Advanced Topics: Scaling, Flash Loans & Security

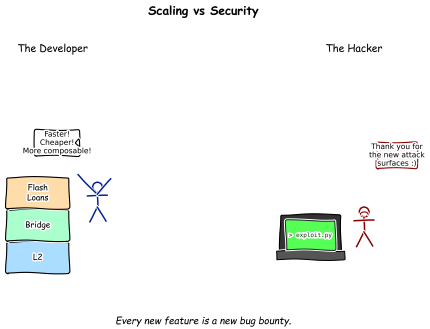
An Ultra-Simple Visual Guide with Deep-Dive Appendix

Prof. Dr. Jörg Osterrieder

BSc Blockchain, Crypto Economy & NFTs

Spring 2026

Faster, Cheaper, Hackable: Pick Two



Blockchain promised unstoppable money. Then we wanted it faster, so we built new layers. Then we wanted more features, so we added bridges between chains.

Every new feature created a new way for hackers to attack.

The core tension: How do you build systems that are both unstoppable and safe?

This lecture explains scaling, flash loans, and security — in plain English with pictures.

Since 2020, bridge hacks alone have caused over 2.5 billion dollars in losses — more than all US bank robberies combined.

Learning Objectives

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

1. **Explain** the scalability trilemma using a real-world analogy [Understand]
2. **Describe** how Layer 2 solutions make Ethereum faster and cheaper [Understand]
3. **Compare** flash loan uses (good) with flash loan attacks (bad) [Analyze]
4. **Evaluate** smart contract risks using a simple security checklist [Evaluate]

No math required. Main slides use only plain English and pictures.
Technical formulas and code are in the Appendix for those who want them.

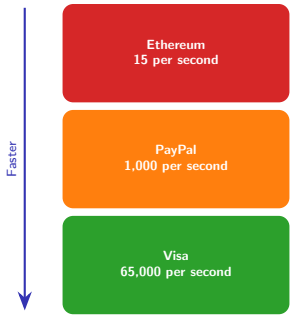
Bloom's levels covered: Understand, Analyze, Evaluate. The Appendix adds Apply.

The Speed Problem

Imagine a highway with only one lane. During rush hour, everyone is stuck. Tolls skyrocket because drivers bid against each other for the few available spots.

That is Ethereum today:

- About 15 cars (transactions) per second
- Visa handles 65,000 per second
- During busy times, fees jump above 100 dollars



When a popular NFT mint happens, Ethereum gas fees can hit 200 dollars — pricing out most everyday users.

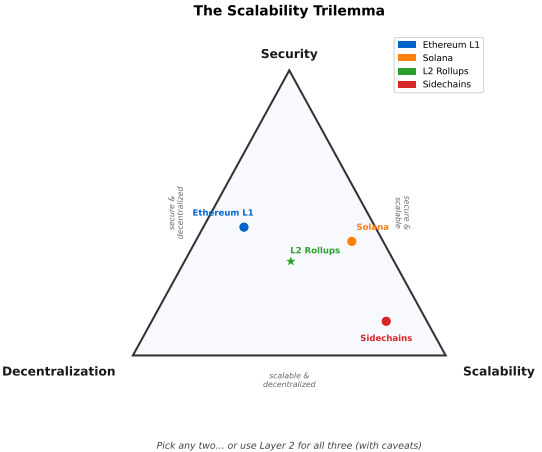
The Trilemma: Pick Two of Three

Think of a triangle with three corners. You can stretch toward any two corners, but the third one always gets worse.

Every blockchain must choose:

- **Fast** — handles lots of transactions
- **Safe** — very expensive to attack
- **Decentralized** — no single point of control

Ethereum chose safe and decentralized. That is why it is slow.



Layer 2 solutions try to cheat the trilemma: they borrow Ethereum's security while adding speed on top.

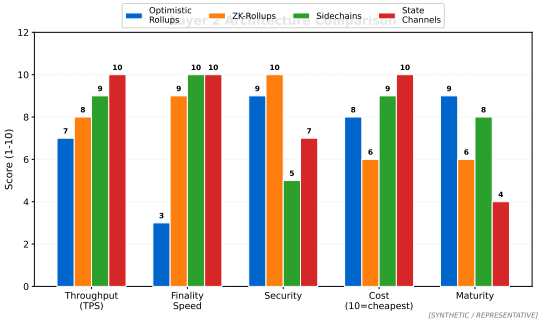
Layer 2: The Express Lane

Layer 1 is the main highway (Ethereum). Every car must wait in line.

Layer 2 is an express lane built alongside the highway. Cars zip through the express lane, then only the final result is posted back to the main highway.

The key idea:

- Do the work off the main chain (fast, cheap)
- Post a summary back to Ethereum (safe)
- Ethereum guards the final result



Layer 2 solutions process transactions off-chain but inherit Ethereum's security by posting compressed data back to it.

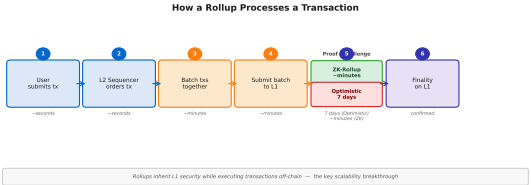
Rollups: Zip Files for Transactions

A **rollup** is like compressing 1,000 emails into a single zip file and sending just the zip.

How it works:

- Collect hundreds of transactions off-chain
- Compress them into one small package
- Post the package to Ethereum

Ethereum only stores the compressed summary, not every single transaction. That is why it is so much cheaper.



Rollups are the most popular Layer 2 approach because they inherit Ethereum's full security guarantees.

Two Flavors of Rollup

Optimistic Rollup

Analogy: A teacher who trusts homework is correct unless someone objects.

Assumes everything is valid.
If someone spots a mistake, they raise a flag within 7 days.

Examples: Arbitrum, Optimism, Base

Trade-off: Withdrawals take 7 days

ZK-Rollup

Analogy: A student who hands in homework with a mathematical proof that every answer is correct.

Proves correctness upfront.
No waiting period needed.

