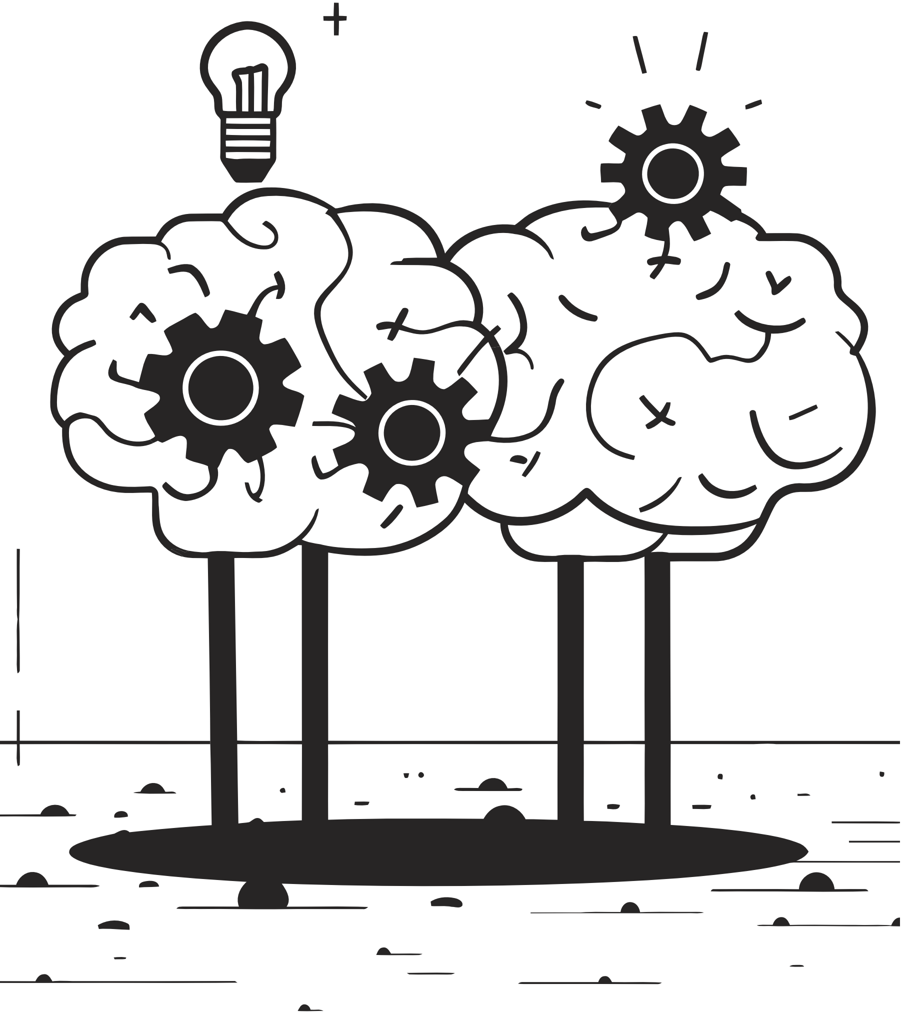


# What are Anti-patterns?

Anti-patterns are common but ineffective solutions that initially appear beneficial but lead to long-term problems, complexity, and failures.

**Why they matter:**

- Understanding anti-patterns helps architects and developers avoid expensive mistakes.
- Anti-patterns represent widely repeated mistakes; recognizing them early saves significant effort.

# Common Microservice Anti-patterns

## 1. Swarm of Gnats Event Anti-pattern

### Description:

- Excessively fine-grained services producing many small, insignificant events.
- Difficult to maintain context or meaningful interactions.

### Impact:

- Poor performance, complexity in debugging, increased infrastructure costs.
- Increased cognitive load due to handling large volumes of trivial events.

### Solution:

- Aggregate events into meaningful business events.
- Clearly define boundaries and scope for event publishing.

