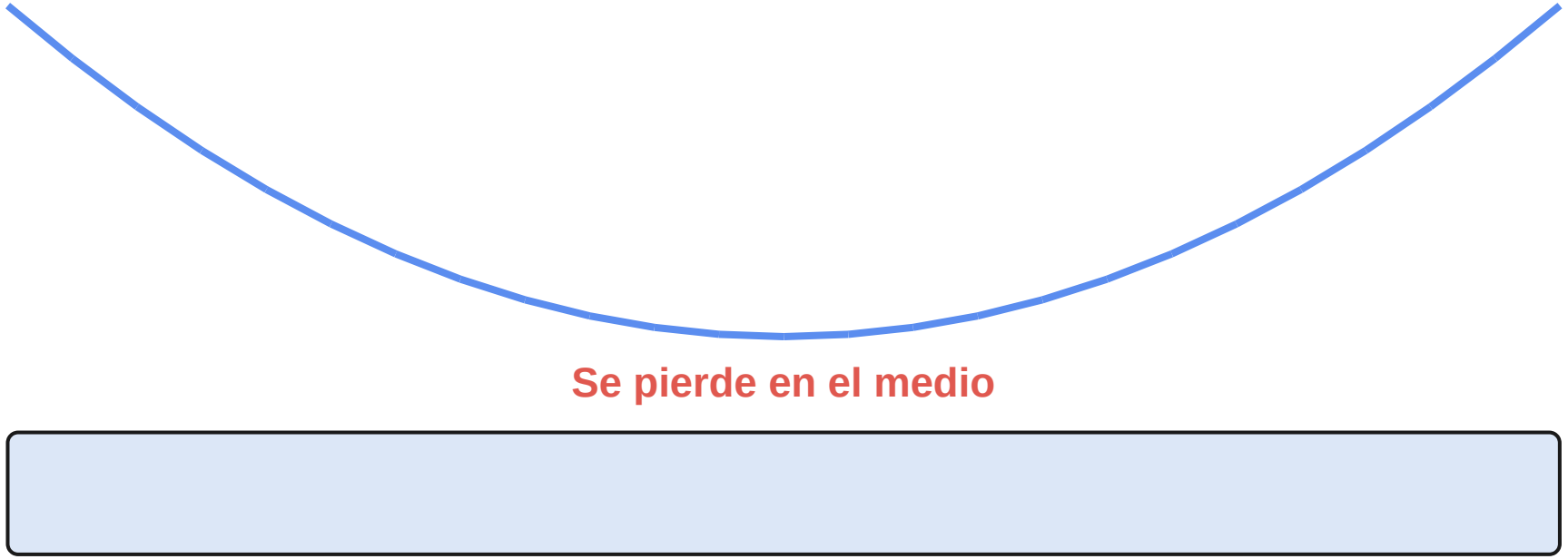
Developer Advocate, GenAI — AWS

4K → 32K → 128K → 1M → 2M tokens

“Solo agrega más tokens, eso lo arregla”

Por qué no funciona

Atención (curva en U)

