# Topic 5.1: What Goes Wrong

## Failures, Hacks, and Systemic Risk in Digital Finance

Joerg Osterrieder

Digital Finance

2025

## Learning Objectives

### What You Will Learn in This Topic

By the end of this session, you will be able to:

1. **Categorize** types of digital finance failures (technical, economic, human)
2. **Explain** how the major types of attacks work, using plain-English descriptions and analogies
3. **Analyze** case studies of historical DeFi hacks
4. **Assess** how the interconnection of protocols creates cascading risk
5. **Develop** genuine risk awareness for evaluating digital finance systems

### Hands-On Component

Colab notebook (NB11) simulating historical DeFi exploits—run reentrancy, flash loan, and oracle manipulation scenarios to understand how attacks work.

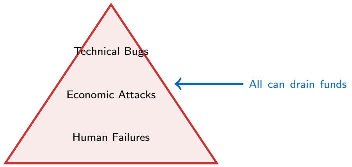
## Prerequisites: What You Should Know

**Required Background:**

- Basic understanding of smart contracts
- Familiarity with DeFi concepts (lending, DEXs)
- Understanding of blockchain transactions

**Helpful but not required:**

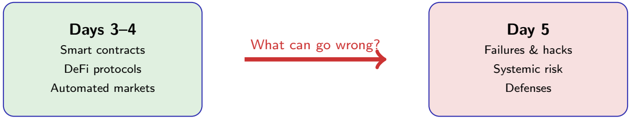
- No coding knowledge needed—all technical concepts explained visually
- Knowledge of token standards (ERC-20)
- Experience with DeFi protocols



**Key Mindset**

“Code is law” until something goes wrong. Then we discover the limits of trustless systems.

## From Building to Breaking: Day 4 → Day 5

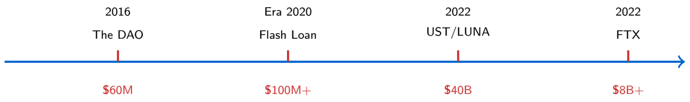


In Days 3–4 we learned how DeFi creates value through smart contracts and automated markets. Today we ask: **what happens when things go wrong?**

**The Stakes Are Real**

Over **$100 billion** has been lost to hacks, exploits, fraud, and collapses in digital finance. Understanding these failures is the key to understanding what makes—or breaks—the entire ecosystem.

## The Big Picture: Why Failures Matter



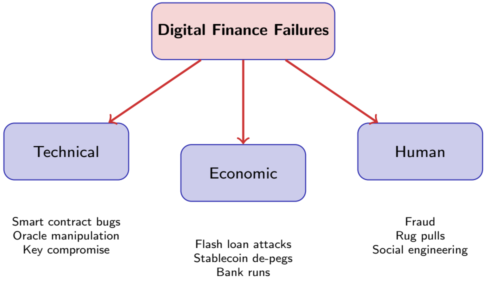
**Cumulative losses in DeFi/crypto exceed $100 billion.**

- Understanding failures is essential for building secure systems
- Each exploit category teaches different lessons
- Risk assessment requires knowing what can go wrong

### Course Context

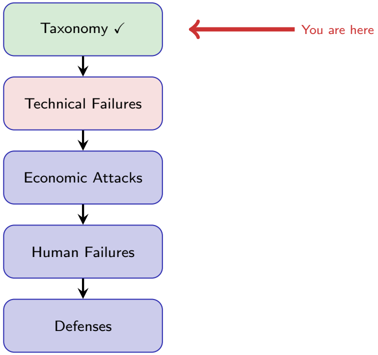
Day 5 examines the "dark side"—failures, regulation, governance, and privacy. This topic focuses on what goes wrong technically and economically.

## A Taxonomy of Failures



**Key Insight:** All three categories can result in total loss of funds, but they require different defenses.

## Roadmap: Where We Are Going



We will explore each category with **real-world case studies** and **everyday analogies** to make the concepts concrete. No coding knowledge is required.