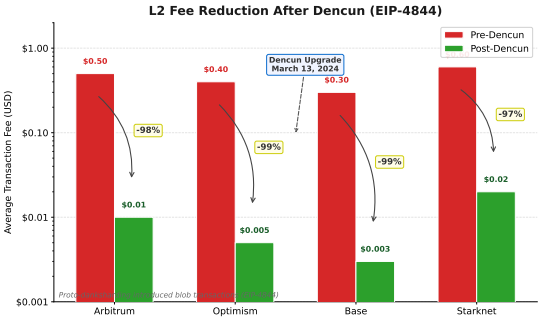
Examples: Starknet, zkSync, Scroll

Trade-off: Harder to build

Industry consensus (2025): Optimistic rollups have more users today. ZK-rollups are the long-term winner.

Arbitrum holds about 45 percent of all Layer 2 value as of early 2026. ZK-rollups are the fastest-growing segment.

How Much Cheaper Is Layer 2?



In March 2024, Ethereum launched the **Dencun upgrade**. It introduced a cheaper way for rollups to post data.

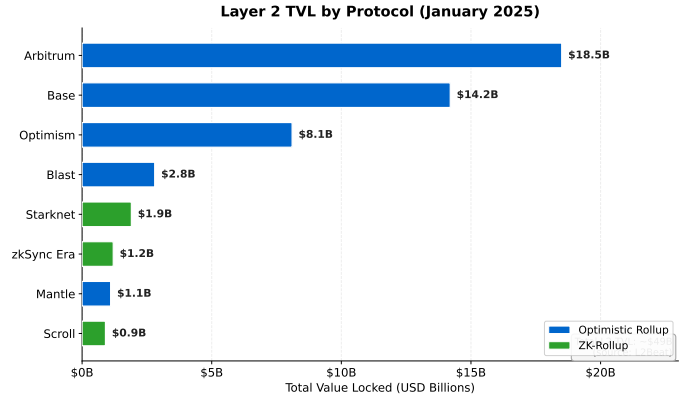
Before Dencun:
A swap on Arbitrum cost about 50 cents.

After Dencun:
The same swap costs about 1 cent.

98 percent fee reduction overnight.

The Dencun upgrade (EIP-4844) was the single biggest fee reduction in Ethereum's history.

Where Is the Money? Layer 2 Landscape



Arbitrum
Largest by value locked.
Optimistic rollup.

Base
Built by Coinbase.
Fast-growing, free to join.

Starknet / zkSync
ZK-rollups. Smaller today
but fastest growth.

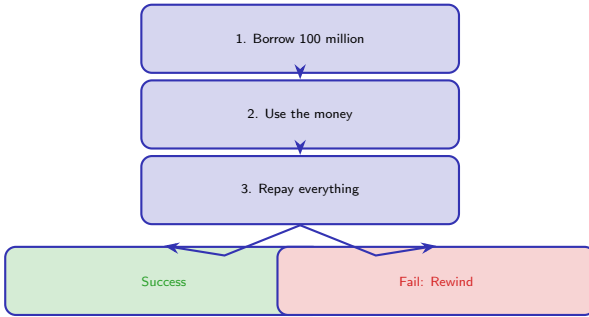
Layer 2 total value locked exceeded 54 billion dollars in 2025 — up 205 percent year-over-year.

What Is a Flash Loan?

Imagine an ATM that lets you withdraw 100 million dollars. You can use the money for anything you want — but you must return every cent within 12 seconds. If you do not, the ATM rewinds time and acts as if nothing happened.

That is a **flash loan**.

- Borrow any amount, zero collateral
- Use it and repay in one transaction
- If repayment fails, everything is canceled



Flash loans are uniquely DeFi — impossible in traditional finance. The lender has zero risk because failure simply erases the transaction.

Flash Loan vs Normal Loan

	Bank Loan	Flash Loan
Collateral	Required (your house, car)	None at all
Duration	Months or years	12 seconds
Identity check	Yes (credit score, ID)	No (anonymous)
What protects the lender	Legal contracts, courts	Code: if not repaid, it never happened
Risk to the lender	Borrower might default	Zero — transaction just reverses

The big shift: Code replaces courts. Math replaces trust.

Aave, the largest flash loan provider, processes over 10 billion dollars in flash loans every month.

What Can You Do with a Flash Loan?

Arbitrage

Token X costs 10 dollars on Exchange A and 10.50 on Exchange B. Buy low, sell high, repay — pocket the difference.

Collateral Swap

You have a loan backed by ETH but want to switch to USDC collateral. Flash loan lets you swap instantly without closing the loan.

Self-Liquidation

Your loan is about to be liquidated with a penalty. Use a flash loan to repay yourself first and save the penalty fee.

The power: You no longer need a million dollars to do million-dollar strategies. Just the skill to find the opportunity.

Flash loans democratize finance — anyone with knowledge can execute strategies that once required massive capital.

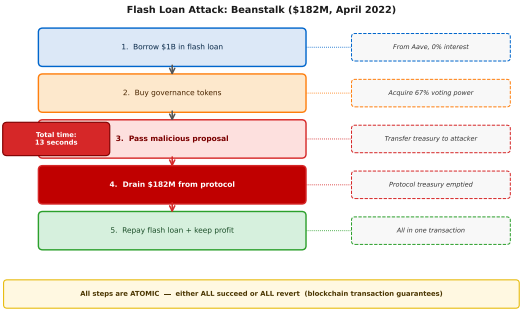
Flash Loan Arbitrage: Zero to 49,000 Dollars

Scenario: Token X costs 10 dollars on one exchange and 10.50 on another. You have zero starting capital.

Steps — all in one transaction:

1. Borrow 1,000,000 dollars (flash loan)
2. Buy 100,000 tokens at 10 dollars each
3. Sell 100,000 tokens at 10.50 dollars each
4. Repay the loan plus a tiny fee

Profit: about 49,000 dollars.
Starting capital: about 5 dollars (just the gas fee).



If any step fails — not enough tokens, price moved, cannot repay — the whole transaction is erased. You lose only the gas fee.

