dApp Analysis: Exploring the Complex System Oriented Approach

Giuseppe Destefanis

London SmartBridgeLab

www.smartbridgelab.co.uk

Episodes

01

Introduction to Complex Systems and dApps

02

The Complex System Oriented Approach for dApps

03

Analysing dApps at Different Levels of Granularity

04

Incorporating Software Metrics and Future Directions

Introduction to Complex Systems and dApps

Introduction to complex systems

Characteristics of complex systems

Overview of decentralized applications (dApps)

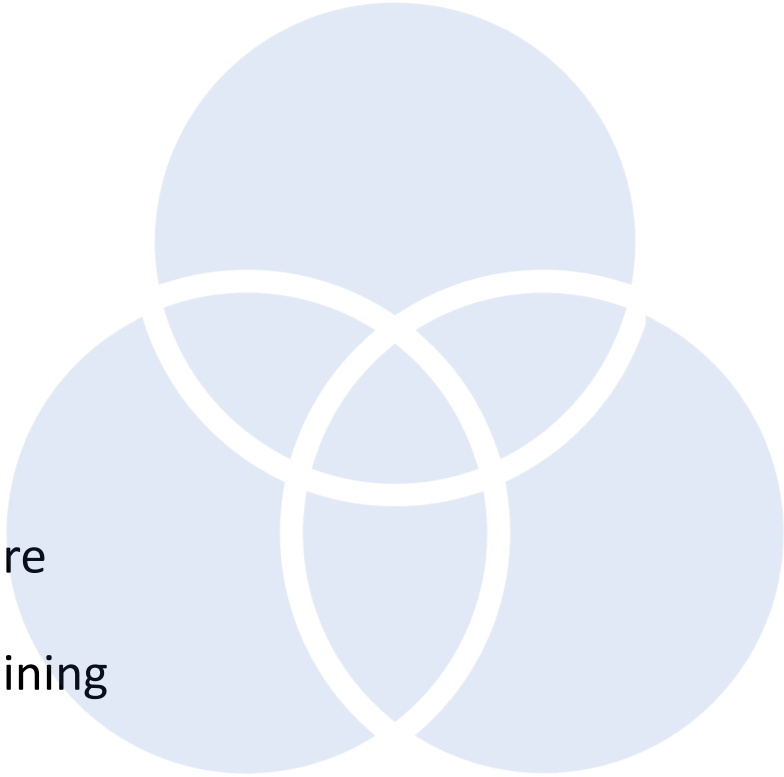
Challenges in understanding and analyzing dApps

Motivation behind applying complex systems theory to dApp analysis

Q&A

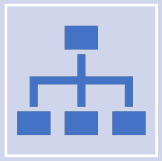
Complex Systems

A cross-disciplinary area focusing on systems where numerous components interact in varied ways, producing behaviors not easily predicted by examining the components alone



Complex Systems (II)

A Wide Range of Connected Elements: a network where each point is a unique part, all linked together. This network showcases the diversity of the system, allowing for many ways for actions and influences to move through it.



Structured Yet Flexible Organization: The way these parts connect is not random but follows a detailed structure that allows the system to be both stable and able to adapt. This structure is crucial for the system's ability to handle changes and challenges.