# Common Microservice Anti-patterns

## 2. Infinity Architecture Anti-pattern

### Description:

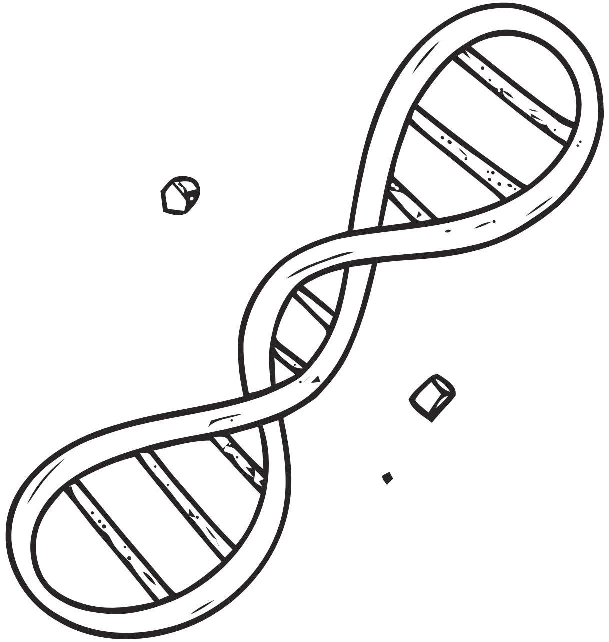
Constant redesign of architecture without clear goals. Endless cycles of refactoring and technology experimentation.

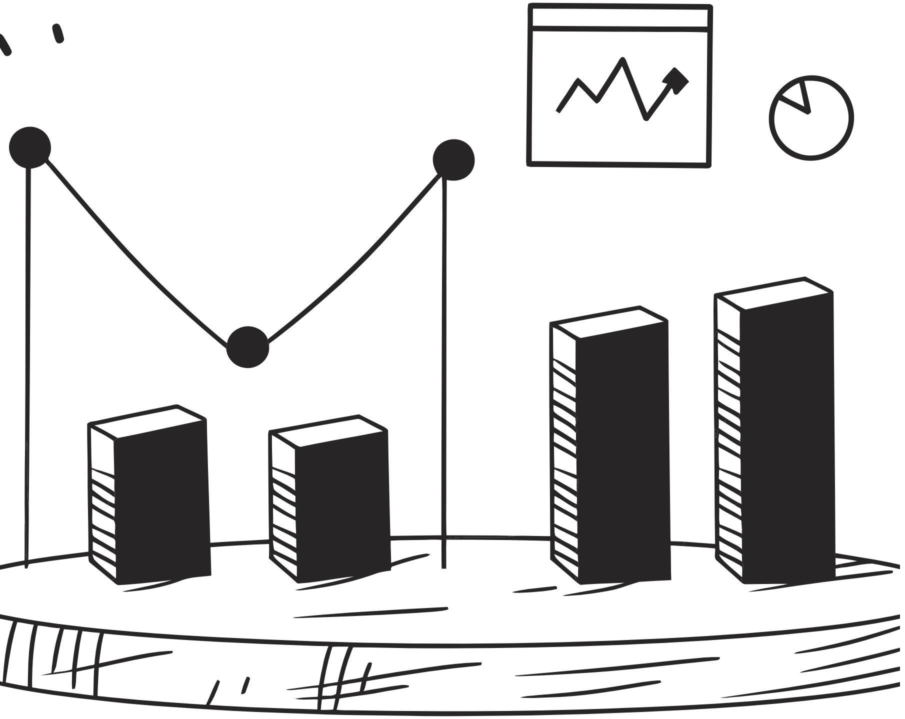
### Impact:

Delays in delivery, lack of stable architecture. Team frustration, productivity losses.

### Solution:

Define clear architectural goals and constraints upfront. Prioritize stable incremental evolution over complete redesigns.