## Category 1: Technical Failures

**Smart Contract Bugs:**

- Reentrancy attacks
- Integer overflow/underflow (like a car odometer rolling from 999,999 back to 000,000)
- Mistakes in the rules—like a vending machine that gives change before checking if you paid
- Access control flaws (when the wrong people can use restricted features)

**Infrastructure Failures:**

- Oracle manipulation (the blockchain's "window to the outside world"—we'll explain oracles in detail shortly)
- Bridge vulnerabilities (bridges connect different blockchains—we'll explore their risks later)
- Consensus bugs (errors in the agreement process between computers)
- Key management failures

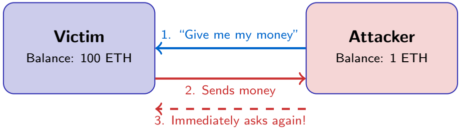
### The DAO Hack (2016)

**Loss:** $60M (3.6M ETH)
**Cause:** Reentrancy bug
**Result:** Ethereum hard fork
**Lesson:** Code is NOT always law

### Wormhole Bridge (2022)

**Loss:** $320M
**Cause:** Signature verification bug
Attacker minted unbacked tokens

## Deep Dive: Reentrancy Attack



**Analogy:** Imagine a bank teller who hands you cash and THEN checks your balance. A clever attacker keeps asking for cash before the teller finishes checking—getting paid multiple times from one withdrawal.

**The Problem:**

1. Victim sends money *before* updating the balance record
2. Attacker immediately requests another withdrawal
3. Balance not yet updated, so the check passes again
4. Repeat until the entire contract is drained

### Prevention

Check-Effects-Interactions pattern: Update records BEFORE sending money.

## Vulnerable vs. Safe: How the Fix Works

**No Code Required**

You don't need to understand programming—focus on the **order of steps** below.

**Dangerous Order:**

1. Check: "Does this person have money?"
2. **Send the money out** ← external call
3. Update records: "They now have $0"

**Why it fails:** The contract sends money BEFORE updating its records—this is the vulnerability. The attacker can keep requesting money before step 3 ever runs.

**Safe Order:**

1. Check: "Does this person have money?"
2. **Update records first**: "They now have $0"
3. Send the money out

**Why it works:** Even if the attacker tries to re-enter, the records already show $0, so the check in step 1 fails.

**Check-Effects-Interactions (CEI) Pattern**

1. **Check:** Validate conditions ("Do they have enough?")
2. **Effects:** Update your own records
3. **Interactions:** Only THEN send money or talk to other contracts

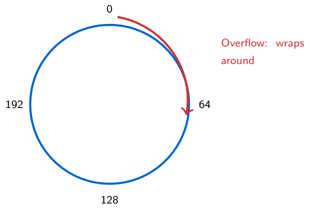
## Integer Overflow and Underflow

**The Problem:**
**Analogy:** Imagine a counter that can only store numbers 0 to 255. What happens when you add 1 to 255? It wraps around to 0—that's **overflow**. What happens when you subtract 1 from 0? It wraps to 255—that's **underflow**.

Older smart contracts used counters like this but with much bigger numbers. Attackers exploited the wrapping to give themselves enormous fake balances.

**Example:**

- Attacker's balance = 0 tokens
- Attacker subtracts 1 token
- Balance wraps to a gigantic number
- Attacker now has "infinite" tokens!



**Solution**

Modern smart contracts include automatic checks that stop the transaction if a number would wrap around.

## Access Control Vulnerabilities

**Common Mistakes:**

- No check on *who* is calling a function
- Features meant to be private are left open to anyone
- Administrative powers not properly restricted
- Setup steps left unprotected

**Vulnerable (no lock on the door):**

- Anyone can call "change the owner"
- Attacker makes *themselves* the owner
- Attacker now controls the entire contract

**Safe (locked door):**

- Before running "change the owner," the contract checks: "Is the person asking *actually* the current owner?"
- If not, the request is rejected
- Only the real owner can transfer control

**Analogy:** Think of a house where the front door has no lock. Anyone can walk in and claim they own it. Adding access control is like installing a lock—only people with the right key can get in.