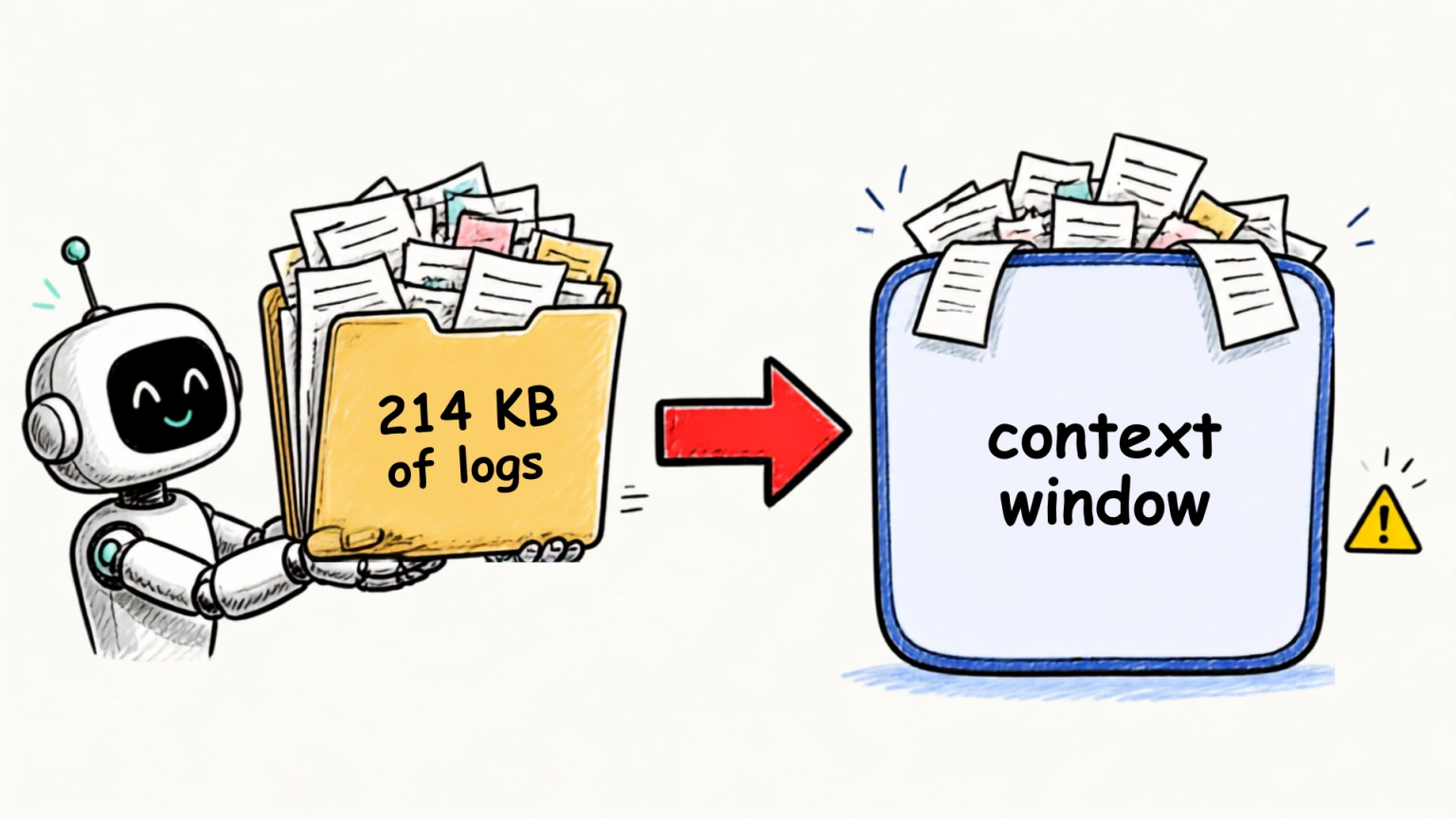


Se pierde en el medio

Ventana de contexto (tokens)

Más tokens ≠ más señal útil

El disparador: una tool devuelve datos grandes



Nunca se lanza un error. El agente simplemente responde peor.

Context Engineering

Darle al modelo la información que necesita, cuando la necesita.

Curar la ventana de contexto para mejores resultados y menor costo de tokens.

Cuatro estrategias de Context Engineering

Externalize

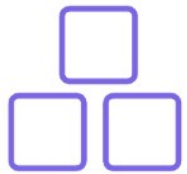
Mueve datos grandes
a
storage persistente

Select

Recupera solo
lo relevante

Compress

Reduce tokens:
resume, compacta



Isolate

Separa el contexto
entre agentes

Esta charla vive sobre todo en Externalize + Select — el patrón Memory Pointer.