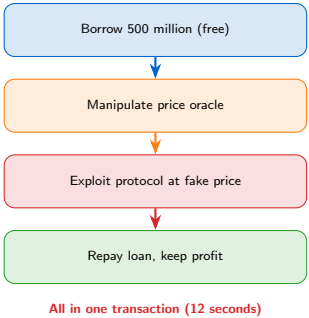
When Flash Loans Turn Dark

The same power that enables legitimate arbitrage also enables attacks. An attacker can:

1. Borrow millions for free
2. Manipulate a price feed
3. Exploit a vulnerable protocol
4. Repay the loan and keep the profit

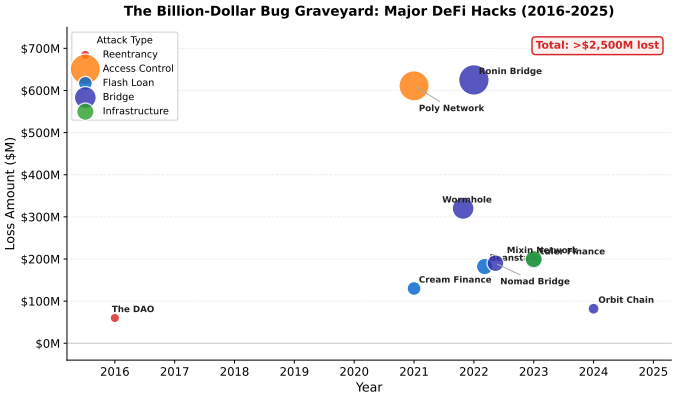
The attacker needs:

- Zero starting capital
- Just knowledge of a bug
- About 5 dollars for the gas fee



Flash loans do not cause the bugs — they amplify existing vulnerabilities by removing the capital barrier for attackers.

Real Flash Loan Attacks



Notable attacks:

- Euler Finance: 197 million dollars (March 2023)
- Beanstalk: 182 million dollars (April 2022)
- Cream Finance: 130 million dollars (Oct 2021)

Common pattern:

- Most attacks manipulated a price feed
- Attacker needed zero starting capital
- Over 500 million dollars stolen (2020–2023)

Oracle manipulation — tricking a protocol into reading a fake price — is the most common flash loan attack vector.

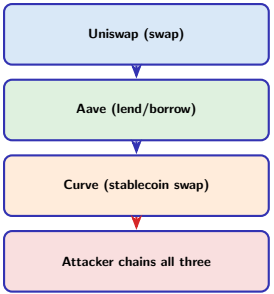
The Money Legos Problem

DeFi protocols snap together like LEGO bricks. That is called **composability** — any protocol can call any other protocol.

The good side:
Build powerful new tools by combining existing ones.

The bad side:
If one brick is rotten, everything stacked on top falls down.

An attacker can chain three protocols together in a single transaction, exploiting a weakness where they connect.



Each protocol is safe alone.
The vulnerability appears only when they interact.

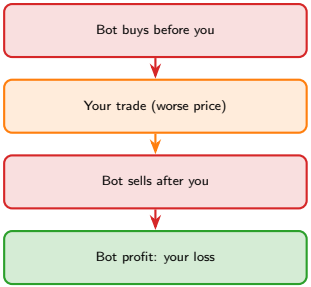
Composability is DeFi's greatest strength and its greatest weakness — "money LEGOs" can also be "attack LEGOs."

The Invisible Tax: MEV

MEV (Maximal Extractable Value) is like a bully who sees your lunch order, runs ahead of you in line, buys your sandwich, then sells it to you at a higher price.

The sandwich attack:

1. You submit a big swap
2. A bot sees your pending trade
3. The bot buys before you (front-run)
4. Your trade executes at a worse price
5. The bot sells right after (back-run)



MEV bots extracted over 600 million dollars from Ethereum users in 2022–2023. Protection: use private mempools like Flashbots.

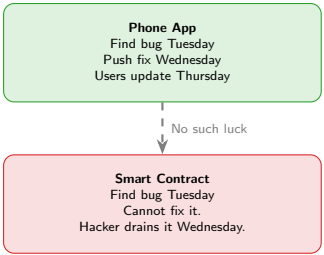
Why Bugs Are Permanent

When you find a bug in a phone app, the developer pushes an update overnight. Problem solved.

Smart contracts are different:

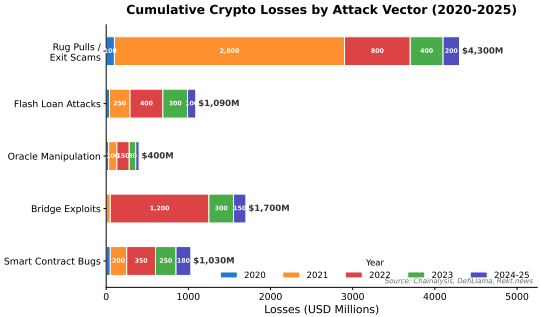
- Once deployed, the code cannot be changed
- There is no "undo" button
- A single bug can drain everything

Analogy: Imagine writing a vending machine's rules in permanent ink. If you made a mistake, the machine gives away free candy forever — and you cannot fix it.



Over 10 billion dollars has been stolen from smart contracts since 2016. One bug can destroy an entire protocol.

The Hall of Shame: Biggest Hacks



Top five hacks by loss:

1. **Ronin Bridge** — 625 million (Mar 2022)
Hackers stole private keys from validators.

2. **Wormhole** — 320 million (Feb 2022)
Bug in signature verification code.

3. **DMM Bitcoin** — 305 million (2024)
Exchange private key theft.

4. **Nomad Bridge** — 190 million (Aug 2022)
Anyone could drain it after a bad update.

5. **Euler Finance** — 197 million (Mar 2023)
Flash loan plus logic bug.

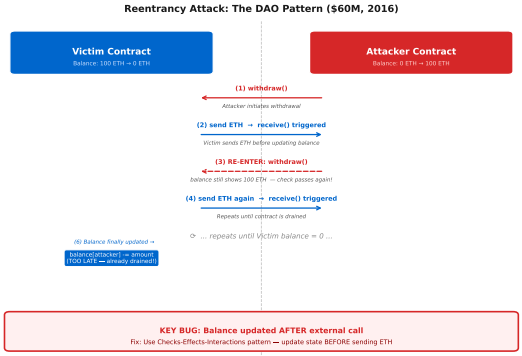
2024 alone saw 2.2 billion dollars stolen across 303 incidents. Bridge hacks dominate the largest losses.

Reentrancy: The Vending Machine Bug

Imagine a vending machine that gives you a candy bar, then checks if you paid. A clever person reaches in, grabs the candy, and triggers the machine again — before it realizes the first candy was taken.