**Best Practice**

Every sensitive function should check "who is asking?" before executing.

## Pause & Reflect

**Let’s take stock of what we’ve covered so far.**

We’ve seen three types of **technical** vulnerabilities:

1. **Reentrancy** — the “ask-for-money-before-records-update” trick
2. **Integer overflow/underflow** — the “odometer wrapping around” problem
3. **Access control flaws** — the “unlocked front door” problem

### Discussion Question

Which of these three do you think is the most dangerous, and why? Consider: which one is hardest to detect, and which one has caused the biggest losses?

**Coming up next:** Economic attacks—where the code works exactly as written, but attackers find clever ways to exploit the *design* itself.

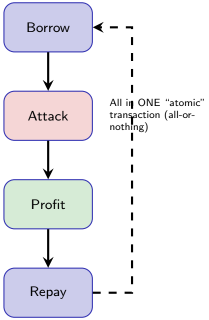
## Category 2: Economic Attacks

**Flash Loan Attacks:**
**Analogy:** Think of a flash loan like a time-travel movie: you can borrow millions, use them, and return them—all in a single instant. If anything goes wrong, it's as if it never happened.

- Borrow millions, attack, repay—all in one transaction
- No collateral needed
- Exploit price discrepancies
- Manipulate governance votes

**How Flash Loans Work:**
"Atomic" means all-or-nothing: either *every* step succeeds, or the entire transaction is cancelled as if it never happened.

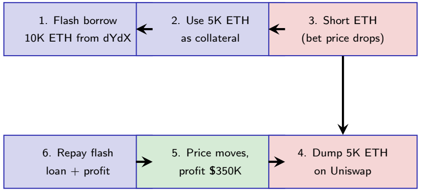
1. Borrow \$100M (no collateral)
2. Execute attack strategy
3. Repay \$100M + small fee
4. If any step fails, everything is reversed



## Flash Loan Attack Example: bZx (2020)

### Key Term: Shorting

**Shorting** means betting that a price will go **down**. If Alice thinks Bitcoin will drop from $50,000 to $40,000, she can "short" it and profit if she's right. If the price goes up instead, she loses money.


1. Flash borrow 10K ETH from dYdX
2. Use 5K ETH as collateral
3. Short ETH (bet price drops)
4. Dump 5K ETH on Uniswap
5. Price moves, profit $350K
6. Repay flash loan + profit

**Key insight:** Capital requirement to manipulate markets dropped from millions to **zero**.

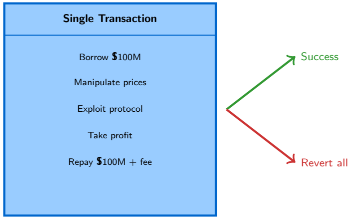
# The Atomic Property of Flash Loans

**What "atomic" means:**

- All operations must complete in ONE transaction
- Either everything succeeds, or nothing happens
- Loan + use + repay = single block
- No collateral at risk (transaction reverses if unpaid)

**Why this enables attacks:**

- Zero capital requirement
- No personal funds at risk
- Can borrow unlimited amounts
- Only pay small fee if attack succeeds



**Democratization of Capital**

Flash loans "democratize" access to large capital—for both legitimate arbitrage AND attacks.

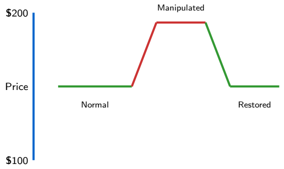
## Oracle Manipulation

**The Problem:**
**Analogy:** An oracle is like a thermometer for the blockchain—it reports real-world information (like prices) that smart contracts cannot see on their own. **Oracle manipulation** is like tampering with that thermometer so it shows a wrong temperature, causing the system to make bad decisions.