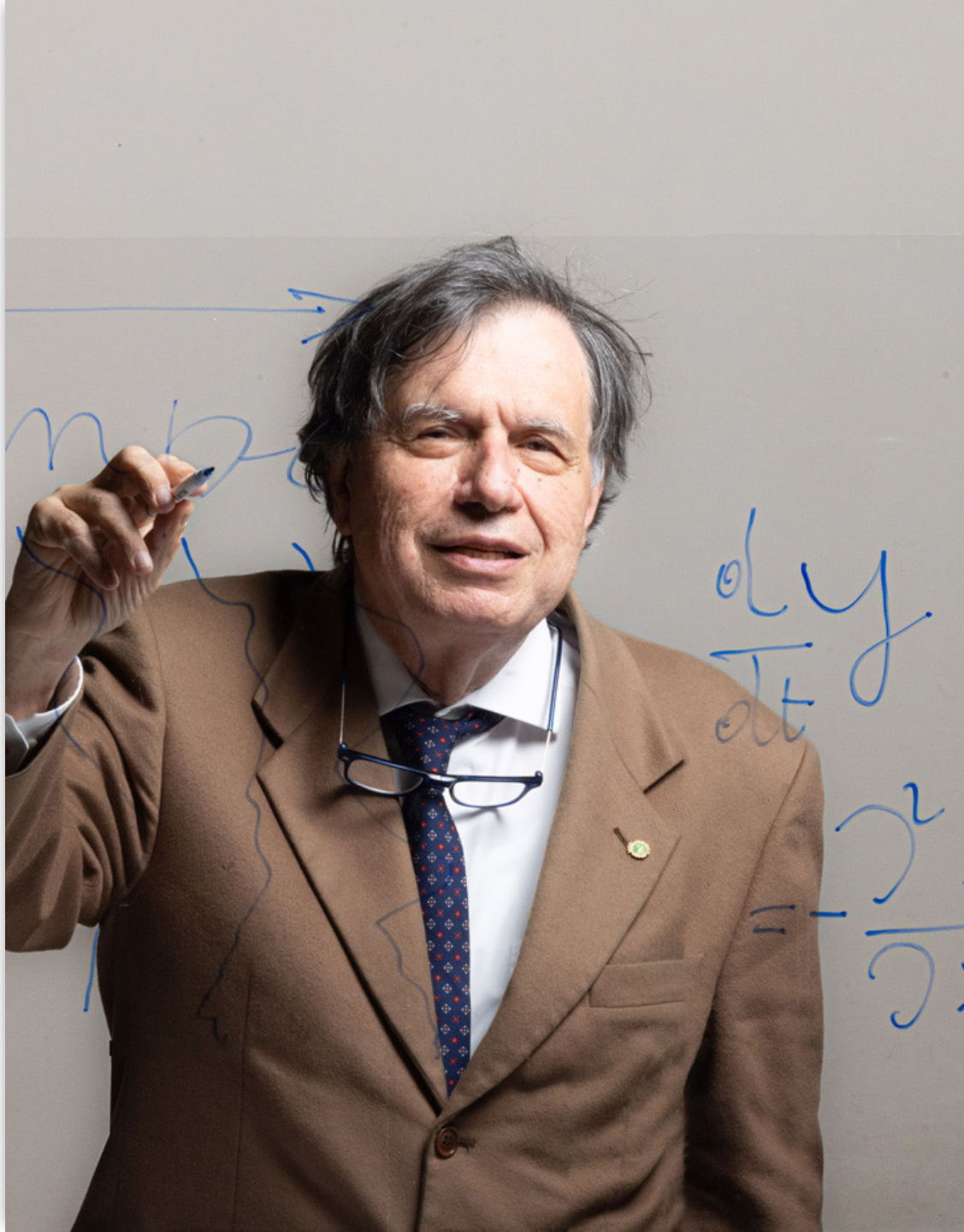
Complex System (III)

Unexpected Outcomes: one of the most interesting features of complex systems is their ability to produce outcomes or patterns that you wouldn't predict just by looking at the individual parts. This shows the system's ability to respond and adapt to different situations in unique ways.

Ability to Change and Grow: These systems can change and improve in response to new challenges or changes in their environment, ensuring they stay relevant and effective.

Parisi’s definition

- “Generally speaking, we could say that a complex system can stay in many different equilibrium states, while a simple system may stay only in one or a few equilibrium states.
- For example, an animal like a dog can do many different actions (e.g., play, sleep, eat, hunt); an animal can switch from one state to another state in a very short time ad the effect of a small perturbation, e.g., suddenly waking up after hearing a suspicious noise.
- In a nutshell what does not change over time (or change irreversibly) is not complex, while a system that can take many different forms or behaviors certainly is..”



Examples of Complex System (I)

- **The human brain**
- Composed of billions of interconnected neurons
- Emergent properties: consciousness, memory, learning, emotion
- Complex dynamics of neural activity patterns
- Plasticity and adaptability in response to experiences and stimuli

Examples of Complex System (II)

Biological Systems

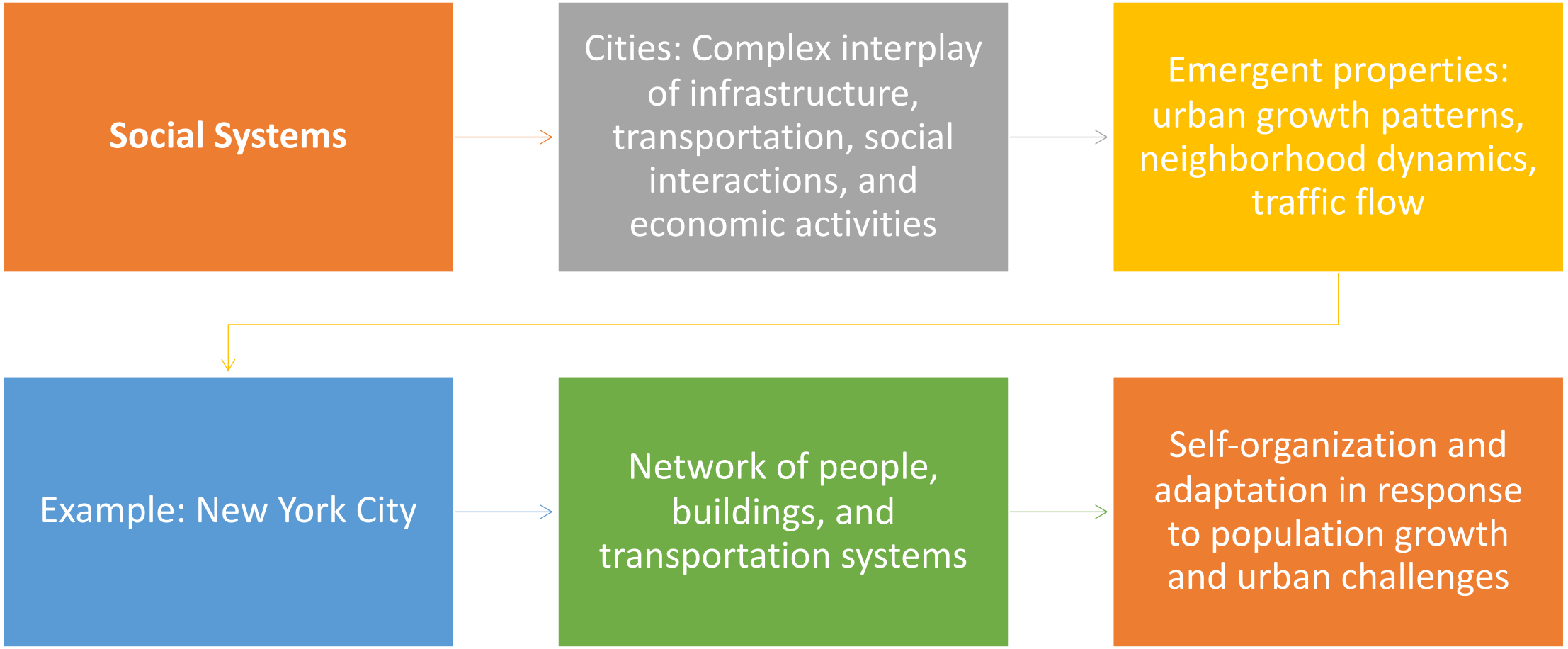
networks of interacting species and their environment;