The DAO hack (June 2016):

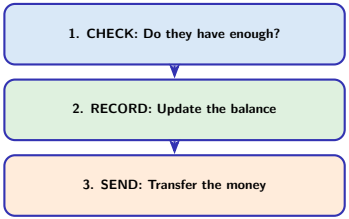
- Biggest Ethereum project at the time
- Attacker called “withdraw” over and over
- The contract sent money before recording it
- 60 million dollars drained



The DAO hack was so severe it split Ethereum into two chains: Ethereum (reversed the hack) and Ethereum Classic (kept it).

The Fix: Check, Record, Then Send

The defense is simple in concept:



This is called **Checks-Effects-Interactions**. Update your records *before* you send money, so a re-entry finds the balance already at zero.

Wrong order (vulnerable):

1. Check balance
2. Send money
3. Update balance

Attacker re-enters between steps 2 and 3

Right order (safe):

1. Check balance
2. Update balance
3. Send money

Re-entry finds balance already zero

The Checks-Effects-Interactions pattern is the single most important security rule in smart contract development.

Access Control: Who Holds the Keys?

The number one cause of losses in 2024 was not a clever code exploit. It was simply: **the wrong person got access**.

Common mistakes:

- Leaving an admin function open to anyone
- Storing private keys insecurely
- One person controlling everything

Parity Wallet (2017):
A random user accidentally became the owner of a shared library and deleted it. 150 million dollars frozen forever.

2024 breakdown:

Private key theft: 44 percent of all losses

Access control failures: leading attack category

DMM Bitcoin: 305 million (key theft)

WazirX: 230 million (exchange hack)

Best practices:

Multi-signature wallets (need 3 of 5 to approve)

Time-locks on critical changes

Role-based permissions

The simplest attacks are the most effective. Most money is lost not to clever code exploits but to stolen or mismanaged keys.

Oracle Manipulation: Lying to the Contract

An **oracle** tells a smart contract what is happening in the real world — like a price feed.

Analogy: Imagine a store that sets all its prices based on what one customer says. If that customer lies and says gold costs 1 dollar, the store sells gold for 1 dollar.

The attack:

1. Flash loan a huge amount
2. Crash or pump a token's price on one exchange
3. The oracle reads the fake price
4. Exploit the protocol using the wrong price

Real examples:

Harvest Finance: 34 million (Oct 2020)
Cream Finance: 130 million (Oct 2021)
Mango Markets: 114 million (Oct 2022)

The fix — use trustworthy oracles:

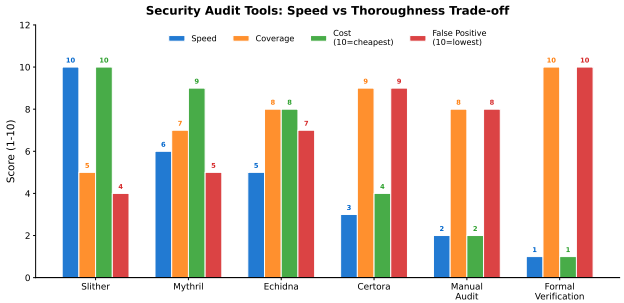
Chainlink: aggregates prices from dozens of sources

TWAP: averages the price over time, not just the current moment

Multiple sources: never trust a single price

A TWAP oracle is like checking the weather forecast from ten stations instead of just looking out one window.

The Security Toolbox



Before launch:

- Automated scanners (minutes)
- Human auditors (weeks, 50k–500k dollars)
- Formal proof that the code is correct

After launch:

- Bug bounties (up to 10 million dollars)
- 24/7 monitoring bots
- Insurance pools

No single tool catches every bug. Security is layers: automated scan, then human review, then public bounty.

Bridge Hacks: The New Frontier

A **bridge** moves assets between different blockchains.
Think of it as a ferry between two islands.

Why bridges get hacked:

- Complex multi-chain code
- Billions of dollars locked up
- Small groups of validators

Over 2 billion dollars stolen from bridges in 2022 alone.

Ronin Bridge (March 2022)
North Korean hackers stole 5 of 9 validator keys.
Loss: 625 million dollars.

Wormhole (February 2022)
Bug let attacker mint tokens without depositing.
Loss: 320 million dollars.

Nomad Bridge (August 2022)
Bad code update let *anyone* drain funds.
Hundreds of people copied the attack.
Loss: 190 million dollars.

Vitalik Buterin warned in 2022: "I am pessimistic about cross-chain applications" — bridges are the weakest link.

Before You Use a DeFi Protocol: The Safety Checklist

Green flags (safer):

- \+ Multiple independent audits
- \+ Active bug bounty program
- \+ Uses Chainlink or TWAP oracles
- \+ Multi-sig admin keys
- \+ Time-lock on upgrades
- \+ Open-source and verified

Red flags (dangerous):

- ! No audit or only one audit
- ! Anonymous team with no track record
- ! Single admin key can drain funds
- ! Uses single-source price oracle
- ! No time-lock on contract changes
- ! Promises unrealistic returns

Rule of thumb: If you cannot explain how a protocol makes money, you are the product.

An audit is a snapshot in time. Security is ongoing — look for protocols that invest in continuous monitoring and bounties.

The Cryptoeconomics Lens: Six Questions

1. PROBLEM: What coordination problem does scaling solve?
→ Ethereum is too slow for mass adoption.

2. INCENTIVES: Why do participants behave correctly?
→ Validators stake money they lose if they cheat. Auditors earn fees for finding bugs.

3. COSTS/BENEFITS: Who pays and who gains?
→ Users pay lower fees on L2. Sequencers earn from ordering transactions.

4. FAILURE MODE: What breaks it?
→ Centralized sequencers. Bridge key theft. Oracle manipulation.

5. DESIGN: What alternatives exist?
→ Optimistic vs ZK rollups. Sidechains vs channels. TWAP vs Chainlink oracles.

6. IMPROVEMENT: How could it be better?
→ Decentralized sequencers. ZK-verified bridges. Formal verification of all contracts.

Every design choice is a trade-off. Faster means less decentralized. Composable means more attack surface.

Apply these six questions to any blockchain system to understand its economics, risks, and design trade-offs.

Key Takeaways

Scaling

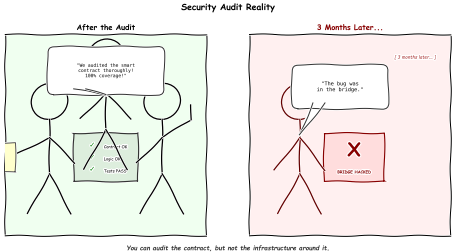
Layer 2 rollups make Ethereum 100 times faster and 98 percent cheaper. Optimistic rollups lead today; ZK is the future.