- DeFi protocols need external price data
- Oracles provide this data on-chain
- If oracle can be manipulated, protocol is vulnerable

**Attack Pattern:**

1. Flash borrow large amount
2. Trade on a DEX (decentralized exchange—recall from Day 4) to move the current "spot" price
3. Protocol reads the manipulated price
4. Exploit (borrow more, liquidate others)
5. Restore price, repay loan



**Defense**

Use TWAP (Time-Weighted Average Price) or decentralized oracles like Chainlink.

## Spot Price vs. TWAP Oracles

**Spot Price Oracle:**

- Current instant price
- Single data source (one DEX)
- Easy to manipulate with one trade
- Cheap and simple

**Vulnerability:**

- Flash loan can move price
- Manipulation in single block
- No historical context

**TWAP Oracle:**

- Time-Weighted Average Price
- Average over time window (e.g., 30 min)
- Resistant to short-term manipulation
- More expensive to attack

**Why it's safer:**

- Attacker must sustain manipulation
- Across many blocks
- Capital locked (not flash loan)

**Best Practice**

Use decentralized oracles (Chainlink) with multiple data sources and TWAP for price-sensitive operations.

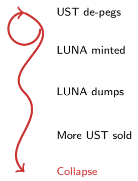
## Category 3: Stablecoin Failures

**UST/LUNA Collapse (May 2022):**
*Recall from Day 4:* Algorithmic stablecoins try to maintain their $1 price through automatic mint/burn mechanisms rather than holding dollar reserves. UST was the largest such experiment.

- Market cap: $18B at peak
- De-pegged from $1 to $0.10
- LUNA: $80 to $0.0001
- Total value destroyed: $40B+

**The Death Spiral:**
**Analogy:** This is similar to a traditional **bank run**—when everyone tries to withdraw at once, the system cannot handle it and collapses.

1. Large UST sell-off
2. Peg breaks, panic ensues
3. LUNA minted to defend peg
4. LUNA hyperinflates
5. Both collapse to zero



**Lesson**

Algorithmic stability requires robust mechanisms—“code” alone is insufficient.

## Category 4: Centralized Exchange Collapses

**FTX Collapse (Nov 2022):**

- 2nd largest crypto exchange
- $32B valuation (more than many national airlines)
- Customer funds misappropriated
- $8B+ missing
- CEO convicted of fraud

**Mt. Gox (2014):**

- 70% of Bitcoin trading
- 850,000 BTC "lost"
- Combination of hack + fraud
- 10+ years to partial recovery

**Common Patterns:**

1. Opaque operations
2. Commingled funds
3. Lack of proof of reserves (a public demonstration that the exchange actually holds all the assets it claims to hold)
4. Regulatory arbitrage
5. Charismatic founders

**Not Your Keys, Not Your Coins**

Centralized custodians reintroduce the trust problems DeFi was designed to solve.

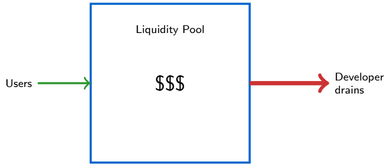
## Category 5: Rug Pulls and Fraud

**Types of Rug Pulls:**

- **Liquidity pull:** Developers drain LP tokens (recall: LP tokens represent your share of a liquidity pool)
- **Limiting sell orders:** Hidden code prevents selling
- **Dumping:** Team sells massive holdings
- **Exit scam:** Project disappears with funds

**Red Flags:**

- Anonymous team
- Unlocked liquidity
- No audit
- Unrealistic promises
- FOMO marketing
- Celebrity endorsements



**2021 Stats:**
$2.8B lost to rug pulls (that's roughly the entire GDP of a small country)
(Chainalysis)

## Bridge Exploits — The Biggest Target

**Analogy:** A blockchain bridge is like a **ferry between islands**. Each island (blockchain) has its own currency. The ferry (bridge) locks your currency on one island and gives you equivalent currency on the other. If pirates (hackers) take over the ferry, they can steal everything on board.