Emergent properties: resilience, biodiversity, nutrient cycling

Example: The Amazon rainforest

Complex interplay of plants, animals, and microorganisms

Examples of Complex System (III)



Examples of Complex System (III)

Financial Markets

- Complex network of investors, financial institutions, and assets
- Emergent properties: market volatility, bubbles, crashes
- Nonlinear dynamics and feedback loops between market sentiment and asset prices
- Adaptive behavior of market participants in response to changing conditions

Examples of Complex System (IV)

Technological Systems: the Internet

- Global network of interconnected computers and devices
- Emergent properties: information flow, social networking, e-commerce
- Complex routing and data transmission protocols
- Robustness and adaptability in the face of network disruptions and attacks

Importance of Studying Complex Systems (I)

Understanding and Predicting Emergent Behaviors

Emergent behaviours arise from the interactions of system components

Examples: flocking behaviour in birds, traffic congestion in cities

Studying complex systems helps to identify the mechanisms driving emergent behaviours

Importance of Studying Complex Systems (II)

Enables the development of predictive models and simulations

Predicting emergent behaviours is key for:

- Anticipating and mitigating potential risks and unintended consequences
- Using beneficial emergent properties for innovation and problem-solving

Designing and Managing Complex Systems Effectively

- Understanding the principles of complex systems informs the design of more resilient and adaptive systems
- Incorporating feedback loops, modularity, and redundancy
- Enabling self-organization and distributed control

Effective management of complex systems requires:

- Monitoring and responding to system dynamics in real-time
- Promotting collaboration and communication among system components
- Continuously adapting to changing conditions and external perturbations

Properties of Complex Systems

Complex systems exhibit several distinctive properties that set them apart from simple systems

Interconnectedness

Complex systems consist of numerous components that are interconnected and interact with each other

The connections between components can be physical, informational, or functional

Example: In a social network, individuals are interconnected through various types of relationships

Diversity of Components

Complex systems often comprise a wide variety of components with different characteristics and behaviors

This diversity contributes to the system's robustness and adaptability

Example: An ecosystem contains a diversity of species, each with its own unique role

Adaptability

Complex systems have the ability to adapt and change in response to internal and external stimuli

Adaptation allows the system to maintain its functionality and stability in the face of perturbations

Example: Financial markets adapt to new information and changing economic conditions

Emergent Behaviour

Complex systems exhibit behaviours and properties that emerge from the interactions of their components

Emergent behaviours cannot be predicted by studying individual components in isolation

Example: the collective intelligence of an ant colony emerges from the interactions of individual ants

Non-Linearity

In complex systems, the relationship between cause and effect is often non-linear

Small changes in initial conditions can lead to disproportionately large effects (the "butterfly effect")

Example: In weather systems, small perturbations can trigger large-scale changes in weather patterns

Dynamic Nature

Complex systems are constantly evolving and changing over time

The system's behaviour and properties can shift as the interactions among components change

Example: Cities are dynamic systems that evolve in response to population growth, economic developments, and technological advancements

Complex Feedback Loops

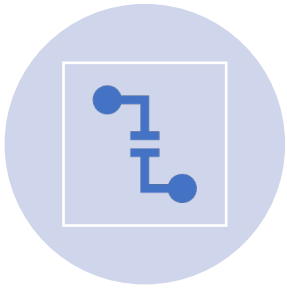
Complex systems often involve intricate feedback loops, where the output of the system influences its input

Feedback loops can be positive (amplifying) or negative (stabilising) and can lead to unexpected system behaviours

Example: In climate systems, the melting of Arctic ice creates a positive feedback loop that amplifies global warming

What about a Java system?