Flash Loans

Borrow millions for free in 12 seconds. Powerful for arbitrage. Dangerous as attack amplifiers. Zero capital needed.

Security

Bugs are permanent. Over 10 billion stolen. Defense: audits, bounties, monitoring, and the Checks-Effects-Interactions pattern.



Scaling creates attack surfaces. Attacks demand better security. Security shapes how we scale. They are inseparable.

What Is Next?

Scaling:

- Full danksharding: 100,000 transactions per second
- ZK-rollups reaching full Ethereum compatibility
- Layer 3: app-specific chains stacked on Layer 2

Security:

- AI-powered bug detection
- Formal verification becoming standard
- Real-time attack monitoring and circuit breakers

Flash loans and MEV:

- Private transaction pools to prevent front-running
- Protocol-level MEV protection
- Better oracle designs resistant to manipulation

The endgame:

Blockchain that is fast enough for everyone, cheap enough for daily use, and secure enough to hold the world's value.

We are still in the early innings. The technology is improving faster than the attacks — but the arms race never stops.

Appendix

Technical Deep Dives

The math, code, and economics behind the intuition

Click blue links in appendix slides to jump back to the main deck.

A1: How Rollup Fees Are Calculated

Total fee for a Layer 2 transaction:

$$\text{Fee}_{\text{L2}} = \text{L2 execution cost} + \text{L1 data posting cost}$$

Before Dencun (calldata):

- L1 data cost = (bytes of calldata) $\times$ 16 gas per byte $\times$ gas price
- A typical swap uses about 180 bytes of calldata
- At 30 gwei gas price: $180 \times 16 \times 30 = 86{,}400$ gas $\approx$ \$0.30–\$2.00

A1: How Rollup Fees Are Calculated

Total fee for a Layer 2 transaction:

$$\text{Fee}_{\text{L2}} = \text{L2 execution cost} + \text{L1 data posting cost}$$

Before Dencun (calldata):

- L1 data cost = (bytes of calldata) $\times$ 16 gas per byte $\times$ gas price
- A typical swap uses about 180 bytes of calldata
- At 30 gwei gas price: $180 \times 16 \times 30 = 86{,}400$ gas $\approx$ \$0.30–\$2.00

After Dencun (blob transactions, EIP-4844):

- Blob data has its own fee market, separate from execution gas
- Blob gas price: typically 1–10 gwei (vs 30+ for execution gas)
- Same swap: $\approx$ \$0.001–\$0.01 for L1 data
- **Result: 90–99% reduction in the L1 data component**

← Back to main slide: The Speed Problem

Blobs are temporary (pruned after about 18 days), which is why they are cheaper — rollups only need data available long enough for challenges.

A1: How ZK Proofs Work (Simplified)

Zero-Knowledge Proof: Prove that a statement is true without revealing the underlying data.

Analogy: Prove you know the combination to a safe by opening it in front of a witness — without telling them the numbers.

A1: How ZK Proofs Work (Simplified)

Zero-Knowledge Proof: Prove that a statement is true without revealing the underlying data.

Analogy: Prove you know the combination to a safe by opening it in front of a witness — without telling them the numbers.

Two main ZK proof systems:

Property	SNARK	STARK
Proof size	Small (∼300 bytes)	Larger (∼50 KB)
Verification cost	Very cheap on-chain	More expensive
Trusted setup	Required (one-time ceremony)	Not required
Quantum resistant	No	Yes
Used by	zkSync, Scroll	Starknet

A1: How ZK Proofs Work (Simplified)

Zero-Knowledge Proof: Prove that a statement is true without revealing the underlying data.

Analogy: Prove you know the combination to a safe by opening it in front of a witness — without telling them the numbers.

Two main ZK proof systems:

Property	SNARK	STARK
Proof size	Small (∼300 bytes)	Larger (∼50 KB)
Verification cost	Very cheap on-chain	More expensive
Trusted setup	Required (one-time ceremony)	Not required
Quantum resistant	No	Yes
Used by	zkSync, Scroll	Starknet

For rollups: The prover computes a proof that all transactions in a batch are valid. The L1 verifier checks this proof in constant time, regardless of how many transactions are in the batch.

$$\text{Verification cost} = O(1) \quad \text{regardless of batch size}$$

← Back to main slide: Two Flavors of Rollup

This constant-time verification is what makes ZK-rollups scale: 1,000 transactions cost the same to verify as 10.

A1: Optimistic Rollup — The Fraud Proof Game

How does Ethereum know if an optimistic rollup cheated?

1. The **sequencer** posts a batch of transactions and claims a new state root
2. For 7 days, anyone can submit a **fraud proof** challenging the state

A1: Optimistic Rollup — The Fraud Proof Game

How does Ethereum know if an optimistic rollup cheated?

1. The **sequencer** posts a batch of transactions and claims a new state root
2. For 7 days, anyone can submit a **fraud proof** challenging the state
3. If challenged, an **interactive bisection game** begins:
   - The challenger and sequencer narrow down the dispute to a single step
   - That single step is re-executed on Ethereum L1
   - If the sequencer was wrong, their stake is slashed (confiscated)
4. If no challenge after 7 days, the state is finalized

A1: Optimistic Rollup — The Fraud Proof Game

How does Ethereum know if an optimistic rollup cheated?

1. The **sequencer** posts a batch of transactions and claims a new state root
2. For 7 days, anyone can submit a **fraud proof** challenging the state
3. If challenged, an **interactive bisection game** begins:
   - The challenger and sequencer narrow down the dispute to a single step
   - That single step is re-executed on Ethereum L1
   - If the sequencer was wrong, their stake is slashed (confiscated)
4. If no challenge after 7 days, the state is finalized

Security assumption: At least 1 honest verifier must be watching. If nobody watches, a malicious sequencer could post invalid state.

Economic incentive: Verifiers earn rewards for catching fraud. Sequencers lose their stake if caught.

← Back to main slide: Two Flavors of Rollup

Arbitrum uses interactive fraud proofs. Optimism is migrating to a similar system. The 7-day delay is the main user-facing trade-off.