**Why bridges are vulnerable:**

- Hold massive TVL (Total Value Locked—the total amount of assets deposited into the bridge)
- Complex multi-chain logic
- Multiple attack surfaces
- Validator key management
- Novel, less-tested code

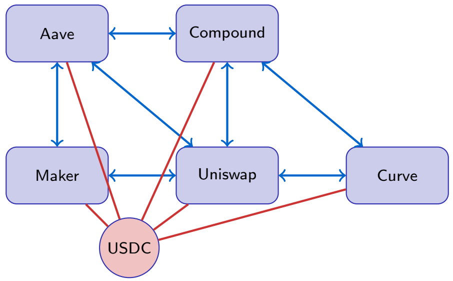
**Major Bridge Hacks:**

| **Bridge** | **Loss** |
|---|---|
| Ronin | \$625M |
| Poly Network | \$611M |
| Wormhole | \$320M |
| Nomad | \$190M |
| Harmony | \$100M |

**2022 Pattern**

Bridges accounted for 69% of crypto stolen in 2022.

## Systemic Risk in Digital Finance



**DeFi Composability (recall: building blocks that connect together) = Systemic Risk:**

- Protocols depend on each other ("money legos")
- Stablecoins are systemic (USDC freeze = cascade)
- Oracle failure affects ALL dependent protocols
- Smart contract bug can propagate through ecosystem

## Largest DeFi Exploits (2020–2024)

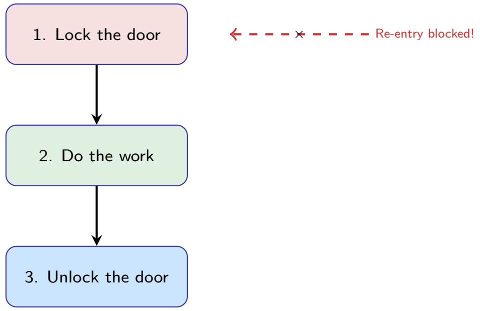
| **Protocol** | **Loss** | **Type** | **Year** |
|---|---:|---|---|
| Ronin Bridge | $625M | Bridge exploit | 2022 |
| Poly Network | $611M | Bridge exploit | 2021 |
| FTX | $477M | Hack post-bankruptcy | 2022 |
| Wormhole | $320M | Bridge exploit | 2022 |
| Nomad Bridge | $190M | Bridge exploit | 2022 |
| Beanstalk | $182M | Flash loan governance | 2022 |
| Wintermute | $160M | Key compromise | 2022 |

### Pattern Recognition

**Bridges are the weakest link.** Cross-chain bridges hold massive amounts of locked assets but have complex attack surfaces.

## Defense Patterns: Reentrancy Guard

**The Door Lock Pattern:**
**Analogy:** When you enter a bathroom, you lock the door. If someone else tries to enter while you're inside, the lock stops them. Once you're done, you unlock it. A reentrancy guard works exactly the same way.



**How it works:**

1. Set the lock when someone enters
2. If a re-entrant call tries to enter...
3. The lock is still set, so the call is rejected
4. Lock released only when the original operation finishes

**Best Practice**

Production smart contracts use well-tested, pre-built reentrancy guard libraries (e.g., from OpenZeppelin) rather than writing their own.

## Defense Patterns: Pull vs. Push Payments

**Push Pattern (Risky):**
**Analogy:** A waiter brings food to every table. If one table is blocked (a difficult customer), the waiter is stuck and nobody else gets served.

**How it works:**

- The contract sends money to each user one by one
- If one recipient fails (bad address, malicious contract), the *entire* payment process stops

**Problems:**

- One bad recipient blocks all others
- Reentrancy risk
- Can run out of processing power ("gas")

**Pull Pattern (Safe):**
**Analogy:** A buffet—everyone goes up and serves themselves when they're ready. If one person has trouble, it doesn't affect anyone else.