Class Networks: we can imagine every class in a Java system as a point in a network. The links form when one class uses another, creating a map of interactions that define the system's structure.

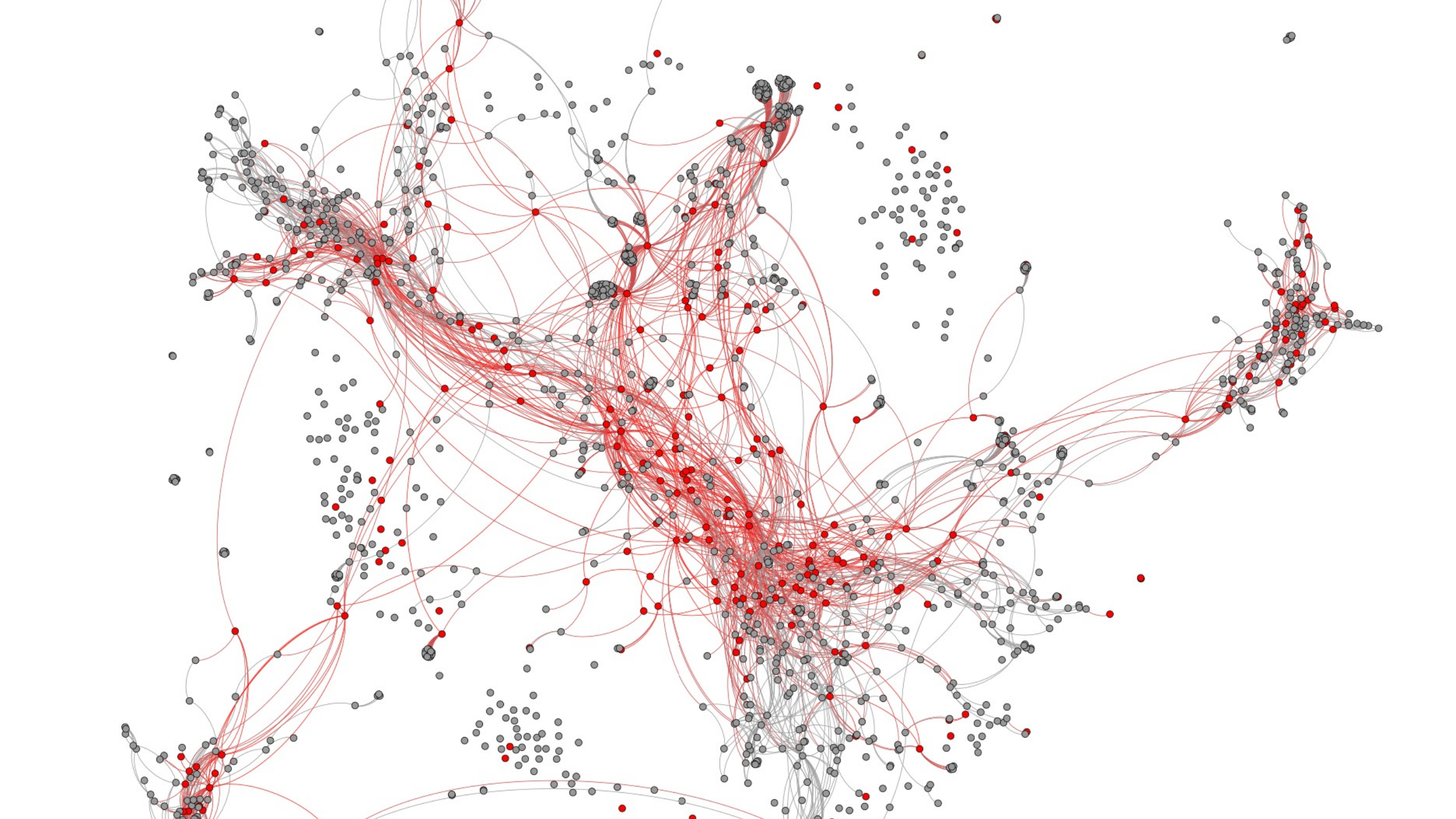


Connections Between Classes: the connections represent how classes communicate. E.g., a line can be drawn when a class calls a method from another class.

This view of Java systems mirrors the idea of complex networks. It's not just about individual classes but how they work together, influence each other, and contribute to the system’s overall behavior.

Seeing a Java system this way helps developers understand how tightly classes are connected, where the critical points are, and how changes in one area might affect others. It’s about spotting patterns, improving design, and making the system more robust.

For developers, this perspective can help visualize large or complex Java systems. It aids in making informed decisions about architecture, identifying potential issues before they become problems, and optimizing how the system works.



