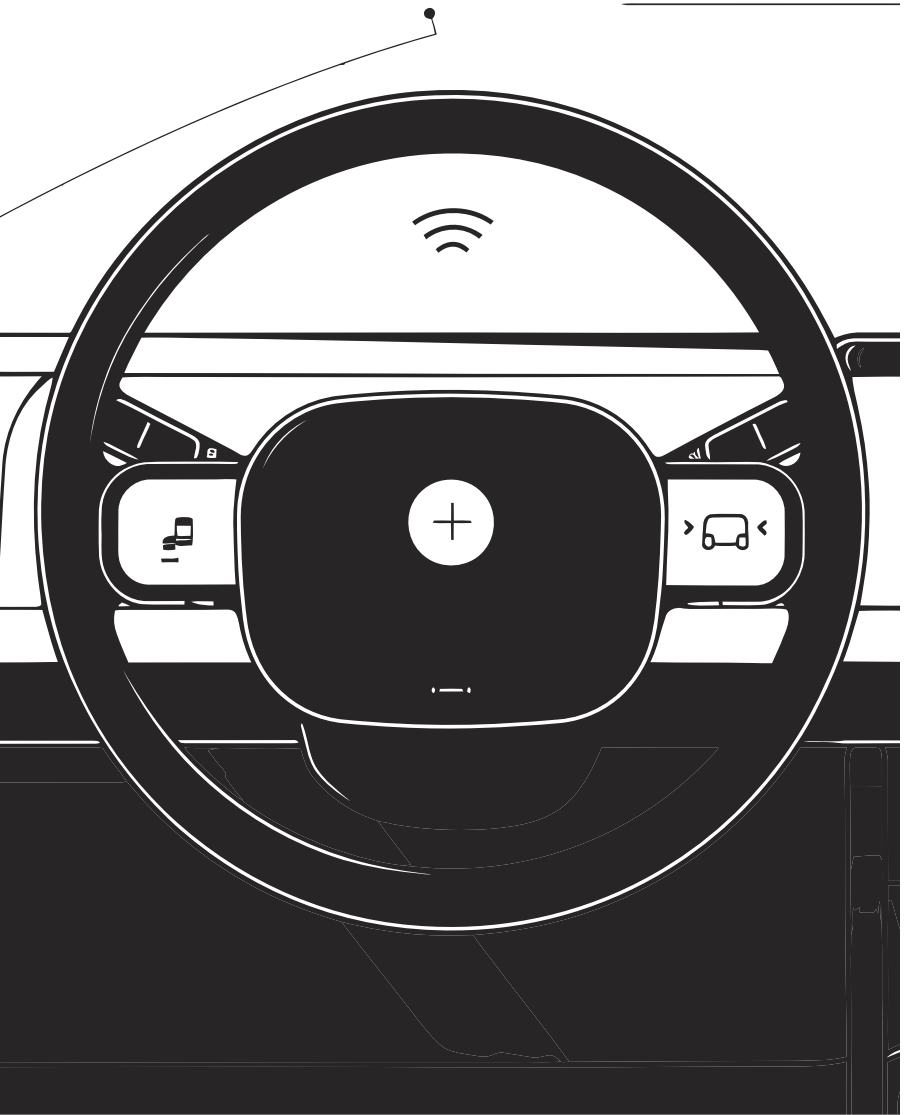
System Management and Monitoring Patterns

Explore patterns enabling effective management, monitoring, and control of integrated enterprise applications.