**How it works:**

- The contract records how much each user is owed
- Users come and withdraw their own funds whenever they want
- Each withdrawal is independent

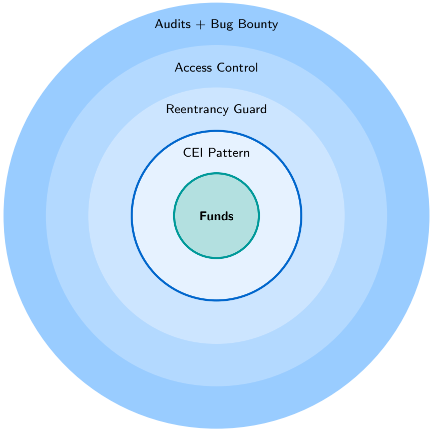
**Benefits:**

- Isolates failures
- User controls timing
- Easier to secure

**Rule of Thumb**

Let users "pull" funds rather than "pushing" to them—like a buffet instead of table service.

## Defense in Depth



**Defense in Depth Principle:**

- Multiple independent security layers
- If one layer fails, others still protect
- No single point of failure

## Hands-On: Analyzing an Exploit (NB11)

### Simulate Real Exploit Scenarios

**In the Colab notebook, we will:**

1. Run simulations of exploit scenarios (reentrancy, flash loans, oracle manipulation)
2. Model how each attack type drains funds step-by-step
3. Identify the attack patterns and vulnerabilities
4. Calculate attacker profit in simulated scenarios
5. Discuss what could have prevented each exploit

#### Access the Notebook

`day_05/notebooks/NB11_DeFi_Exploits.ipynb`

We'll simulate real exploit types to understand how they work. All code is pre-written—you interact with it, not write it.

**Time: 20-25 minutes for guided exploration**

## Notebook Objectives

**Part 1: Reentrancy Simulation**

- Model a vulnerable contract
- Simulate the recursive withdrawal attack
- Observe how balance checks fail

**Part 2: Flash Loan Attack**

- Simulate borrowing without collateral
- Model price manipulation
- Track fund movements
- Calculate profit margins

**Part 3: Oracle Manipulation**

- Simulate spot price vs TWAP oracles
- Model how attackers move prices
- Compare defense mechanisms

**Key Takeaway**

Simulations help you understand exploit mechanics without risking real funds—learn the patterns that lead to vulnerabilities.

## Discussion: Risk Awareness

**Questions to Consider:**

1. Should smart contracts be audited before deployment?
2. Who is liable when code fails?
3. Is “code is law” a feature or a bug?
4. How do we balance innovation with safety?

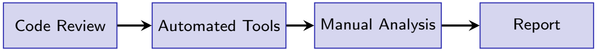
**Key Takeaways:**

- Failures are inevitable—design for them
- Technical, economic, and human risks compound
- Transparency enables post-mortem but not prevention
- Systemic risk grows with interconnection

**Risk Framework**

For any protocol: What can go wrong technically? Economically? Who has the keys? What happens when it fails?

## The Security Audit Process



**Audit Severity Levels:**

- **Critical:** Direct loss of funds—MUST fix before deployment
- **High:** Significant risk—should fix
- **Medium:** Moderate risk—recommended to fix
- **Low:** Minor issues—consider fixing
- **Informational:** Suggestions for improvement

### Important Caveat

Audits are NOT guarantees. Recall from our case studies: the Ronin Bridge ($625M loss) and Wormhole ($320M) were both audited. Audits reduce risk but don't eliminate it.

## Bug Bounty Programs

**How Bug Bounties Work:**

- Protocols offer rewards for finding vulnerabilities
- Ethical hackers report bugs instead of exploiting
- Rewards scale with severity
- Continuous security testing

**Major Platforms (Global):**

- Immunefi (DeFi-focused, global)
- HackerOne (global, general-purpose)
- Code4rena (audit competitions)
- Slowmist (Asia-Pacific focused)
- Hacken (Eastern Europe)

*Context:* Had the bZx ($350K exploit) or Beanstalk ($182M) protocols offered competitive bounties, ethical hackers might have reported the vulnerabilities first.