- What about a dApp?

dApps

- A decentralized application (dApp) is a type of software application that runs on a blockchain.

dApps

- A decentralized application (dApp) is a type of software application that runs on a blockchain.
- Decentralized Nature: dApps function across a network of distributed nodes. This decentralization ensures that no single entity controls the app.

dApps

- A decentralized application (dApp) is a type of software application that runs on a blockchain.
- Decentralized Nature: dApps function across a network of distributed nodes. This decentralization ensures that no single entity controls the app.
- Built on Blockchain: dApps are primarily developed on blockchain platforms like Ethereum.

dApps

- A decentralized application (dApp) is a type of software application that runs on a blockchain.
- Decentralized Nature: dApps function across a network of distributed nodes. This decentralization ensures that no single entity controls the app.
- Built on Blockchain: dApps are primarily developed on blockchain platforms like Ethereum.
- Smart Contracts: At the core of every dApp are smart contracts, which automate transactions and enforce the rules of the dApp without the need for intermediaries.

dApps

- Open Source and Incentivized: Most dApps are open source, meaning their code is available for review and improvement by anyone. They often have an in-built incentive system, rewarding users with tokens or digital assets for contributing to the network.
- Use Cases: dApps are used for a variety of applications, from financial services and games to social media and decentralized marketplaces. Their applications are continually expanding as the technology matures.

Uniswap-like dApp

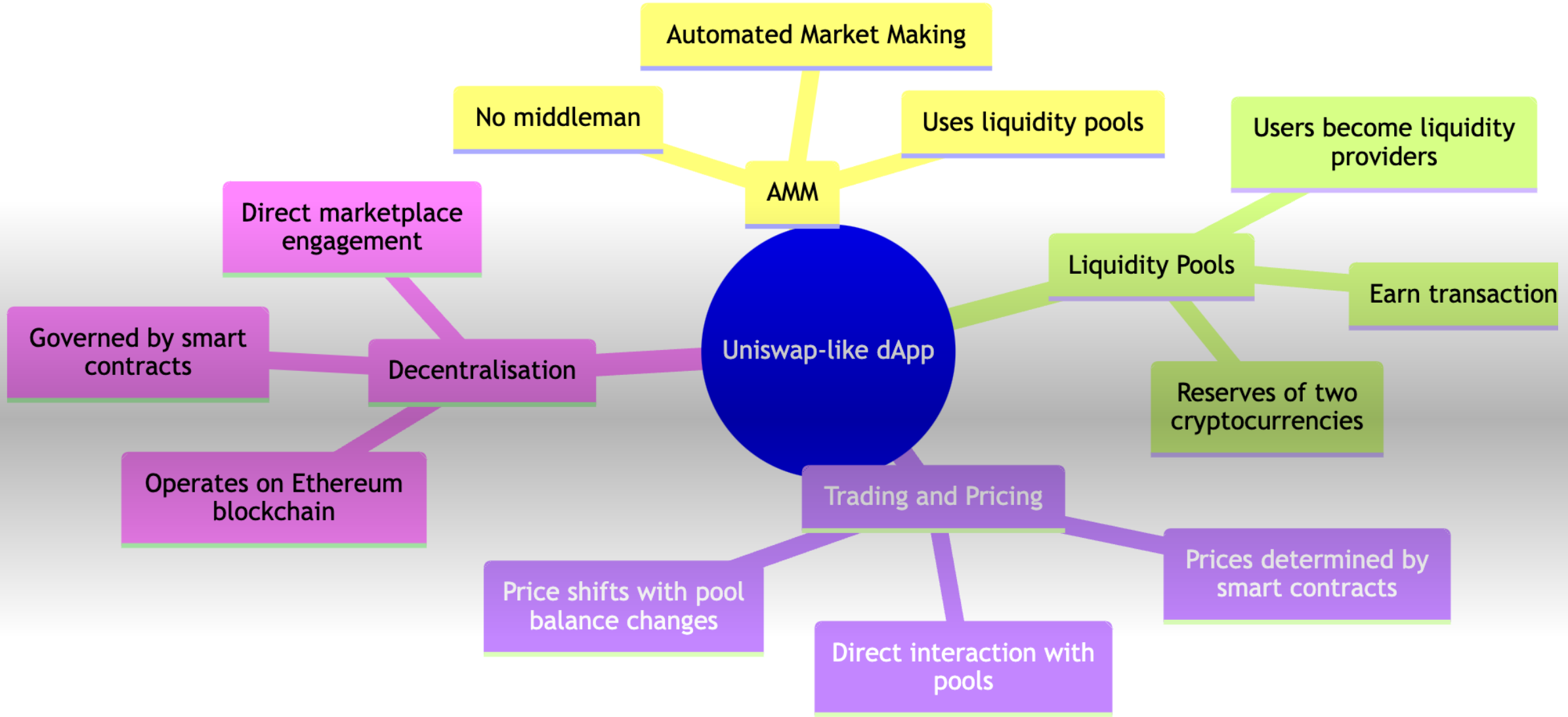
Automated Market Making: AMM platforms remove the need for traditional market matchmakers by utilizing liquidity pools. This innovation allows for continuous trading without intermediaries, making the process more efficient and open.