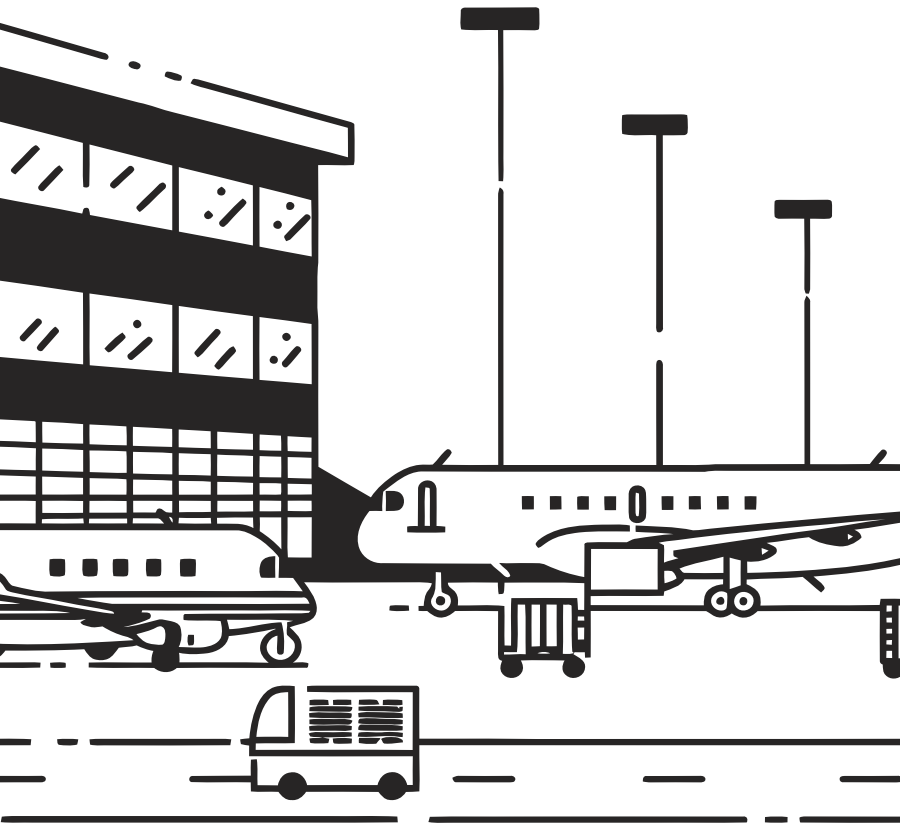
- Control Bus
- Detour
- Wire Tap
- Message Store

Achieve visibility, flexibility, and maintainability in complex integrations.



Why Monitoring and Management Matters

- Early detection of issues
- Easier troubleshooting
- Increased reliability and uptime
- Flexibility to quickly adapt in production environments



1. Control Bus

Definition:

- Allows centralized management of distributed systems.
- Sends commands to manage components remotely (start, stop, reconfigure).

Use Case:

- Remote configuration/control of integration endpoints.

How the Control Bus Works

Mechanics:

- Dedicated command messages sent via a special control channel.
- Endpoints listen for commands and execute actions.

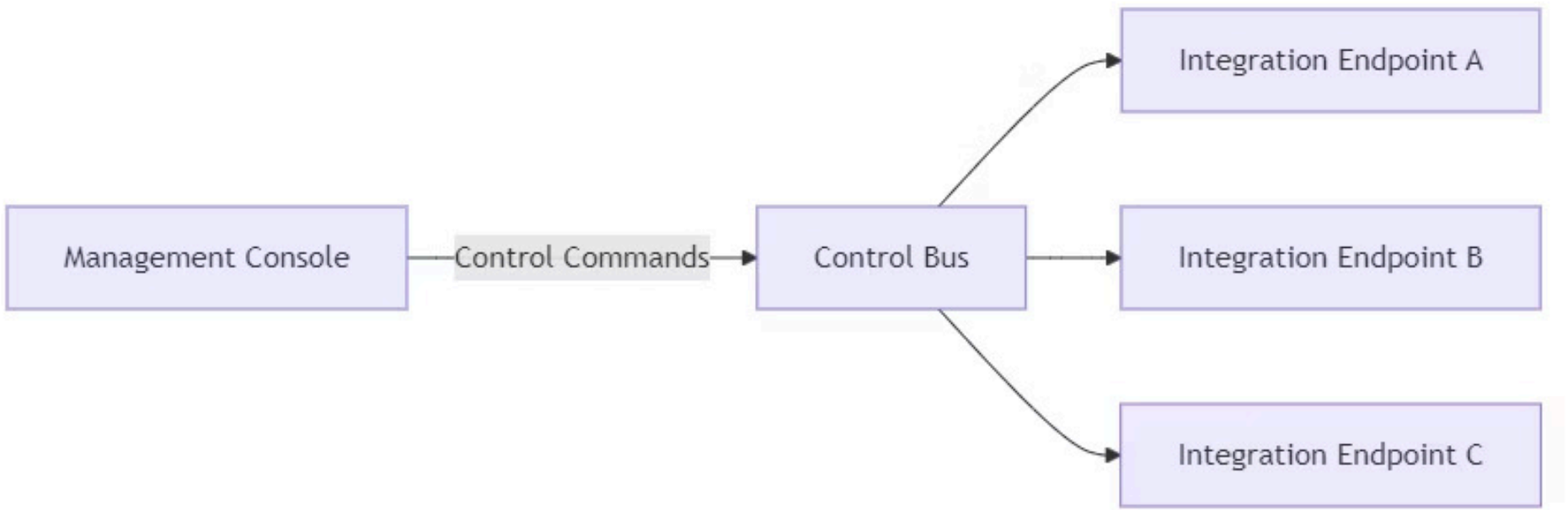
Examples of Commands:

- Pause/resume routes
- Adjust logging levels
- Restart components

Pros:

- Centralized management
- Enhanced operational agility

Control Bus – Diagram



2. Detour Pattern

Definition

Temporarily redirects message flows via an alternative path.

Purpose

Useful for maintenance, debugging, or handling exceptional conditions.

Scenario Example

Redirecting critical messages during maintenance windows.

How Detour Works

Dynamic Configuration

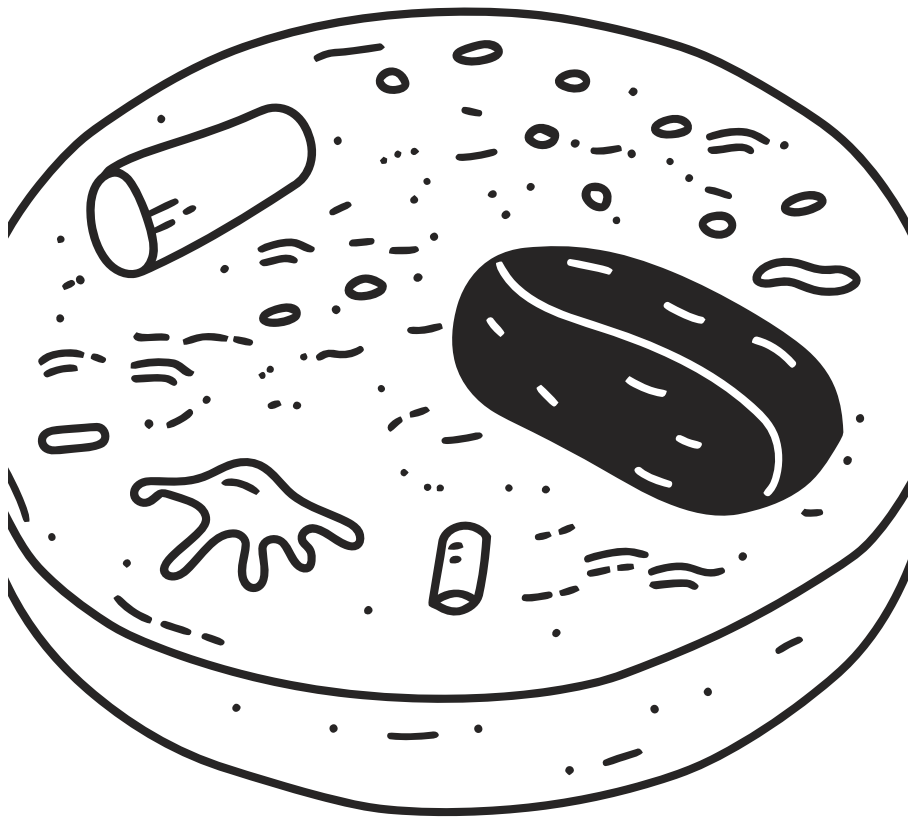
Dynamically alters routes via configuration or control commands.