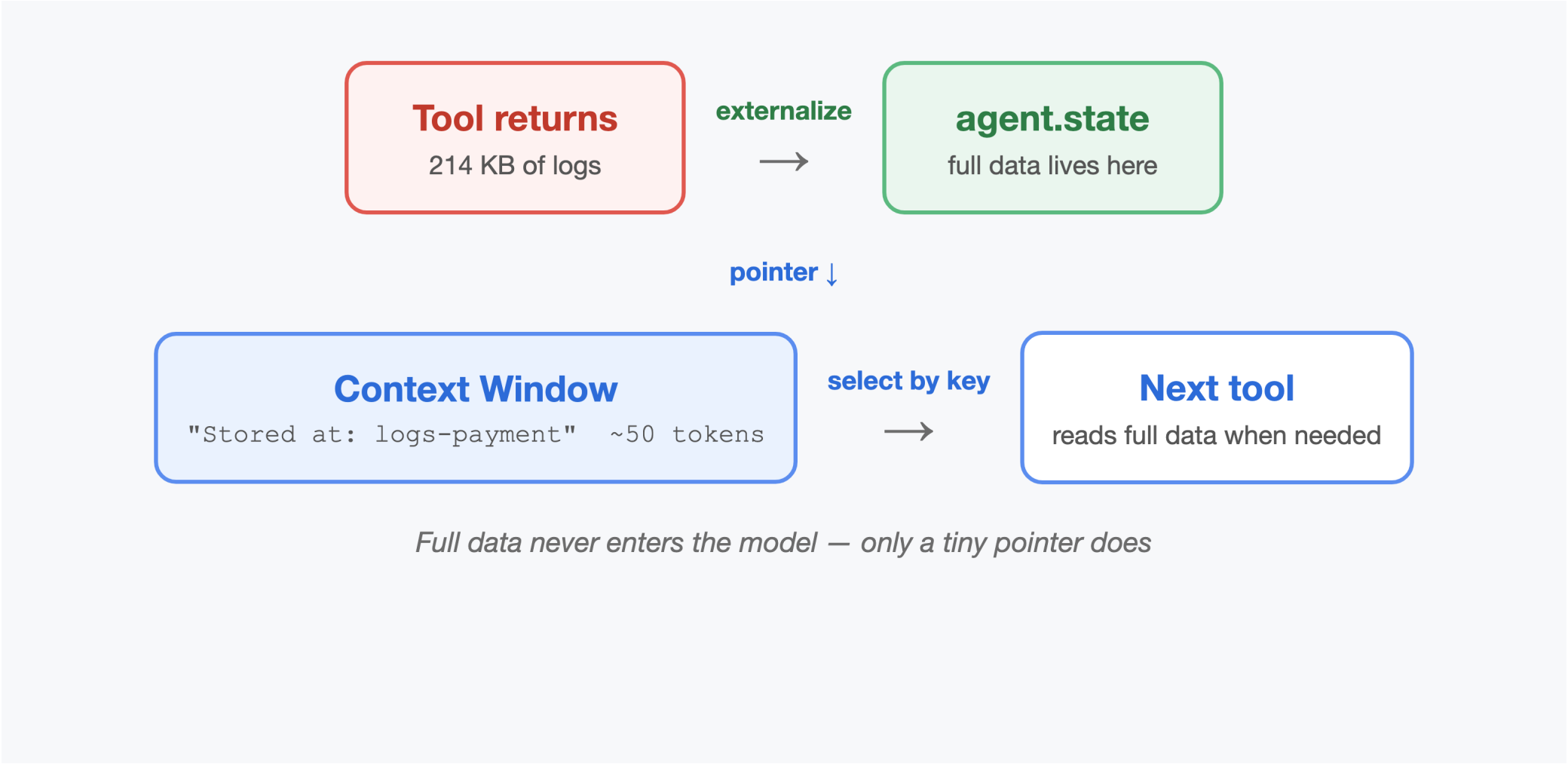
El punto de partida: Conversation Management

Compress

El default incorporado — antes de cualquier patrón avanzado

El patrón Memory Pointer

Externalize Select



Full data never enters the model — only a tiny pointer does

Externalize

Select

```python
@tool(context=True)
def fetch_application_logs(app_name, tool_context):
    logs = generate_logs(hours)      # ~214KB
    pointer = f"logs-{app_name}"
    # Externalize → agent.state (un solo agente)
    tool_context.agent.state.set(pointer, logs)
    return f"Stored at: {pointer}"   # ~50 tokens

@tool(context=True)
def analyze_error_patterns(logs_pointer, tool_context):
    # Select → lee el dato completo solo cuando hace falta
    logs = tool_context.agent.state.get(logs_pointer)
```