# Common Microservice Anti-patterns

## 3. Out-of-context Scorecard Anti-pattern

### Description:

Measuring service health or performance using irrelevant or generic metrics. Metrics do not reflect actual business or service value.

### Impact:

False sense of security, poor decision-making based on misleading data. Missed critical service problems due to irrelevant metrics.

### Solution:

Align metrics directly to business outcomes and service-specific goals. Regularly revisit and refine metrics.

# Common Microservice Anti-patterns

## 4. Stovepipe Architecture Anti-pattern

### Description

Isolated, vertically aligned services with no integration strategy. Duplication of functionality, data silos, limited reuse.

### Impact

Inefficient data sharing, duplication of effort, increased maintenance costs. Reduced flexibility and poor adaptability.

### Solution

Promote integration and reuse across services. Employ common data management and API standards.

# Common Microservice Anti-patterns

## 5. Architecture by Implication Anti-pattern

### Impact

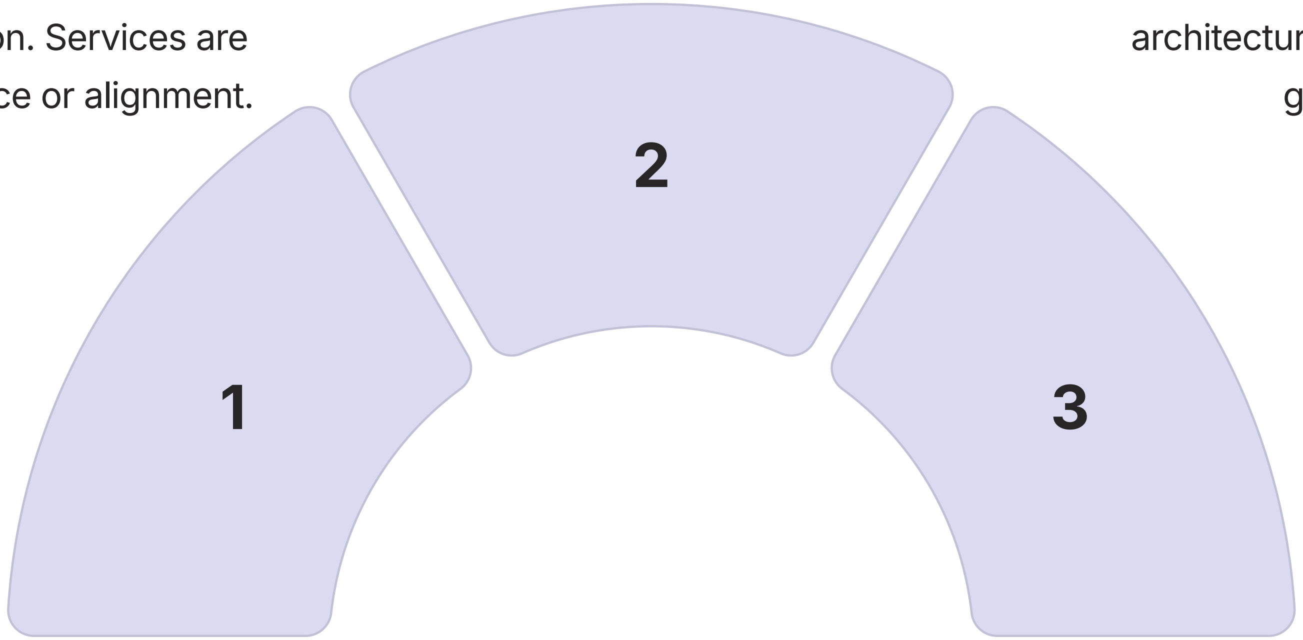
Inconsistent design, unpredictable results, technical debt accumulation. Difficulty scaling and evolving the system.

### Description

Lack of explicit architectural strategy; architecture emerges accidentally from decisions made in isolation. Services are built without clear guidance or alignment.