Alternative Processing

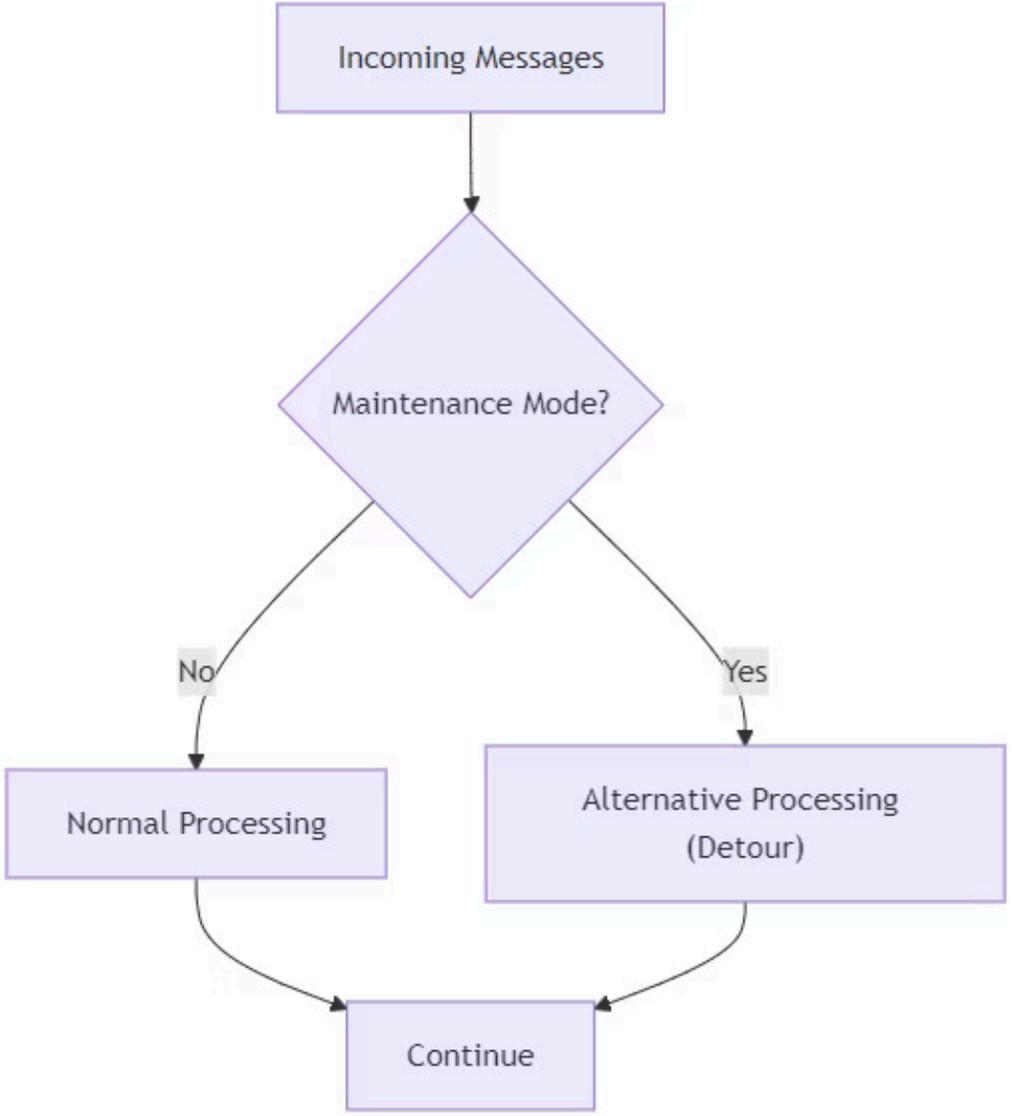
Temporarily sends messages through alternate processing routes.

Benefits

Minimizes downtime and facilitates debugging and testing.