**Typical Rewards:**

| Severity | Reward |
|---|---|
| Critical | $50K–$10M |
| High | $10K–$50K |
| Medium | $1K–$10K |
| Low | $100–$1K |

**Incentive Alignment**

Good bounties make ethical disclosure more profitable than exploitation.

## Responsible Disclosure

**Ethical Discovery Process:**

1. Find vulnerability
2. Document the issue thoroughly
3. Contact protocol team privately
4. Give reasonable time to fix
5. Coordinate public disclosure

**DO NOT:**

- Exploit for personal gain
- Disclose publicly before fix
- Demand excessive payment
- Threaten the protocol

**White Hat vs. Black Hat:**

**White Hat (ethical):**

- Reports vulnerabilities
- Helps fix issues
- Collects bounty legally

**Black Hat (malicious):**

- Exploits for profit
- Causes financial harm
- Faces legal consequences

**Gray Area**

Some hackers exploit then return funds (like the Poly Network $611M case)—legally and ethically ambiguous.

## Executive Summary

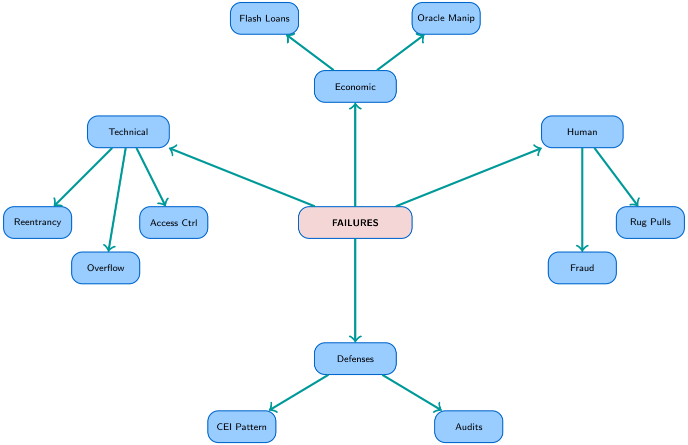
### Key Takeaways from Topic 5.1

1. **Failures fall into three categories:** technical bugs, economic attacks, and human fraud
2. **Technical exploits** (reentrancy, overflow, access control) can be prevented with secure design patterns
3. **Economic attacks** (flash loans, oracle manipulation) exploit protocol design assumptions
4. **Bridges are the biggest target**—69% of 2022 crypto theft involved bridges
5. **Defense in depth:** Multiple security layers (CEI, guards, audits, bounties) reduce but don't eliminate risk

**The Big Idea**

Understanding how things fail is essential for building secure systems. "Code is law" only works when the code is correct.

## Concept Map: Digital Finance Failures



## Key Terms & Definitions (I)

| | |
|---|---|
| Reentrancy Attack | An exploit where a malicious contract calls back into the victim contract before the first execution completes, allowing repeated withdrawals. |
| Flash Loan | An uncollateralized loan that must be borrowed and repaid within a single atomic blockchain transaction. |
| Oracle Manipulation | Artificially moving market prices to deceive protocols that rely on external price data for decision-making. |
| Check-Effects-Interactions (CEI) | A secure coding pattern: validate conditions (check), update state (effects), then make external calls (interactions). |
| TWAP | Time-Weighted Average Price—an oracle mechanism that averages prices over time to resist short-term manipulation. |

## Key Terms & Definitions (II)

| | |
|---|---|
| Rug Pull | A type of fraud where project developers drain funds from a liquidity pool or abandon a project after collecting investments. |
| Bridge Exploit | An attack targeting cross-chain bridges, which hold large amounts of locked assets and have complex security surfaces. |
| Defense in Depth | A security strategy using multiple independent layers of protection, so failure of one layer doesn't compromise the system. |
| Bug Bounty | A program offering rewards to ethical hackers who discover and responsibly disclose security vulnerabilities. |
| Systemic Risk | The risk that failure of one protocol cascades through interconnected systems, affecting the entire ecosystem. |