Liquidity Pools: These are the heart of AMM platforms, comprising reserves of two types of assets (e.g., Ethereum and a stablecoin). Users contribute to these pools and become liquidity providers, receiving transaction fees as rewards, similar to earning interest but with more risk and potential for higher returns.

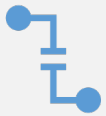
Uniswap-like dApp

Direct Trading and Dynamic Pricing: users trade directly with these liquidity pools. Prices are not set by order books but are determined by a formula based on the current ratio of assets in the pool, ensuring immediate and fair trade executions.

Decentralized Operation: built on blockchain technology like Ethereum, these platforms operate without a central authority. This setup gives users full control over their trades and assets, with all actions governed securely by smart contracts.


Automated Market Making
No middleman
Uses liquidity pools
AMM
Direct marketplace engagement
Governed by smart contracts
Decentralisation
Operates on Ethereum blockchain
Uniswap-like dApp
Users become liquidity providers
Liquidity Pools
Earn transaction
Reserves of two cryptocurrencies
Trading and Pricing
Price shifts with pool balance changes
Direct interaction with pools
Prices determined by smart contracts

Is this dApp a CS?



Interconnectedness: the dApp's numerous liquidity pools represent a network of interconnected parts, where each pool is linked to others through the shared practice of AMM and the overarching smart contract architecture. This network forms a web of financial interactions and dependencies.

Diversity of Components: each liquidity pool operates under a uniform set of rules but is distinguished by its unique assortment of assets, trading volume, and user-provided liquidity. This variety enhances the overall strength and adaptability of the system.

Adaptability: the system exhibits a high degree of adaptability, adjusting to changes such as significant trades or shifts in market sentiment. Liquidity providers can freely enter and exit pools, while traders can execute trades at any time, all of which contribute to the system's fluidity and its ability to reach new equilibrium states.

Emergent Behavior: the pricing mechanism within each pool—determined by the relative quantities of the two assets and the AMM algorithm—is a prime example of emergent behavior. It is not dictated by any single participant but rather arises from the collective action of all traders and liquidity providers.

Non-linearity: The relationship between trading activities, liquidity provision, and price is highly non-linear. Large trades can lead to significant price impacts, which in turn can trigger cascading effects as other participants respond to the new prices.

Dynamic Nature: the system can rapidly transition between states in response to small perturbations—a large trade or a sudden shift in market conditions can quickly alter a pool's state, much like an animal reacting to its environment.

Complex Feedback Loops: the governance model introduces complex feedback loops into the system, where token holders can propose and vote on changes that may alter the dApp's rules and parameters, influencing the behavior of all other components.

dApp Architecture

Factory Contract: responsible for creating new smart contracts for each cryptocurrency pair, acting as a registry and deploying an Exchange Contract for new market pairs as needed

Exchange Contract (or Pool Contract): dedicated to handling trades and liquidity for specific token pairs, these contracts use the Automated Market Making (AMM) algorithm to set prices based on the current supply of the tokens within the pool

Router Contract: serves to bridge users with Exchange Contracts, streamlining the trade execution and liquidity management by directing interactions to the correct Exchange Contracts for the user's chosen token pairs

dApp Architecture (II)

Liquidity Pool Tokens (LP Tokens) Contract: issues LP tokens to users who provide liquidity, representing their share in the pool. These tokens can be exchanged back for the underlying assets at any time, effectively managing the issuance and redemption process.

Governance Contract: facilitates a governance system where token holders have the power to vote on protocol changes. This contract oversees the proposal, voting, and implementation phases of governance decisions.

WETH Contract: addresses the need for trading Ethereum (ETH) in a tokenized form within pools that primarily handle ERC-20 tokens by wrapping ETH into Wrapped Ether (WETH), allowing for direct trades against other ERC-20 tokens.

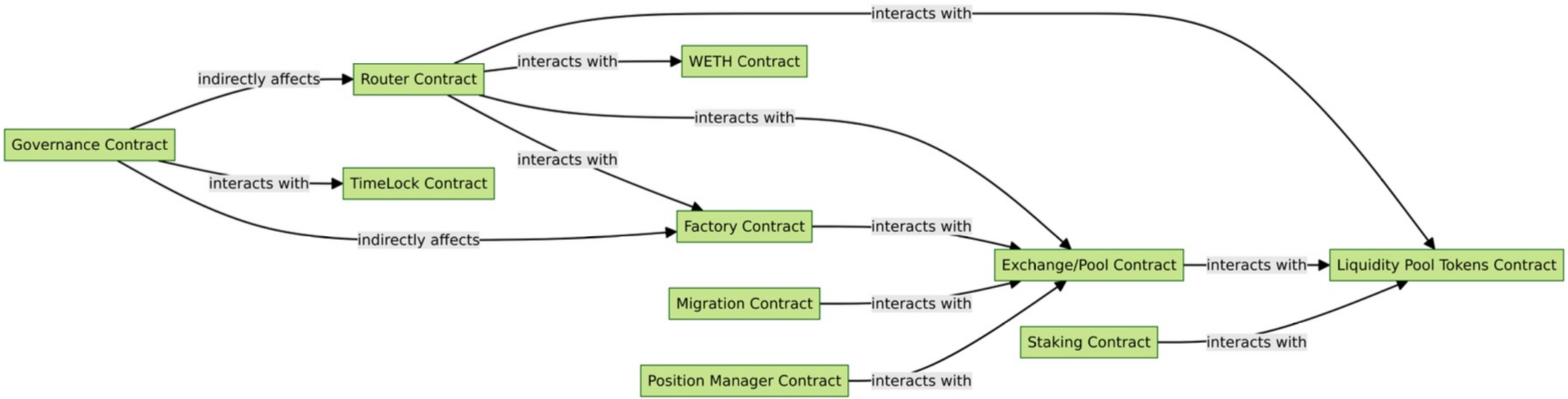
dApp Architecture (III)

