Virtualización, Ciberseguridad e IA en FinTech y Cripto

Grado de Ciberseguridad e Inteligencia Artificial – Arquitectura de Sistemas Virtualizados

---

Francisco López Lasanta

> Status: Connected_

¿Qué vais a ver hoy?

- **No** es una charla de trading “visual”.
  No veremos gráficas ni indicadores.
- Hablaremos de los **sistemas** que **evalúan** eventos y **ejecutan** decisiones automáticamente.

- Cómo una **decisión se transforma en código**.
- Por qué el **software gestiona riesgo financiero**.
- Qué papel juegan **virtualización, seguridad e IA**.
- Cuando algo falla en este entorno, **no hay botón de undo**.

¿Quién soy yo?

Francisco López Lasanta

Head of Trading Platform @ GSR

- Technical Manager @ Skyscanner (2016-2018)
- Lead Architect @ Ericsson (2010-2016)
- Engineering Manager @ Optimi (2003-2010)
- Ingeniero Informático (UGR)

$ whoami --locations

Malaga · Granada · Atlanta · Barcelona · London · Malaga

Punto de Partida:
Construir una Plataforma de **Trading**

El objetivo no es hacer trading.

Es construir el sistema que lo ejecuta y genera dinero.

- Mercados electrónicos en tiempo real
- Capital real en riesgo
- Decisiones ejecutadas por software
- Sistemas operando 24/7 sin intervención humana

> Aquí el software no apoya al negocio:
>
> **el software es el negocio $$$**

¿Qué es Trading?

> ***Trading*** *= tomar decisiones bajo incertidumbre para generar retorno económico gestionando el riesgo de forma controlada*

Ejemplos:

- El riesgo aumenta → se reduce exposición
- El riesgo supera un límite → se cierra la posición

El sistema de Trading:

- ✕ No opina
- ✕ No intuye
- ⚡ Evalúa eventos
- ⚡ Ejecuta decisiones

¿Qué es Cripto?

Cripto es un sistema financiero basado en **software y criptografía**, no en intermediarios.

Bitcoin (2008)

Primer sistema monetario descentralizado.

Libro mayor distribuido (blockchain).

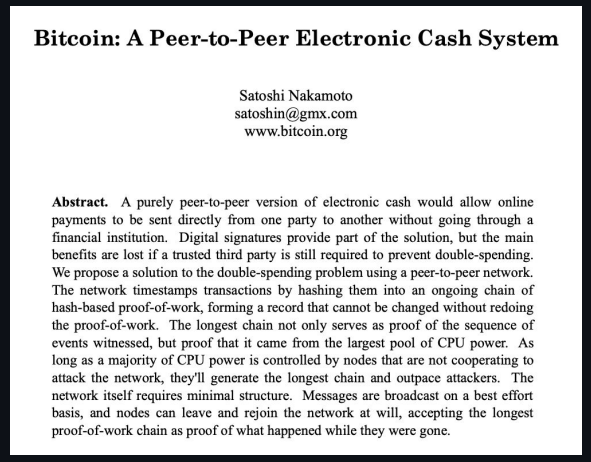
Consenso

PoW: seguridad por cómputo.

PoS: seguridad por capital bloqueado.

Reglas

El software define las reglas. No hay reversión manual. El riesgo es técnico.



Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

Mercados Electrónicos: TradFi vs Cripto

Finanzas Tradicionales

- Mercados regulados
- Horarios definidos
- Intermediarios
- Errores a veces reversibles