STRANDS AGENTS SDK

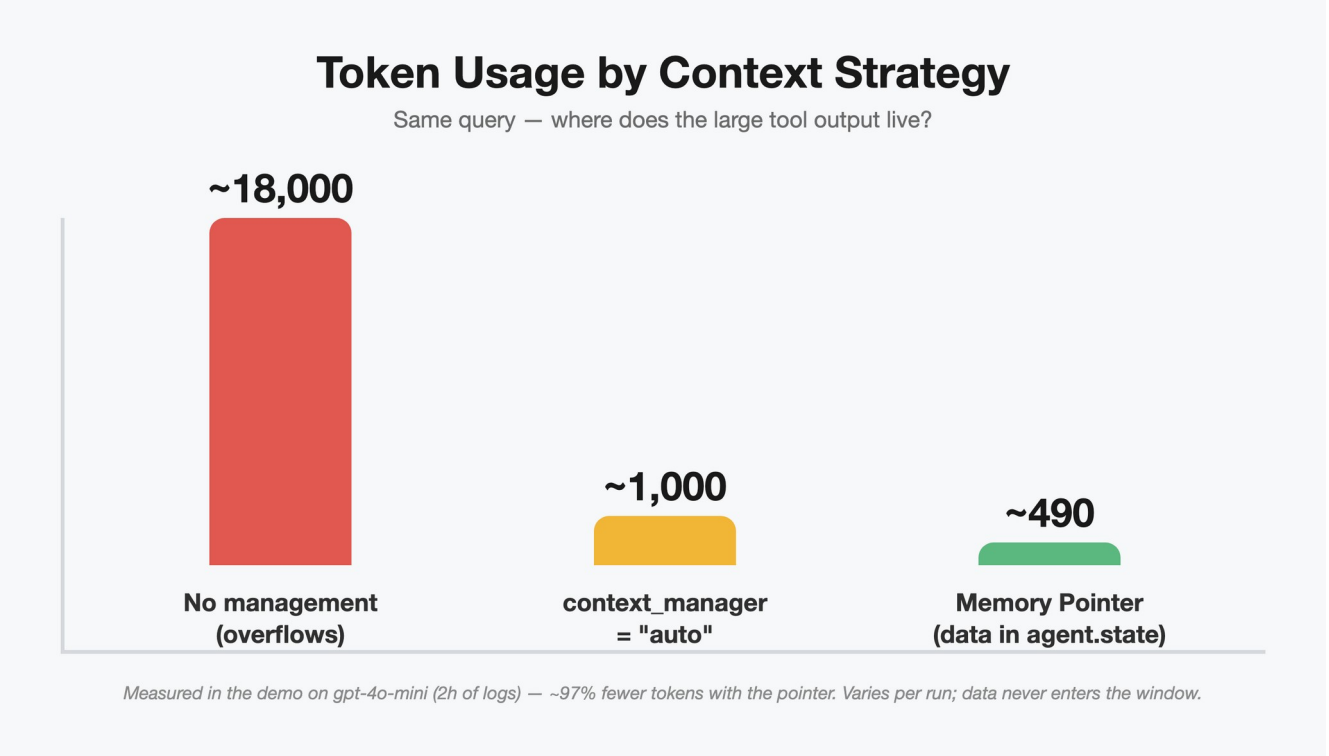
Resultado — un solo agente

Externalize Select

~97%

menos tokens

sin perder información



Externalize

Cuando se cierra la sesión: ¿dónde vive el dato?

Mismo patrón, cambia el backend de storage — la referencia dura lo que dure lo que hay detrás.