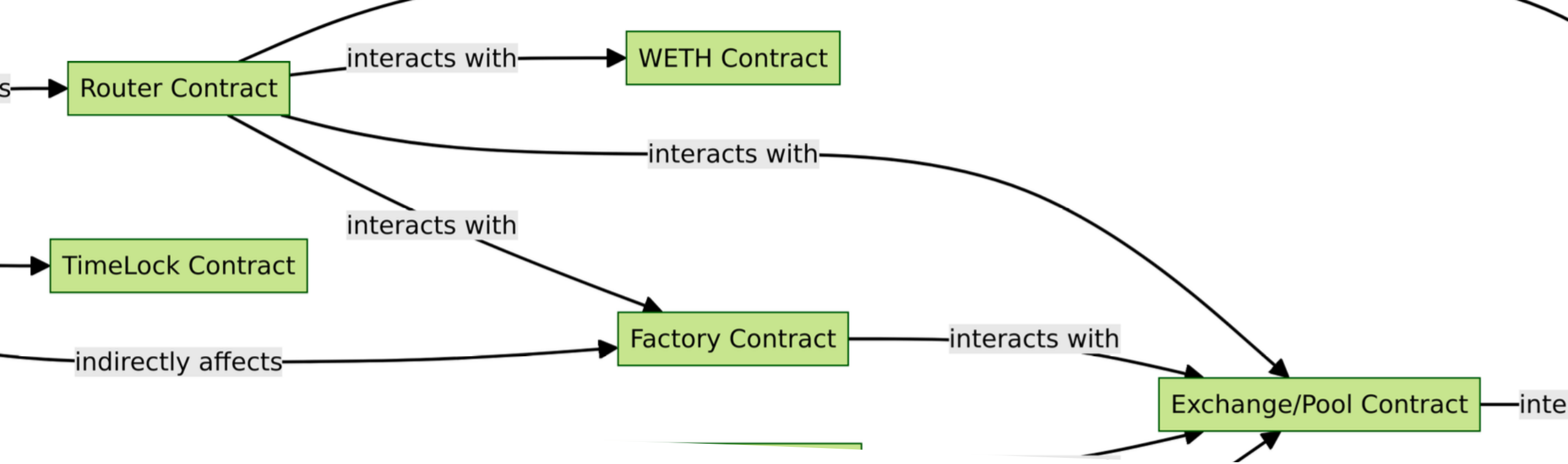
Staking Contract: offers users the opportunity to stake their LP tokens or other supported tokens to earn rewards, which can be derived from trading fees or other incentive mechanisms designed within the dApp.

TimeLock Contract: introduces a delay between the approval of governance proposals and their execution, providing a window for community response to new changes.

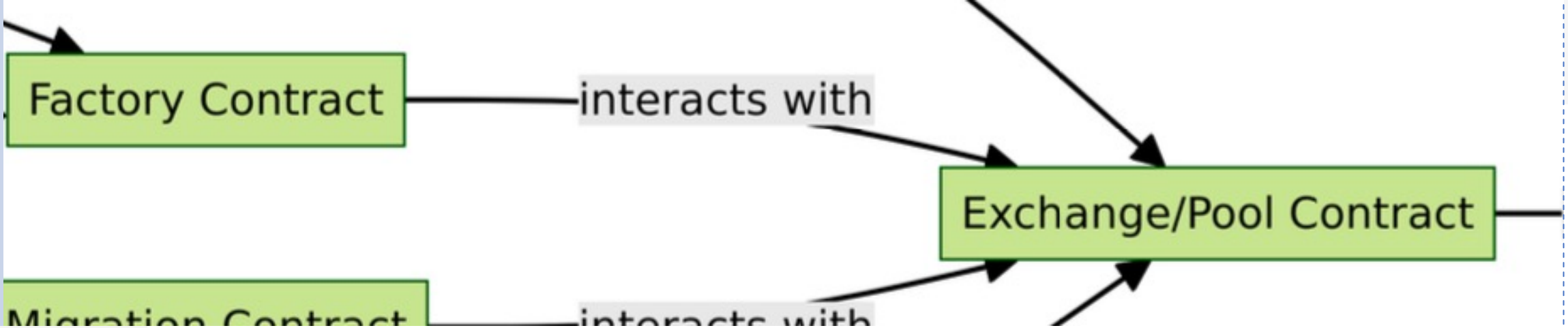
Migration Contract: supports the dApp through version upgrades by facilitating the transfer of liquidity from one version to the next, ensuring a smooth transition for liquidity providers.