Detour Pattern – Diagram



3. Wire Tap Pattern

Non-intrusive monitoring

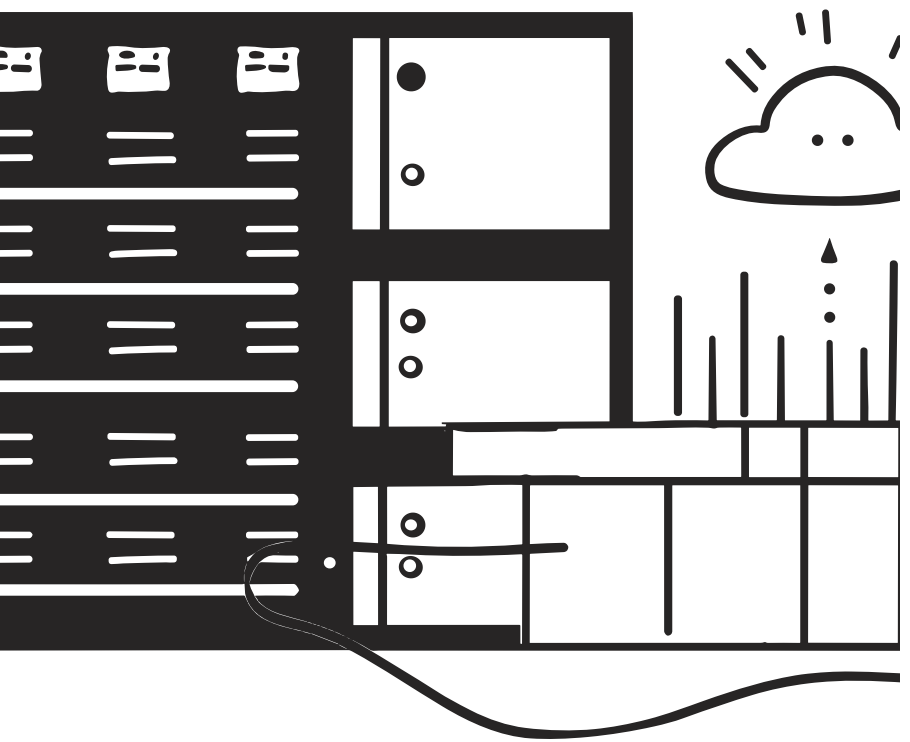
Observes without disruption

Message duplication

Copies messages asynchronously

Real-time analysis

Enables logging, auditing, and monitoring



How Wire Tap Works

Original Flow Continues

Original messages continue unaffected through the primary channel.

Message Duplication

Duplicate messages are created without impacting the original flow.

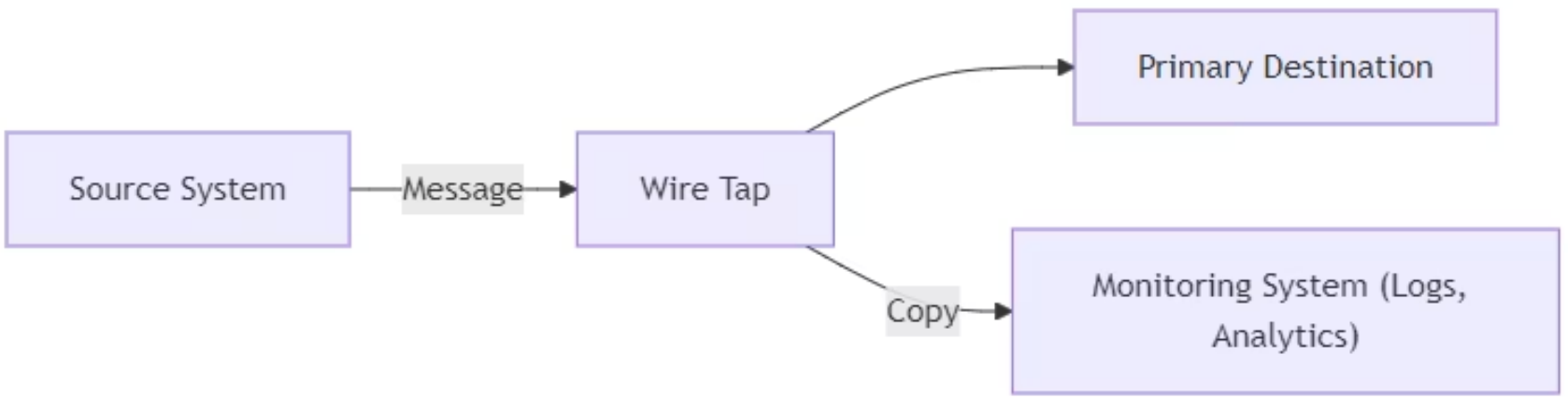
Secondary Channel

Duplicate messages sent to secondary monitoring channels for analysis.

Benefits Realized

Zero impact on primary flow while enabling real-time monitoring and troubleshooting.

Wire Tap – Diagram



4. Message Store Pattern

Definition:

- Persistently stores messages for later retrieval or auditing.
- Crucial for error recovery and historical tracking.

Use Case:

- Auditing transactions or replaying failed messages.

How Message Stores Work

Persistent Storage

Persistently stores messages for later retrieval or auditing

Historical Tracking

Maintains record of all message transactions

Error Recovery

Crucial for recovering from system failures

Auditing

Enables auditing transactions or replaying failed messages

Message Store – Diagram

Patterns Comparison

	Purpose	Complexity	Primary Benefit
Control Bus	Centralized Management	Medium	Operational agility
Detour	Flexible Routing	Low	Minimizes downtime
Wire Tap	Non-intrusive Logging	Low	Real-time monitoring
Message Store	Auditing & Recovery	Medium	Reliable message tracking

Real-World Applications

Financial Systems

Auditing transactions using Message Store.