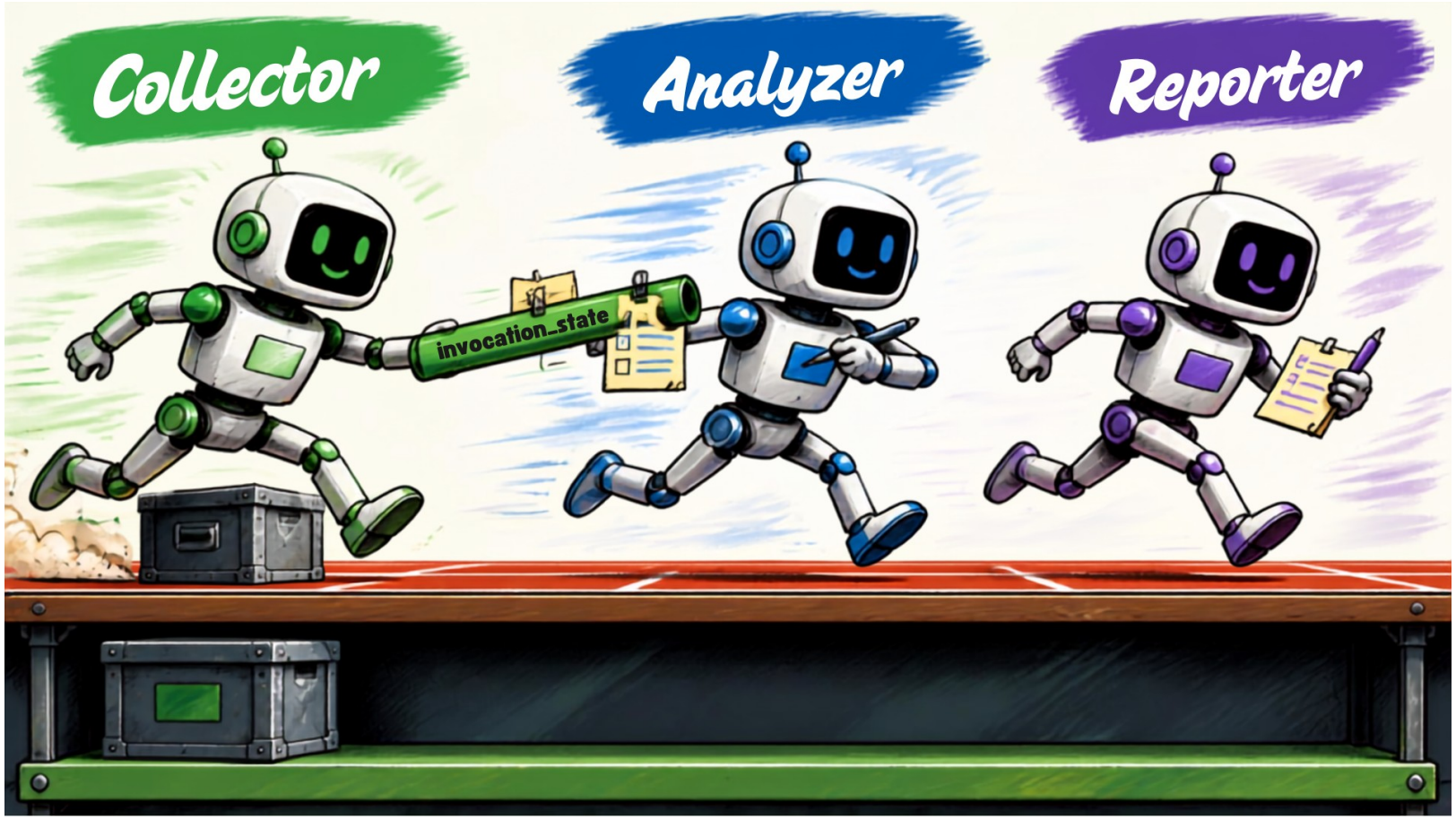
Externalize

```python
# Strands 1.56 — los backends de storage viven en strands.storage
from strands.storage import LocalFileStorage, S3Storage
from strands.vended_plugins.context_offloader import ContextOffloader

# Carpeta local para dev …
storage = LocalFileStorage(base_dir="./artifacts")
# … o durable + compartido en producción — misma interfaz
storage = S3Storage("my-bucket", prefix="tool-results/")

agent = Agent(model=MODEL, tools=[...],
    plugins=[ContextOffloader(storage=storage)])
```

STRANDS AGENTS SDK

Externalize Select Isolate

Ahora escálalo: un patrón, muchos agentes



Mismo patrón — el dato vive en invocation_state compartido, no en agent.state.


Externalize


Select


Isolate

```python
@tool(context=True)
def fetch_logs_swarm(app_name, tool_context):
    logs = generate_logs(hours)              # 145KB+
    pointer = f"logs-{app_name}"
    # Externalize a estado COMPARTIDO (multi-agente)
    tool_context.invocation_state[pointer] = logs
    return f"Stored as '{pointer}'. Hand off to analyzer."
collector = Agent(name="collector", tools=[fetch_logs_swarm])
analyzer  = Agent(name="analyzer",  tools=[analyze, latency])
reporter  = Agent(name="reporter",  tools=[generate_report])
swarm = Swarm([collector, analyzer, reporter])
```