### Solution

Establish explicit, clearly documented architecture guidelines. Regular architecture reviews and active governance.



# Common Microservice Anti-patterns

## 6. Frozen Caveman Anti-pattern

1

### Description

Services that never evolve after initial deployment, stuck in outdated technology stacks. Fear of breaking existing functionality discourages improvements.

2

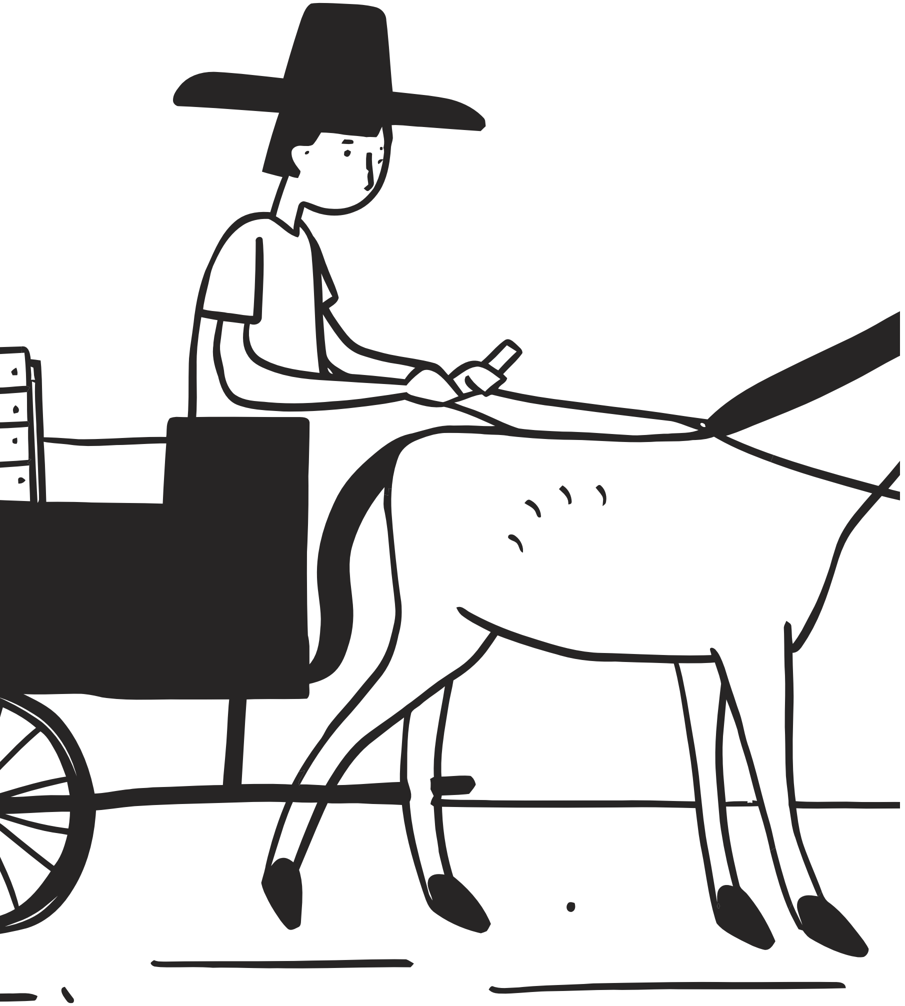
### Impact

Increased security risks, difficulty hiring skilled personnel. Technical debt and diminished competitive advantage.

3

### Solution

Promote regular technology updates as part of operational discipline. Implement continuous integration/deployment to ease regular updates.



# Common Microservice Anti-patterns

## 7. Cart Before the Horse Anti-pattern

1

### Description

Choosing technologies before understanding requirements or context. Decisions based solely on trends rather than suitability.

2

### Impact

Poor fit for actual use cases, leading to complexity and rework. Excessive operational overhead.