A2: Flash Loan — Pseudocode Walkthrough

How a flash loan works inside one Ethereum transaction:

```
// Step 1: Borrow
pool.flashLoan(borrower, token, amount);

// Step 2: The pool transfers tokens to borrower
token.transfer(borrower, amount);

// Step 3: Borrower executes their strategy
borrower.executeOperation(token, amount) {
// Buy on Exchange A, sell on Exchange B
// Or: repay a loan, swap collateral, etc.
}

// Step 4: Pool checks repayment
require(token.balanceOf(pool) >= amount + fee);

// If Step 4 fails: ENTIRE transaction reverts
// Nothing happened. Lender lost nothing.
```

A2: Flash Loan — Pseudocode Walkthrough

How a flash loan works inside one Ethereum transaction:

```
// Step 1: Borrow
pool.flashLoan(borrower, token, amount);

// Step 2: The pool transfers tokens to borrower
token.transfer(borrower, amount);

// Step 3: Borrower executes their strategy
borrower.executeOperation(token, amount) {
// Buy on Exchange A, sell on Exchange B
// Or: repay a loan, swap collateral, etc.
}

// Step 4: Pool checks repayment
require(token.balanceOf(pool) >= amount + fee);

// If Step 4 fails: ENTIRE transaction reverts
// Nothing happened. Lender lost nothing.
```

Key insight: The blockchain processes all four steps atomically. Either all succeed or all fail. The lender's risk is mathematically zero.

Fee: Aave charges 0.09%. dYdX charges 0% (they earn from the ecosystem).

← Back to main slide: What Is a Flash Loan?

Flash loans exploit Ethereum's atomic transaction model: the EVM either commits all state changes or reverts all of them.

A2: Flash Loan Attack — Beanstalk ($182M)

April 17, 2022: Attacker drains the Beanstalk stablecoin protocol.

Attack sequence (single transaction):

Step	Action	Value
1	Flash-borrow tokens from Aave, Uniswap, SushiSwap	$1,000,000,000

A2: Flash Loan Attack — Beanstalk ($182M)

April 17, 2022: Attacker drains the Beanstalk stablecoin protocol.

Attack sequence (single transaction):

Step	Action	Value
1	Flash-borrow tokens from Aave, Uniswap, SushiSwap	$1,000,000,000
2	Deposit into Beanstalk to acquire governance voting power	(now largest voter)

A2: Flash Loan Attack — Beanstalk ($182M)

April 17, 2022: Attacker drains the Beanstalk stablecoin protocol.

Attack sequence (single transaction):

Step	Action	Value
1	Flash-borrow tokens from Aave, Uniswap, SushiSwap	$1,000,000,000
2	Deposit into Beanstalk to acquire governance voting power	(now largest voter)
3	Propose malicious governance action: “transfer treasury to me”	(proposal created)

A2: Flash Loan Attack — Beanstalk ($182M)

April 17, 2022: Attacker drains the Beanstalk stablecoin protocol.

Attack sequence (single transaction):

Step	Action	Value
1	Flash-borrow tokens from Aave, Uniswap, SushiSwap	$1,000,000,000
2	Deposit into Beanstalk to acquire governance voting power	(now largest voter)
3	Propose malicious governance action: “transfer treasury to me”	(proposal created)
4	Vote YES with flash-borrowed voting power	(proposal passes)

A2: Flash Loan Attack — Beanstalk ($182M)

April 17, 2022: Attacker drains the Beanstalk stablecoin protocol.

Attack sequence (single transaction):

Step	Action	Value
1	Flash-borrow tokens from Aave, Uniswap, SushiSwap	$1,000,000,000
2	Deposit into Beanstalk to acquire governance voting power	(now largest voter)
3	Propose malicious governance action: “transfer treasury to me”	(proposal created)
4	Vote YES with flash-borrowed voting power	(proposal passes)
5	Execute proposal: treasury funds transferred to attacker	+$182,000,000
6	Repay all flash loans	−$1,000,000,000
	Net profit to attacker	**$182,000,000**

A2: Flash Loan Attack — Beanstalk ($182M)

April 17, 2022: Attacker drains the Beanstalk stablecoin protocol.

Attack sequence (single transaction):

Step	Action	Value
1	Flash-borrow tokens from Aave, Uniswap, SushiSwap	$1,000,000,000
2	Deposit into Beanstalk to acquire governance voting power	(now largest voter)
3	Propose malicious governance action: "transfer treasury to me"	(proposal created)
4	Vote YES with flash-borrowed voting power	(proposal passes)
5	Execute proposal: treasury funds transferred to attacker	+$182,000,000
6	Repay all flash loans	−$1,000,000,000
	Net profit to attacker	**$182,000,000**

Root cause: Beanstalk's governance allowed instant proposal execution with no time delay. The fix: governance timelocks (24–48 hour delay between proposal and execution).

← Back to main slide: When Flash Loans Turn Dark

The attacker did not exploit a bug in the code — they used the governance system exactly as designed. The design itself was the flaw.

A3: Reentrancy — Vulnerable vs Safe Code

Vulnerable pattern (The DAO style):

```
// VULNERABLE: sends ETH before updating balance
function withdraw(uint amount) public {
require(balances[msg.sender] >= amount);
msg.sender.call{value:  amount}(""); // external call FIRST
balances[msg.sender] -= amount; // state update AFTER
}
```

A3: Reentrancy — Vulnerable vs Safe Code

Vulnerable pattern (The DAO style):

```
// VULNERABLE: sends ETH before updating balance
function withdraw(uint amount) public {
require(balances[msg.sender] >= amount);
msg.sender.call{value:  amount}(""); // external call FIRST
balances[msg.sender] -= amount; // state update AFTER
}
```

Safe pattern (Checks-Effects-Interactions):

```
// SAFE: updates balance before sending ETH
function withdraw(uint amount) public {
require(balances[msg.sender] >= amount); // CHECK
balances[msg.sender] -= amount; // EFFECT (state update FIRST)
msg.sender.call{value:  amount}(""); // INTERACTION (external call LAST)
}
```

A3: Reentrancy — Vulnerable vs Safe Code

Vulnerable pattern (The DAO style):

```
// VULNERABLE: sends ETH before updating balance
function withdraw(uint amount) public {
require(balances[msg.sender] >= amount);
msg.sender.call{value: amount}(""); // external call FIRST
balances[msg.sender] -= amount; // state update AFTER
}
```

Safe pattern (Checks-Effects-Interactions):

```
// SAFE: updates balance before sending ETH
function withdraw(uint amount) public {
require(balances[msg.sender] >= amount); // CHECK
balances[msg.sender] -= amount; // EFFECT (state update FIRST)
msg.sender.call{value: amount}(""); // INTERACTION (external call LAST)
}
```

Extra protection — reentrancy guard:

```
bool locked = false;
modifier noReentrancy() {
require(!locked, "No re-entrancy");
locked = true;
_;
locked = false;
}
```

← Back to main slide: The Vending Machine Bug

OpenZeppelin's ReentrancyGuard is the industry standard. Always use it on functions that transfer value.