Cripto

- 24/7
- Global
- Sin intermediarios
- Sin reversión

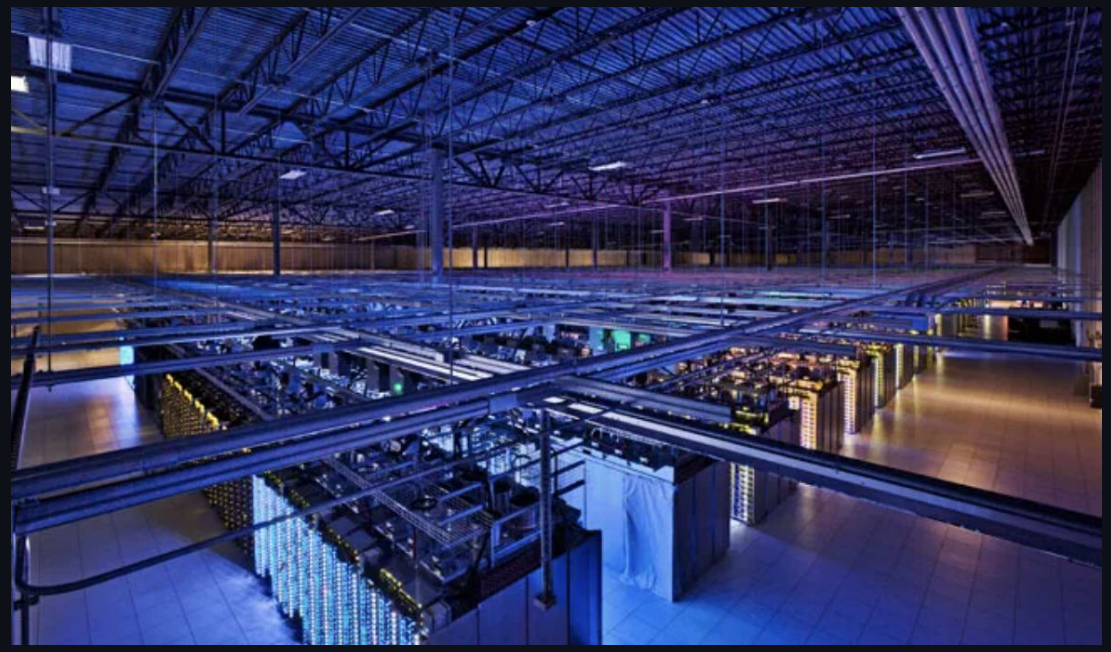
> Modelos:
>
> CEX (Centralized Exchanges) off-chain: Binance
>
> DEX (Decentralized Exchanges) on-chain: HyperLiquid

*Desde el punto de vista del software, ambos son **mercados electrónicos**.*

Trading (Hoy) = Software

El trading moderno se ejecuta mediante sistemas de software.

- APIs en lugar de interfaces gráficas
- Señales en lugar de intuición humana
- Código en lugar de decisiones manuales



Trading **no son gráficas en una pantalla**.

Trading es un **sistema basado en datos y algoritmos que ejecuta decisiones de forma autónoma**

$ systemctl start trading . . .

Primitivas del Trading

Instrumentos

- Spot
- Futuros / Perpetuos
- Opciones (Call / Put)

Órdenes

- Market
- Limit
- Cancel
- Stop (derivado)

Una orden es solo un **mensaje** enviado a un exchange.

Ejemplos:

> BUY 1 BTC @ MARKET

> SELL 1 BTC @ LIMIT 50,000

El Libro de Órdenes (Order Book)

El order book representa todas las **órdenes de compra y venta** de un mercado.

Contiene:

- Precios
- Cantidades
- Prioridad temporal (L3)

Se actualiza:

- Miles de veces por segundo
- Con cada orden, cancelación o fill

Es la **fuente de verdad** del mercado en tiempo real.

Arquitectura de un Sistema de Trading

Pipeline de Ejecución





Feed

Entrada de eventos (candles, trades, order book)

Engine

Estrategias de Trading

Risk

Controles de seguridad

Gateway

Salida de órdenes de compra, venta, cambio o cancelación

Exchange

Cruce de órdenes

Cada componente tiene responsabilidades claras. El sistema reacciona a eventos.

```
$ while read event; do engine | risk | gateway; done < market-data
```

Tecnologías en Sistemas de Trading

Research

Python, Notebooks,

Backtesting

Producción

C++, Rust, Java, Go

Servicios orientados a eventos

Comunicación

REST, WebSocket, FIX, TCP, UDP,

ITCH, OUCH, SQF

Infraestructura

Docker, Kubernetes, Cloud

(AWS, Google)

Las tecnologías se eligen por **control, predictibilidad y fiabilidad**.

El Sistema Crece

Con el tiempo, el sistema escala: más instrumentos, estrategias, eventos y riesgo.

Los problemas dejan de ser solo de lógica. Aparecen requisitos de **escala, aislamiento y operación**.

Aquí entran **virtualización y cloud**.

Tecnologías de Virtualización en Trading

Compute

Máquinas Virtuales

Contenedores (Docker)

Orquestación

Kubernetes

Auto-scaling, Health checks

Red

VPCs, Subredes

Network policies, Load balancers

IaC

Terraform, Helm

Ansibe, Argo, CI/CD

La infraestructura se diseña igual que el código.

```
$ kubectl get pods --all-namespaces
```

Ejemplo: Plataforma de Trading Distribuida

Arquitectura

- Múltiples clusters en distintas regiones
- Cercanía física y lógica a los exchanges
- Comunicación entre clusters

Desafíos

- **Latencia:** externa (exchange) e interna (codecs, colas, scheduling)
- **Resiliencia:** Failover automático, caída de zonas o servicios

La topología impacta directamente en riesgo y ejecución.

Cada cluster ejecuta decisiones **locales**, coordinadas a nivel **global**.

Virtualización y Seguridad por Diseño

La seguridad se implementa aprovechando las primitivas de la virtualización.

La seguridad no se añade. Se diseña desde el principio.