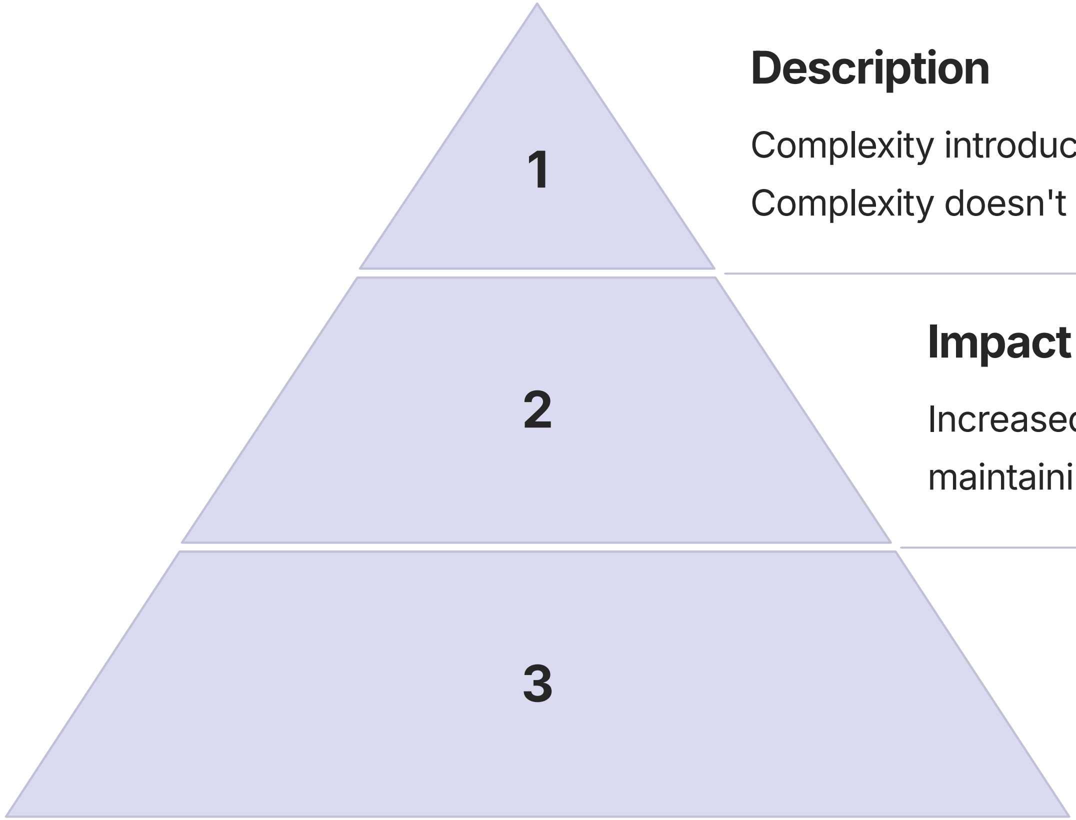
3

### Solution

Clearly define requirements and architecture goals first. Evaluate technology choices based on specific needs and context.

# Common Microservice Anti-patterns

## 8. Accidental Complexity Anti-pattern



### Description

Complexity introduced due to overly sophisticated solutions, tools, or practices. Complexity doesn't add actual value to the system.

### Impact

Increased operational burden, slower development cycles. Difficulty maintaining and troubleshooting.

### Solution

Keep solutions as simple as possible. Regularly review and challenge complexity to justify its necessity.