A3: The DAO Attack — Step by Step (June 2016)

The DAO: A decentralized investment fund that raised $150 million in ETH.

1. Attacker deposits 1 ETH into The DAO

A3: The DAO Attack — Step by Step (June 2016)

The DAO: A decentralized investment fund that raised $150 million in ETH.

1. Attacker deposits 1 ETH into The DAO
2. Calls `splitDAO()` to withdraw their 1 ETH
3. The DAO sends 1 ETH to the attacker's contract

A3: The DAO Attack — Step by Step (June 2016)

The DAO: A decentralized investment fund that raised $150 million in ETH.

1. Attacker deposits 1 ETH into The DAO
2. Calls `splitDAO()` to withdraw their 1 ETH
3. The DAO sends 1 ETH to the attacker's contract
4. Attacker's `receive()` function immediately calls `splitDAO()` again
5. The DAO checks the attacker's balance — it was never decremented (still shows 1 ETH)

A3: The DAO Attack — Step by Step (June 2016)

The DAO: A decentralized investment fund that raised $150 million in ETH.

1. Attacker deposits 1 ETH into The DAO
2. Calls `splitDAO()` to withdraw their 1 ETH
3. The DAO sends 1 ETH to the attacker's contract
4. Attacker's `receive()` function immediately calls `splitDAO()` again
5. The DAO checks the attacker's balance — it was never decremented (still shows 1 ETH)
6. The DAO sends another 1 ETH
7. Loop repeats: 1 ETH $\rightarrow$ 2 $\rightarrow$ 4 $\rightarrow$ 8 $\rightarrow$ ... until 3.6 million ETH drained

A3: The DAO Attack — Step by Step (June 2016)

The DAO: A decentralized investment fund that raised $150 million in ETH.

1. Attacker deposits 1 ETH into The DAO
2. Calls `splitDAO()` to withdraw their 1 ETH
3. The DAO sends 1 ETH to the attacker's contract
4. Attacker's `receive()` function immediately calls `splitDAO()` again
5. The DAO checks the attacker's balance — it was never decremented (still shows 1 ETH)
6. The DAO sends another 1 ETH
7. Loop repeats: 1 ETH $\rightarrow$ 2 $\rightarrow$ 4 $\rightarrow$ 8 $\rightarrow$ ... until 3.6 million ETH drained

Aftermath:

- $60 million stolen (at 2016 prices)
- Ethereum community voted to hard fork and reverse the theft
- Those who disagreed kept the original chain: **Ethereum Classic**
- This remains the most consequential smart contract hack in history

← Back to main slide: The Vending Machine Bug

The DAO hack led directly to the creation of formal security auditing practices for smart contracts.

A4: MEV — The Hidden Economics of Transaction Ordering

MEV (Maximal Extractable Value) = profit available by reordering, inserting, or censoring transactions within a block.

Who captures MEV?

- **Searchers**: Bots that find MEV opportunities (arbitrage, liquidations)
- **Builders**: Construct blocks to maximize value extraction
- **Validators**: Select the highest-paying block (via MEV-Boost auction)

A4: MEV — The Hidden Economics of Transaction Ordering

MEV (Maximal Extractable Value) = profit available by reordering, inserting, or censoring transactions within a block.

Who captures MEV?

- **Searchers**: Bots that find MEV opportunities (arbitrage, liquidations)
- **Builders**: Construct blocks to maximize value extraction
- **Validators**: Select the highest-paying block (via MEV-Boost auction)

MEV supply chain (post-Merge, PBS model):

$$\text{Searcher} \xrightarrow{\text{bundles}} \text{Builder} \xrightarrow{\text{bids}} \text{Relay} \xrightarrow{\text{best block}} \text{Validator}$$

Scale:

- Over $600 million extracted in 2023
- Approximately 90% of Ethereum blocks use MEV-Boost
- Users lose an estimated 0.1–0.5% per swap to MEV

← Back to main slide: The Invisible Tax: MEV

MEV is not going away — the question is whether it flows to validators (centralization risk) or is redistributed to users.

A4: Sandwich Attack — Worked Example

Scenario: Alice submits a swap of 100 ETH for USDC on Uniswap. A MEV bot spots this in the mempool.

Order	Transaction	Price Impact
1	**Bot front-run**: Buy ETH (pushes price up)	ETH: \$2,000 $\rightarrow$ \$2,010

A4: Sandwich Attack — Worked Example

Scenario: Alice submits a swap of 100 ETH for USDC on Uniswap. A MEV bot spots this in the mempool.

Order	Transaction	Price Impact
1	**Bot front-run**: Buy ETH (pushes price up)	ETH: \$2,000 → \$2,010
2	**Alice's swap**: Buys at the inflated price	Alice pays \$2,010 instead of \$2,000

A4: Sandwich Attack — Worked Example

Scenario: Alice submits a swap of 100 ETH for USDC on Uniswap. A MEV bot spots this in the mempool.

Order	Transaction	Price Impact
1	**Bot front-run**: Buy ETH (pushes price up)	ETH: \$2,000 → \$2,010
2	**Alice's swap**: Buys at the inflated price	Alice pays \$2,010 instead of \$2,000
3	**Bot back-run**: Sells ETH at higher price	ETH: \$2,010 → \$2,005
	Bot profit (from Alice's slippage)	\$500
	Alice's loss (extra cost)	−\$1,000

A4: Sandwich Attack — Worked Example

Scenario: Alice submits a swap of 100 ETH for USDC on Uniswap. A MEV bot spots this in the mempool.