STRANDS AGENTS SDK

Resultado — Swarm multi-agente

3

agentes coordinan
con un puntero

145 KB+

logs procesados,
cero en cualquier ventana

COMPLETED

collector → analyzer
→ reporter

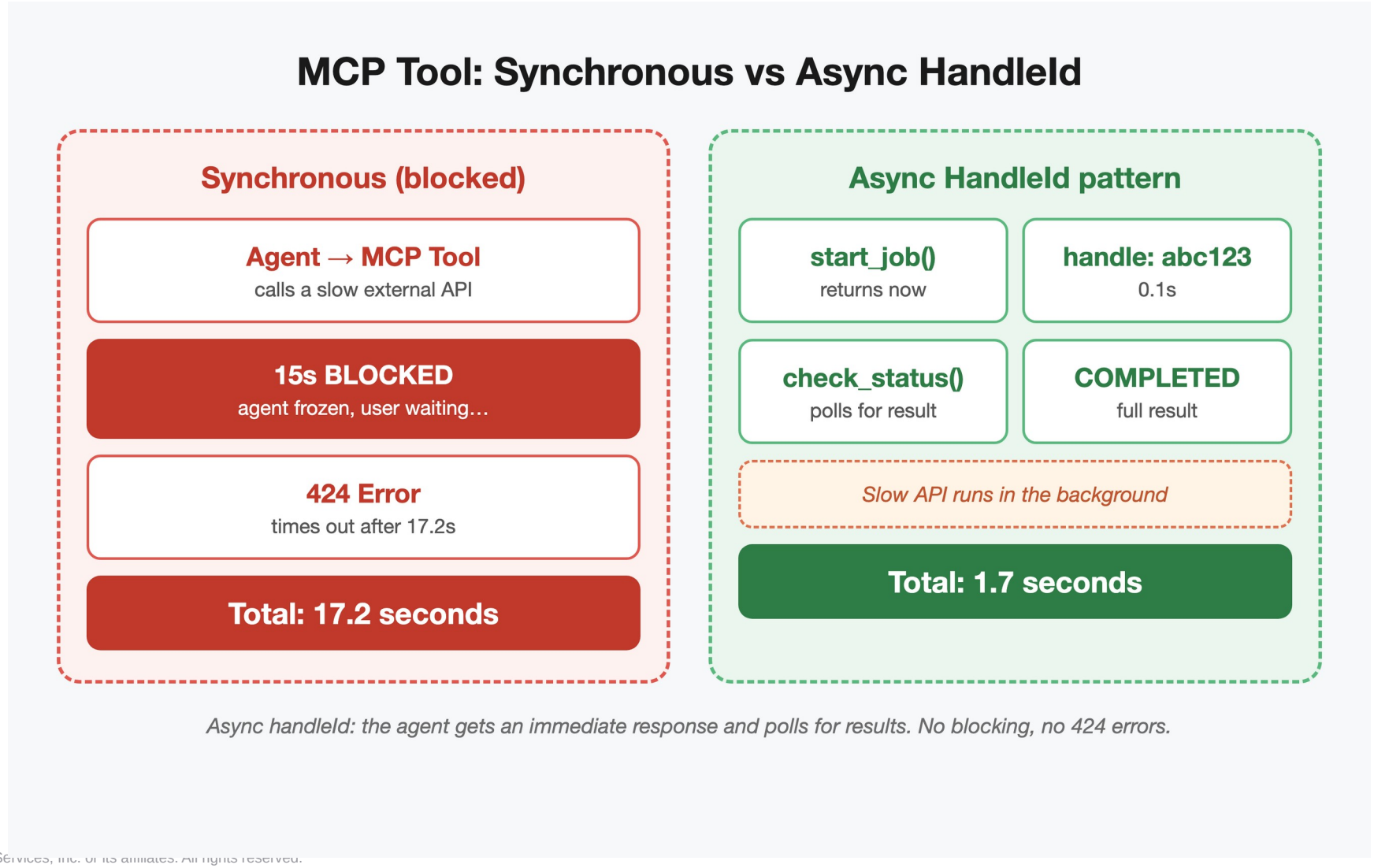
Un agente → agent.state · Multi-agente → invocation_state

Dos fallas más que golpean a los agentes

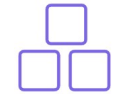
Mismo lente — elige la estrategia que encaja con la falla.

Isolate

Cuando una tool nunca regresa



Async handleId: the agent gets an immediate response and polls for results. No blocking, no 424 errors.