# Common Microservice Anti-patterns

## 9. Elephant Migration Anti-pattern

### Description

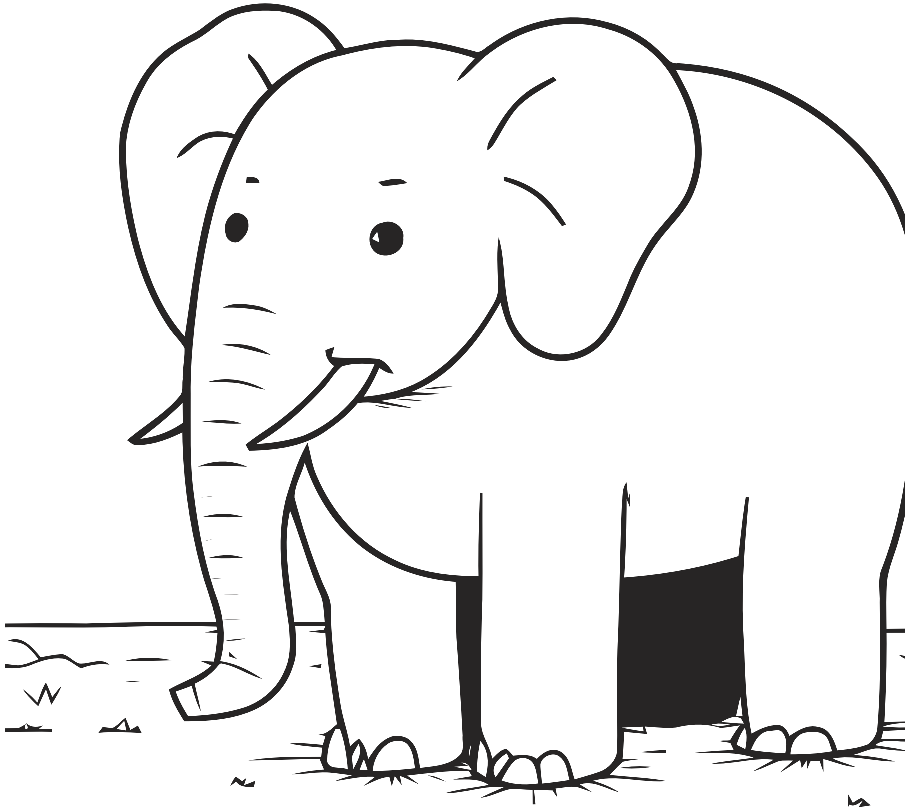
Attempting a single large-scale migration to microservices ("big bang" approach). Huge, risky transitions rather than incremental evolution.

### Impact

Significant business disruptions, risk of major failures. Increased risk of project cancellation or severe delays.

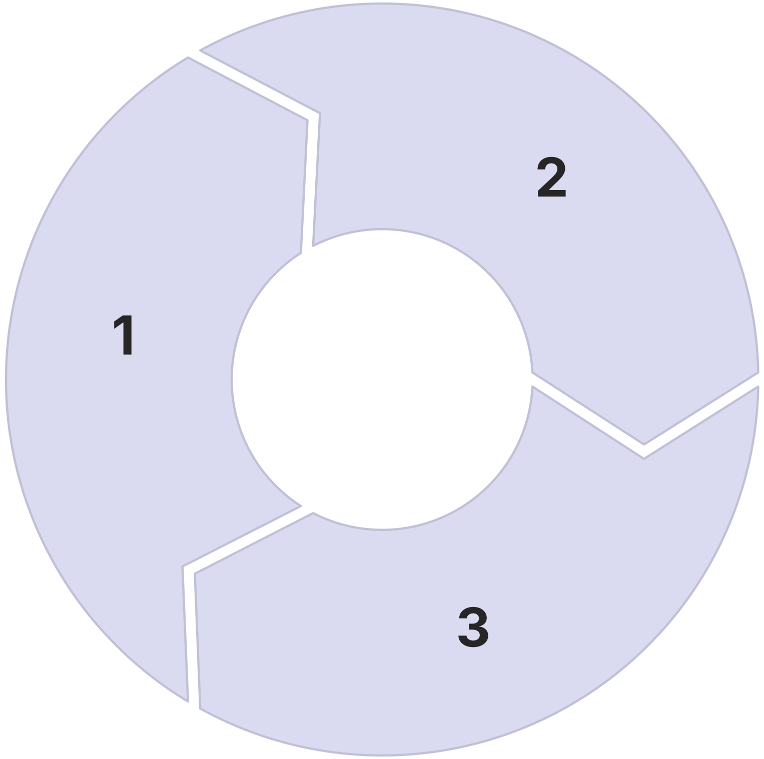
### Solution

Incrementally migrate using strategies like the Strangler Pattern. Plan small, controlled, reversible migration steps.