Order	Transaction	Price Impact
1	**Bot front-run**: Buy ETH (pushes price up)	ETH: \$2,000 → \$2,010
2	**Alice's swap**: Buys at the inflated price	Alice pays \$2,010 instead of \$2,000
3	**Bot back-run**: Sells ETH at higher price	ETH: \$2,010 → \$2,005
	Bot profit (from Alice's slippage)	\$500
	Alice's loss (extra cost)	−\$1,000

Protection strategies:

- **Private mempools** (Flashbots Protect): Submit transactions privately so bots cannot see them
- **Tight slippage limits**: Set maximum acceptable price deviation (e.g., 0.5%)
- **Smaller trades**: Less profitable for bots to sandwich

← Back to main slide: The Invisible Tax: MEV

Flashbots Protect is free to use and prevents most sandwich attacks by hiding your transaction from public mempools.

A5: Who Gets Paid and Why — The Incentive Map

Actor	Role	Earns	Loses if They Cheat
L2 Sequencer	Orders transactions	Transaction fees	Stake (slashed)

A5: Who Gets Paid and Why — The Incentive Map

Actor	Role	Earns	Loses if They Cheat
L2 Sequencer	Orders transactions	Transaction fees	Stake (slashed)
Fraud Prover	Watches for invalid state	Reward from slashed stake	Gas cost if wrong
ZK Prover	Generates validity proofs	Prover fees	Computational cost

A5: Who Gets Paid and Why — The Incentive Map

Actor	Role	Earns	Loses if They Cheat
L2 Sequencer	Orders transactions	Transaction fees	Stake (slashed)
Fraud Prover	Watches for invalid state	Reward from slashed stake	Gas cost if wrong
ZK Prover	Generates validity proofs	Prover fees	Computational cost
Auditor	Reviews smart contract code	Audit fee ($50K–$500K)	Reputation
Bug Hunter	Finds vulnerabilities	Bounty ($1K–$10M)	Time invested

A5: Who Gets Paid and Why — The Incentive Map

Actor	Role	Earns	Loses if They Cheat
L2 Sequencer	Orders transactions	Transaction fees	Stake (slashed)
Fraud Prover	Watches for invalid state	Reward from slashed stake	Gas cost if wrong
ZK Prover	Generates validity proofs	Prover fees	Computational cost
Auditor	Reviews smart contract code	Audit fee ($50K–$500K)	Reputation
Bug Hunter	Finds vulnerabilities	Bounty ($1K–$10M)	Time invested
MEV Searcher	Finds arbitrage opportunities	MEV profit	Gas cost of failed bundles
Bridge Validator	Confirms cross-chain transfers	Validation fees	Stake (slashed)

A5: Who Gets Paid and Why — The Incentive Map

Actor	Role	Earns	Loses if They Cheat
L2 Sequencer	Orders transactions	Transaction fees	Stake (slashed)
Fraud Prover	Watches for invalid state	Reward from slashed stake	Gas cost if wrong
ZK Prover	Generates validity proofs	Prover fees	Computational cost
Auditor	Reviews smart contract code	Audit fee ($50K–$500K)	Reputation
Bug Hunter	Finds vulnerabilities	Bounty ($1K–$10M)	Time invested
MEV Searcher	Finds arbitrage opportunities	MEV profit	Gas cost of failed bundles
Bridge Validator	Confirms cross-chain transfers	Validation fees	Stake (slashed)

The core design principle: Every system actor must have more to lose from cheating than to gain. This is the economic foundation of blockchain security.

$$\text{Cost of attack} > \text{Profit from attack} \implies \text{System is secure}$$

← Back to main slide: Why Bugs Are Permanent

Cryptoeconomic security works because rational actors will not attack a system where the cost exceeds the reward.

A5: The Scaling-Security Design Space

Every Layer 2 design makes explicit trade-offs:

Solution	Inherits L1 Security?	Trust Model	Finality	TPS
Optimistic Rollup	Yes	1 honest verifier	7 days	2,000–4,000
ZK-Rollup	Yes	Math (zero trust)	Minutes	2,000–10,000
State Channel	Yes	Both parties online	Instant	Millions
Sidechain	No	Own validator set	Chain-specific	1,000–10,000
Validium	Partial	Off-chain DA committee	Minutes	10,000+

A5: The Scaling-Security Design Space

Every Layer 2 design makes explicit trade-offs:

Solution	Inherits L1 Security?	Trust Model	Finality	TPS
Optimistic Rollup	Yes	1 honest verifier	7 days	2,000–4,000
ZK-Rollup	Yes	Math (zero trust)	Minutes	2,000–10,000
State Channel	Yes	Both parties online	Instant	Millions
Sidechain	No	Own validator set	Chain-specific	1,000–10,000
Validium	Partial	Off-chain DA committee	Minutes	10,000+

Key insight: Rollups inherit L1 security because they post data to Ethereum. Sidechains do not — they rely on their own, typically smaller, validator set.

Design question for students: If you were building a DeFi protocol handling $1 billion, which L2 architecture would you choose and why?

← Back to main slide: The Trilemma

The trade-off is not just technical — it is economic. More security means higher costs, which means higher fees for users.

A5: Security Economics — The Cost of Not Auditing

Is a $500,000 audit worth it?

Cost item	Amount
Professional audit (Trail of Bits, 8 weeks)	$500,000
Bug bounty program (annual, Immunefi)	$100,000
Monitoring tools (Forta, OpenZeppelin Defender)	$50,000
Total security investment	**$650,000**

A5: Security Economics — The Cost of Not Auditing

Is a $500,000 audit worth it?

Cost item	Amount
Professional audit (Trail of Bits, 8 weeks)	$500,000
Bug bounty program (annual, Immunefi)	$100,000
Monitoring tools (Forta, OpenZeppelin Defender)	$50,000
Total security investment	**$650,000**

Cost of NOT investing in security:

- Average DeFi hack loss: $25 million (2024 data, 303 incidents, $2.2B total)
- Reputation damage: Protocol TVL drops 80–100% after a major hack
- Many hacked protocols never recover

ROI calculation: Every $1 spent on security saves an estimated $100+ in avoided losses.

← Back to main slide: Why Bugs Are Permanent

The Wormhole bug bounty paid $10 million for a critical vulnerability — the protocol held $300 million at the time. Best ROI imaginable.