Position Manager Contract: a feature akin to what was introduced in Uniswap V3, allowing for concentrated liquidity positions within specific price ranges, managing the allocation, modification, and removal of capital.


Governance Contract
indirectly affects
Router Contract
interacts with
WETH Contract
interacts with
Exchange/Pool Contract
interacts with
Liquidity Pool Tokens Contract
interacts with
TimeLock Contract
indirectly affects
interacts with
Factory Contract
interacts with
Migration Contract
interacts with
Staking Contract
interacts with
Position Manager Contract
interacts with



- The Router Contract interfaces with the WETH Contract for wrapping and unwrapping Ether, and with the Exchange/Pool Contract for trade execution and liquidity management.



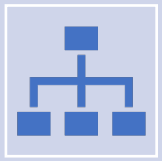
- The Factory Contract is tasked with creating new Exchange/Pool Contracts for different token pairs, essential for the dApp’s trading functionality

External dependencies

Highlighted in the network is the interaction with the WETH Contract, which is external to the dApp, illustrating the dApp's reliance on external components for essential operations like trading ETH.

This reliance exposes the dApp to risks associated with those contracts, such as vulnerabilities or downtime in the WETH Contract potentially impacting the dApp's ability to process ETH trades.

Modular architecture



The graph demonstrates the modular and interdependent architecture of the dApp, allowing for an analysis of how the system responds to changes and external events.

For instance, if an external event caused a surge in interactions with the WETH Contract, it could create bottlenecks impacting the Router Contract’s efficiency.

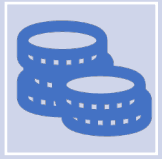
Contract level

At the contract level, we examine the network of smart contracts and how they interact with one another.

We can explore the network of smart contracts that constitute the backbone of a dApp, focusing on how these contracts, as distinct entities, interact and form the foundation of the application.

Function level

Function-level analysis focuses on the interactions between individual functions across different contracts. This granularity helps identify specific points of failure that might not be apparent at the contract level.



Let’s consider a function in the Liquidity Pool Tokens Contract that issues LP tokens based on the liquidity provided. The function’s logic might involve complex calculations to determine the amount of LP tokens to issue based on the current state of the liquidity pool.

Function level (II)

Suppose this function contains a rounding error when calculating the user's share or fails to properly handle integer overflows (for example, when large amounts of liquidity are added to the pool).

In such cases, an attacker could deliberately create transactions that exploit these vulnerabilities to either receive more LP tokens than they are entitled to or disrupt the token distribution mechanism, affecting the fairness and integrity of the liquidity pool.

Complex Systems Oriented Approach for dApps Analysis

Giuseppe Destefanis
Department of Computer Science
Brunel University London, UK
giuseppe.destefanis@brunel.ac.uk

***Abstract*—This position paper discusses on applying complex systems theory to analyse decentralised applications (dApps). It highlights the need for understanding smart contract networks' dynamics that are at the base of such platforms. Through a discussion that includes architecture, use cases, and both network and function-level scrutiny, including traditional and new software metrics, we argue how insights from complex systems can improve the understanding of dApps' behavior, vulnerabilities, and areas for optimisation. This position paper lays a foundation for enhancing the resilience, efficiency, and security of dApps within the blockchain environment.**

***Index Terms*—complex systems, smart contracts, dApps, blockchain**

I. Introduction

Complex systems are a cross-disciplinary area of science focusing on the study of systems where numerous components interact in different ways, producing behaviours that are not easily predicted by examining the components alone. These systems are distinguished by their vast set of different, interconnected elements and complex organisational frameworks. among the possible properties) not obvious from the properties of the individual parts. This concept is interesting for the study of decentralised systems and blockchain technology, as it helps in capturing the non-linear interactions and the collective behavior of smart contracts within dApps, which emerge from the interplay of their constituent smart contracts.

This position paper aims to examine how applying complex systems theory can improve our understanding of decentralised applications. Through the lens of a case study dApp similar to Uniswap[1], an extensive, open marketplace on the Ethereum blockchain for cryptocurrency exchanges without traditional goods or services, we will explore how this theoretical approach can yield more profound insights into dApps' architecture and behaviours. It stands as a paradigm of decentralisation, operating autonomously without any central authority, and is instead driven by a network of smart contracts and active user engagement.

To grasp the workings of this Uniswap-like dApp, let us consider the following components.

Conclusion

Read the provided introductory materials on complex systems and dApps

Write a short reflection (200-300 words) on how the characteristics of complex systems relate to dApps, based on your understanding