Isolate

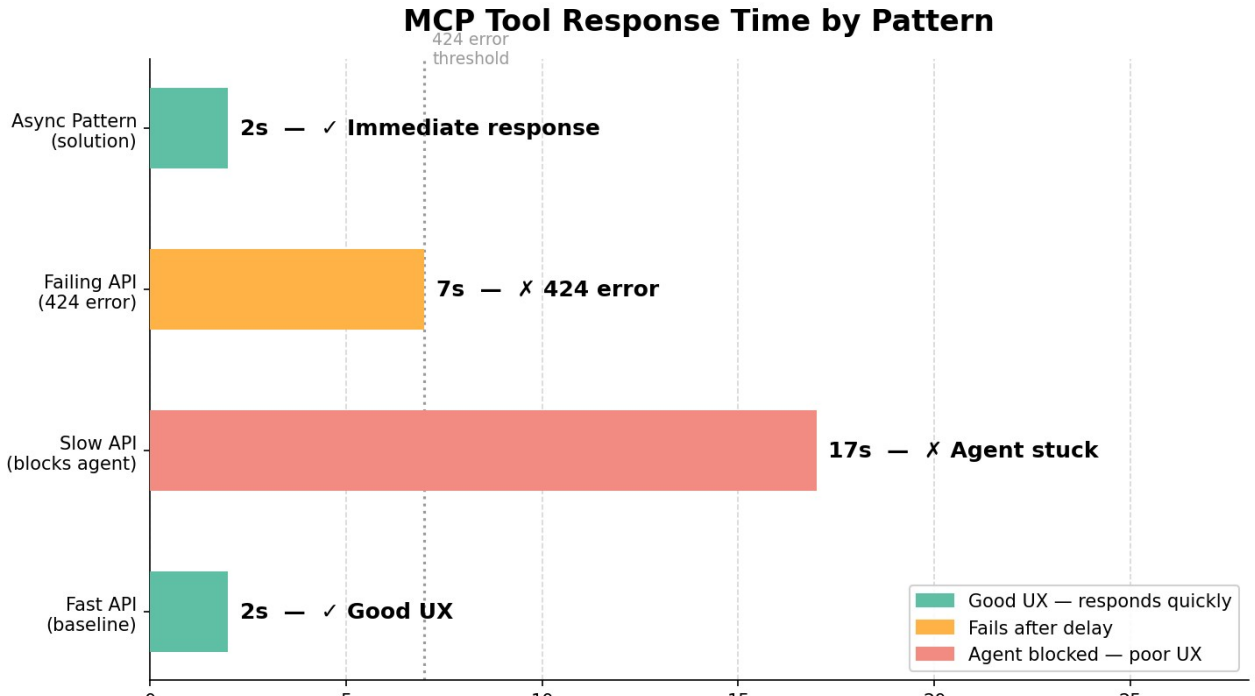
```python
@mcp.tool()
async def start_long_job(task: str) -> str:
    job_id = str(uuid.uuid4())[:8]
    # Isolate: corre fuera del loop, devuelve un handle YA
    asyncio.create_task(process_job(job_id, task))
    return f"JOB_STARTED: {job_id} - usa check_job_status"
@mcp.tool()
async def check_job_status(job_id: str) -> str:
    job = JOBS.get(job_id)
    return job["status"]  # processing | completed: <result>
```

Resultado — MCP Timeouts

Isolate

17.2s → 1.7s

tiempo de respuesta · sin más 424



Cuando el agente no para

Problema

- La tool dice “puede haber más resultados”
- El agente reintenta la misma llamada
- Llamadas redundantes, tokens quemados, sin avance

Estados claros + límite duro

- Las tools devuelven SUCCESS / FAILED
- El agente para con una señal clara
- LimitToolCounts limita llamadas por tool
- Techo duro sin importar el LLM