Telecom Networks

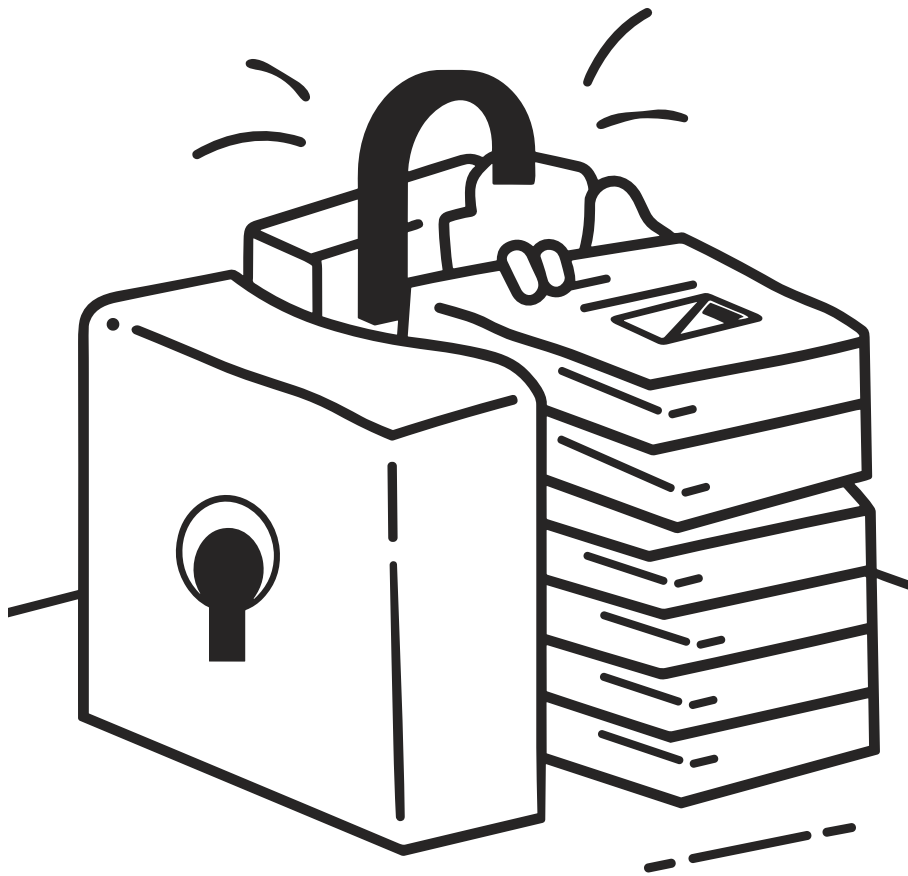
Detouring traffic during maintenance.

Retail Systems

Wire Tap for customer analytics.

Healthcare Systems

Control Bus for configuration management.



Applicability per Integration Style

	Message Broker	RPC	File Transfer	Shared Database (Monolith)
Control Bus	✅ Native support through dedicated control topics	⚠️ Possible but limited (via management APIs/endpoints)	❌ Difficult (no direct command channel)	⚠️ Possible but indirect (using database flags/configuration tables)
Detour	✅ Easy via topic rerouting	⚠️ Possible but complex (endpoint rerouting)	❌ Very limited due to batch nature	❌ Difficult (requires application-level logic changes)
Wire Tap	✅ Native (easy duplication to monitoring topics)	⚠️ Requires interceptor logic (middleware)	✅ Possible (copying files asynchronously)	⚠️ Indirect via database triggers or logging
Message Store	✅ Easy, integrated in broker (Kafka, RabbitMQ persistence)	⚠️ Possible via logging or API call logs	✅ Natural fit (files already stored persistently)	✅ Natural fit (database already stores state/messages)

Applicability per Integration Style

Message Brokers

Excel in implementing all patterns due to inherent asynchronous architecture and built-in routing capabilities.

RPC-based integrations

Generally struggle with asynchronous monitoring patterns (Wire Tap, Detour). Monitoring requires middleware or additional services.

File Transfer integrations

Naturally support Message Store (files stored persistently), but lack the flexibility for Control Bus or Detour due to batch and asynchronous nature.

Monolithic Shared Database systems

Leverage existing storage naturally for Message Store, but managing dynamic routing (Detour) or command-based management (Control Bus) requires significant custom application logic.