# Fallacies in Microservice Architectures

## 1. Fallacy of Compensating Updates



### Misconception:

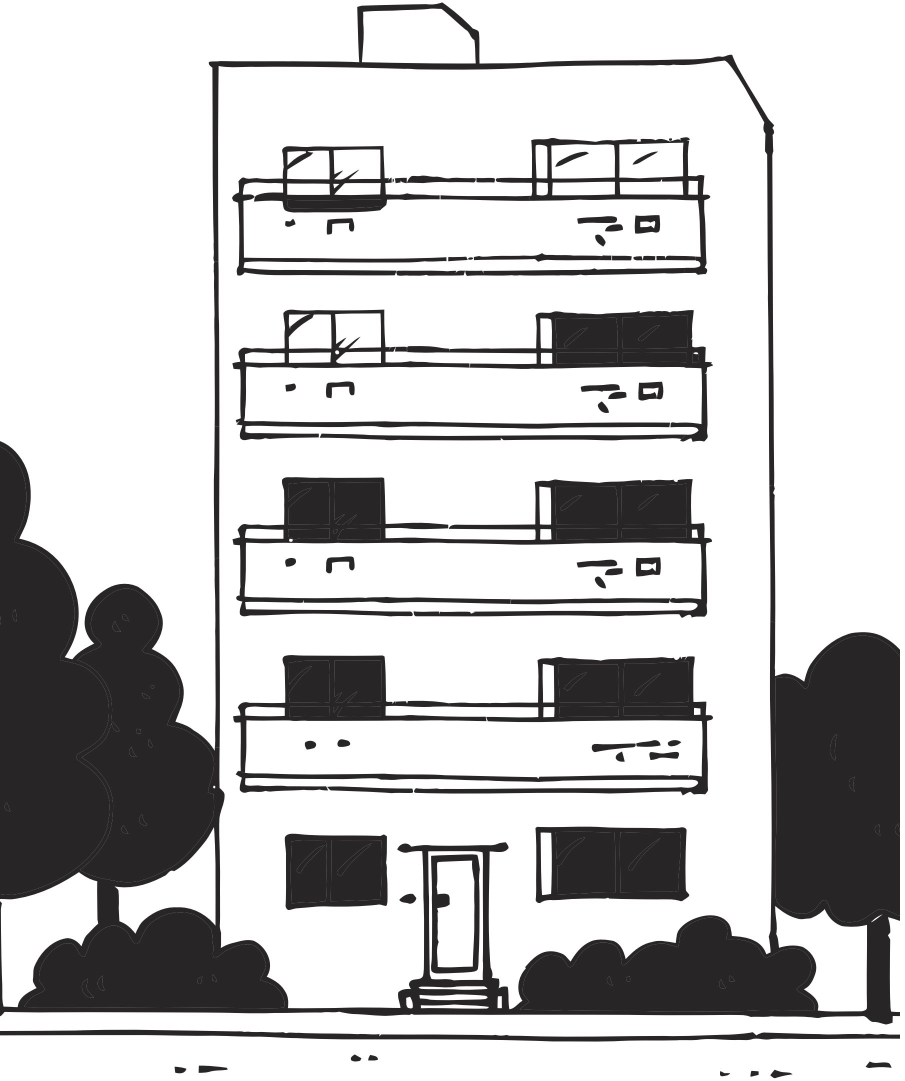
Belief that eventual consistency is simple to implement through compensating transactions.