Compress

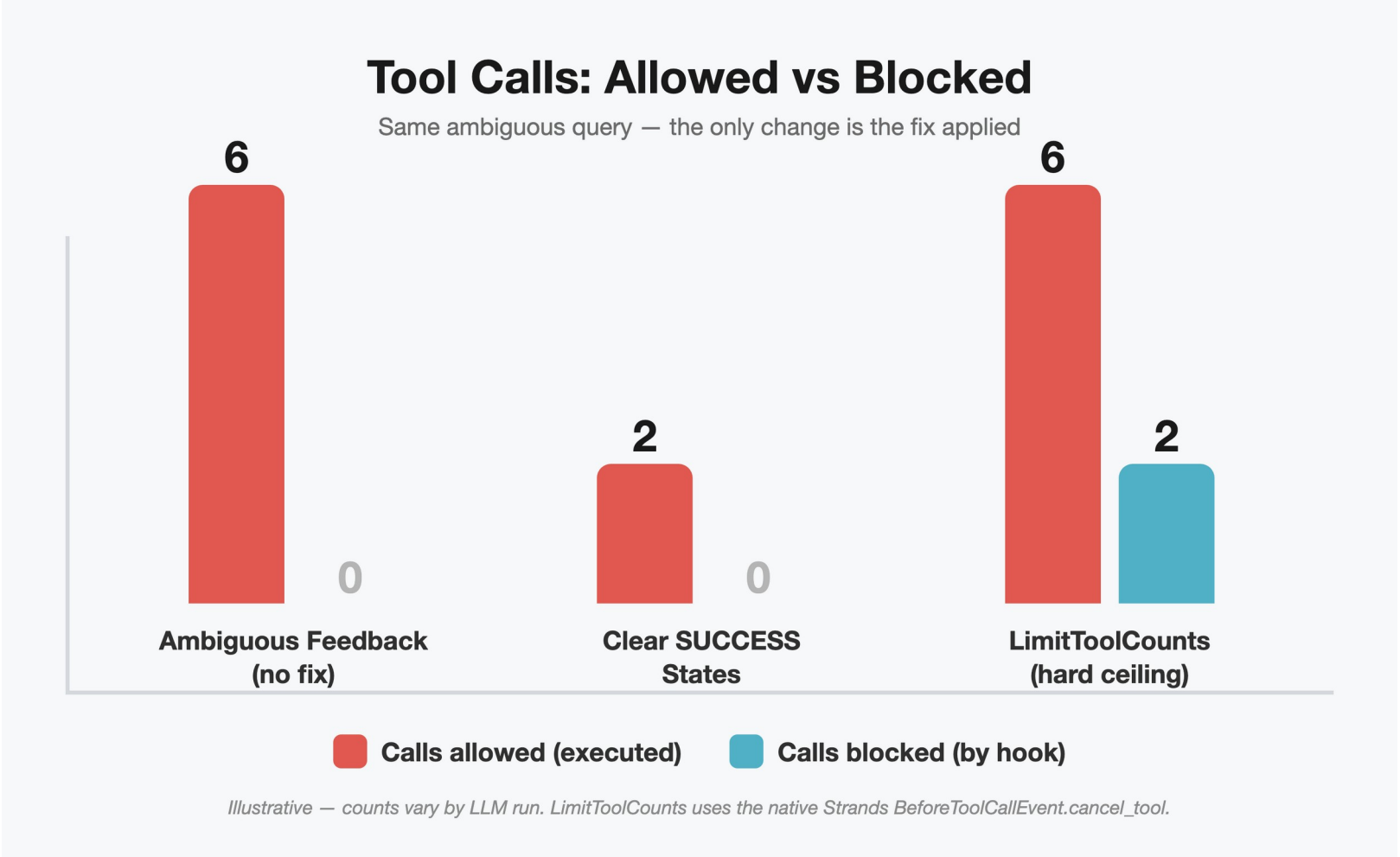
```python
@tool
def book_hotel(hotel, guest, nights):
    # SUCCESS / FAILED claro termina el loop
    return f"SUCCESS: Booking {conf} confirmed"
# LimitToolCounts — receta del Strands Hooks Cookbook
limit = LimitToolCounts(max_tool_counts={
    "search_flights": 3, "check_hotel_price": 3})
agent = Agent(tools=[...], hooks=[limit])
```

STRANDS AGENTS SDK

Resultado — Reasoning Loops

Compress

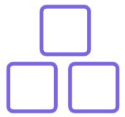
Estados claros paran el loop · LimitToolCounts limita el resto



Empareja la estrategia con la falla

La tool devuelve datos grandes (logs, docs, datasets)

Externalize + Select — memory pointer



Muchos agentes necesitan el mismo dato grande

invocation_state compartido + contexto acotado

El historial crece turno a turno

context_manager="auto" (Strands 1.56)

¡Gracias!

Elizabeth Fuentes Leone

Developer Advocate, GenAI — AWS

Todos los demos en Strands Agents (1.56)

Recursos de la charla

events.elifuentes.tech/events/context-overflow-memory-pointer

Blog y redes

elifuentes.tech