## Common Misconceptions

| Misconception | Reality |
|---|---|
| "Audited means safe" | Many exploited protocols had multiple audits; audits reduce but don't eliminate risk |
| "DeFi eliminates all trust" | Trust shifts from institutions to code, oracles, and governance—different trust, not zero trust |
| "Smart contracts are always correct" | Code can have bugs, logic errors, and unforeseen interactions just like any software |
| "Flash loans are inherently malicious" | Flash loans have many legitimate uses (arbitrage, collateral swaps); they're tools that can be misused |

**Critical Thinking**

Always ask: What are the assumptions? What happens if they're wrong? Who holds the keys?

## Self-Assessment: Question 1

### Question

A bank teller hands you cash and THEN checks your account balance. If you keep asking for cash before the teller finishes checking, what type of smart contract attack does this resemble?

A. A flash loan attack

B. A reentrancy attack

C. An oracle manipulation

D. A rug pull

## Self-Assessment: Question 1

### Question

A bank teller hands you cash and THEN checks your account balance. If you keep asking for cash before the teller finishes checking, what type of smart contract attack does this resemble?

A. A flash loan attack

B. A reentrancy attack

C. An oracle manipulation

D. A rug pull

**Answer: B**

**Explanation:** A reentrancy attack occurs when a contract sends money (like the teller handing out cash) before updating its records. The attacker exploits this gap by requesting money again before the records are updated, draining the contract repeatedly.

## Self-Assessment: Questions 2–3

### Question 2

Why are flash loans especially dangerous, even though the attacker has to return the money?

**Answer:** Because the attacker can temporarily control enormous amounts of capital (millions of dollars) to manipulate prices or exploit vulnerabilities, and if the attack fails, they lose nothing—the transaction simply reverses. The risk is entirely one-sided.

### Question 3

A restaurant where the waiter brings food to every table (push) vs. a buffet where guests serve themselves (pull)—which is safer for smart contracts and why?

**Answer:** The pull (buffet) pattern is safer. In the push pattern, one problematic recipient can block payments to everyone else. In the pull pattern, each user withdraws independently, so one failure doesn't affect others. This also reduces reentrancy risk.

## What's Next: Topic 5.2

### Preview: Regulatory Landscapes

**In Topic 5.2, we'll explore:**

- How **governments respond** to digital finance failures
- Comparison of **regulatory frameworks** (US, EU, Asia)
- The **securities law** question: When is a token a security?
- **AML/KYC requirements** and their implications
- Emerging **stablecoin regulations**

**Connection**

Topic 5.1 examined *what goes wrong* technically and economically.
Topic 5.2 examines *how regulators respond* to these failures.

**Preparation:** Consider how the failures we discussed today might influence regulatory approaches.

## Resources for Further Learning

**Essential Reading:**

- Rekt News (`rekt.news`)—detailed post-mortems of DeFi exploits
- "Smart Contract Security Best Practices" (ConsenSys Diligence)
- Chainalysis Crypto Crime Reports (annual)

**Online Resources:**

- Course notebook: `NB11_DeFi_Exploits.ipynb`
- DeFiLlama Hacks Dashboard (`defillama.com/hacks`)
- Immunefi Bug Bounty Platform (`immunefi.com`)

**Technical Deep Dives:**

- SWC Registry (Smart Contract Weakness Classification)
- OpenZeppelin Security Blog
- Samczsun's Blog (security researcher)

### Course Materials

All slides and notebooks available on the course website.

## Questions?

**Questions & Discussion**

What Goes Wrong?

**Contact:** joerg.osterrieder@ifi.uzh.ch

**Next Topic:** T5.2 — Regulatory Landscapes