- Contenedores por servicio
- Límites estrictos de CPU y memoria
- Sin ejecución como root (principio de mínimo privilegio)

- Filesystem de solo lectura
- Network policies (deny by default)
- Blast radius reducido (fallos y compromisos acotados)

```
$ kubectl exec trading-engine -- id
uid=1000 gid=1000
```

```
$ kubectl exec trading-engine -- curl https://google.com
curl: (7) Failed to connect
```

Dinero ⇒ Seguridad

Cuando el software mueve dinero,
la seguridad es parte del sistema.

- ⚠ Un **bug** puede costar dinero
- ↺ Un **fallo** puede ser irreversible
- ⚙ Un **ataque** puede ser sistémico

Aquí la seguridad no es opcional:
es **supervivencia**.

```
$ send_tokens --to=0xDEAD...BEEF --amount=100
```

Wallets 101

Una wallet no “guarda” dinero. Una wallet gestiona claves criptográficas que permiten controlar direcciones en una blockchain.

Clave privada → Clave pública → Address → Estado en la blockchain

La wallet define cómo se generan, almacenan y usan esas claves. No existe una wallet universal: existen wallets por ecosistema.

Wallets Virtuales

- MetaMask
- Phantom
- Rabby
- Trust Wallet

Wallets Hardware

- Ledger
- Trezor
- Cypherock

```
$ geth account new --keystore ./keystore
```

Address: {0x3a1B...F9c2}

Custodia Institucional

Cuando una organización gestiona millones en cripto, la custodia individual no escala.

La custodia institucional reduce riesgo operativo, humano y sistémico.

Tecnologías y Principios

Multi-Party Computation (**MPC**) → no existe una clave única

Hardware Security Modules (**HSMs**) → la clave no sale del HW

Políticas de firma → nadie puede mover fondos solo

Whitelists , Time-locks, SegRoles (trading ≠ firma ≠ custodia)

Plataformas

Fireblocks, Copper

Coinbase Custody

Anchorage Digital

La confianza se sustituye por **arquitectura y criptografía**.

Zero-Knowledge Proofs

En sistemas financieros tradicionales, la confianza se delega en terceros (bancos, auditores, reguladores).

En cripto no hay intermediarios confiables, todo es público por defecto, pero la privacidad sigue siendo necesaria

Un Zero-Knowledge Proof permite demostrar que una afirmación es cierta sin revelar los datos que la sustentan.

“Probar sin revelar”

Ejemplos:

- Proof of Reserves en exchanges
- Verificación de solvencia
- Cumplimiento sin revelar datos sensibles

Stack Tecnológico

zk-SNARKs → pruebas cortas y rápidas de verificar

zk-STARKs → sin trusted setup, más transparentes

Circom / Halo2 → definir y generar pruebas ZK

zkSync / StarkNet → ZK en producción sobre Ethereum

Q: eres mayor de edad? A: sí (sin revelar edad ni identidad)

Inteligencia Artificial como Apoyo

La IA **no está en el path crítico del dinero.**

- ML clásico, Deep Learning, LLMs
- Research, análisis, operaciones, observabilidad
- El core de ejecución es determinista

Predicción ≠ ejecución

Probabilidad ≠ responsabilidad

Ejemplos de Uso de IA

Research

Generación de señales,
Feature engineering,
Backtesting asistido

Sentimiento

NLP, LLMs, Feeds en tiempo real

Operaciones

Detección de anomalías,
Alerting inteligente,
Root-cause analysis

Reporting

Informes automáticos,
Resúmenes ejecutivos

La IA acelera el trabajo. El sistema sigue siendo responsable.

Mojo: Un Puente entre Research y Producción

Problema Habitual

- Research en Python (rápido)
- Producción en C++/Rust (rendimiento)

Solución: Mojo

- Superset de Python
- Rendimiento cercano a lenguajes de sistemas
- Control de memoria y acceso a hardware
- Ejecución eficiente en CPU y GPU
- Basado en MLIR para optimización de bajo nivel

SOFTMAX.🔥

```
def softmax(lst):
  norm = np.exp(lst - np.max(lst))
  return norm / norm.sum()

struct NDArray:
  def max(self) -> NDArray:
    return self.pmap(SIMD.max)

struct SIMD[type: DType, width: Int]:
  def max(self, rhs: Self) -> Self:
    return (self >= rhs).select(self, rhs)
```

Objetivo: reducir la fricción entre **prototipo** y **producción**.

$ EOL

Todo es ingeniería.

Software,

Hardware,

Matemáticas,

Estadística,

...

- El trading moderno es **software distribuido**.
- La virtualización **controla el fallo**.
- La seguridad **protege el dinero**.
- Cripto **elimina intermediarios**.
- La IA apoya, pero **no ejecuta**.

Q&A

Gracias por vuestra atención.

Francisco López Lasanta

https://gsr.io

https://www.linkedin.com/in/flopezlasanta

https://github.com/fran0x