### Reality:

Compensating actions are complex, error-prone, and hard to manage at scale.

### Mitigation:

Carefully design and thoroughly test compensation strategies. Prefer simpler eventual consistency models and minimize complex compensations.



# Fallacies in Microservice Architectures

## 2. Fallacy of API Versioning

### Misconception:

Belief that frequently creating new API versions solves backward compatibility issues.

### Reality:

Multiple versions quickly become costly and hard to manage, creating confusion and technical debt.

### Mitigation:

Plan APIs for evolution with backward-compatible changes whenever possible. Use clear lifecycle policies for API versions and limit concurrent versions in production.

# Fallacies of Distributed Computing

### Common mistaken assumptions:

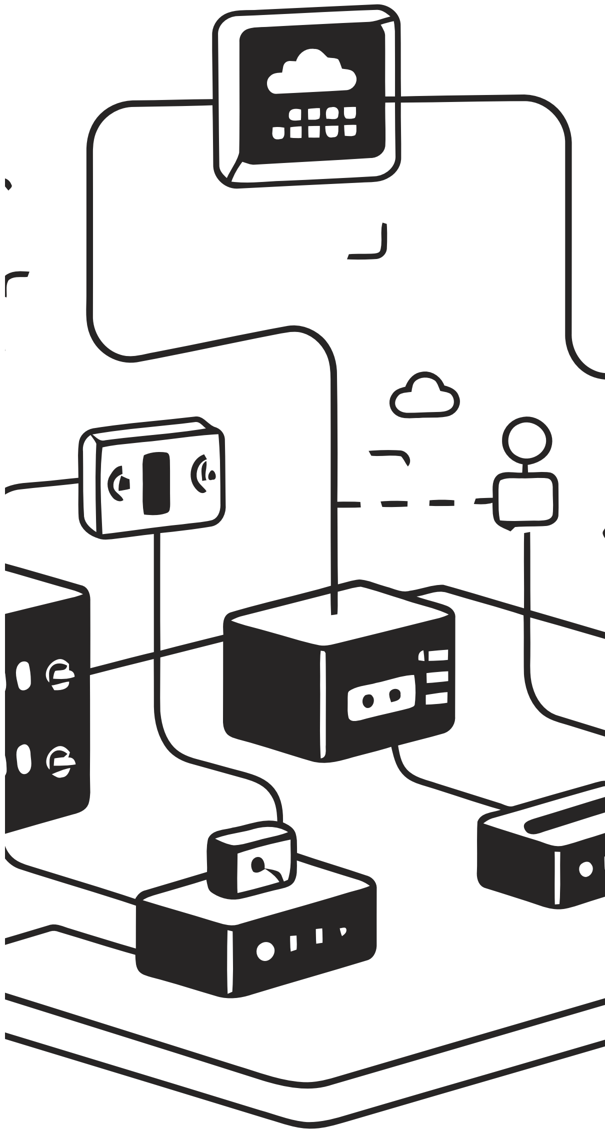
- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- The network is secure.
- Topology doesn't change.
- There is only one administrator.
- Transport cost is zero.
- The network is homogeneous.

### Impact of these fallacies:

- System instability, poor performance, unpredictable failures.

### Mitigation:

- Design for failure (timeouts, retries, fallbacks).
- Account explicitly for network latency and unreliability.
- Implement rigorous security measures at every service boundary.



# Avoiding Anti-patterns and Fallacies: Practical Guidelines

- **Architecture Governance:** Maintain clear, regularly reviewed architecture guidelines.
- **Incremental Change:** Avoid large, disruptive changes—prefer incremental and controlled evolutions.
- **Realistic Metrics:** Select relevant, context-specific service metrics linked to business outcomes.
- **Continuous Learning:** Educate teams about common pitfalls and best practices in distributed systems.
- **Simplicity First:** Challenge complexity regularly—simplify whenever possible.