# Arquitectura de Software

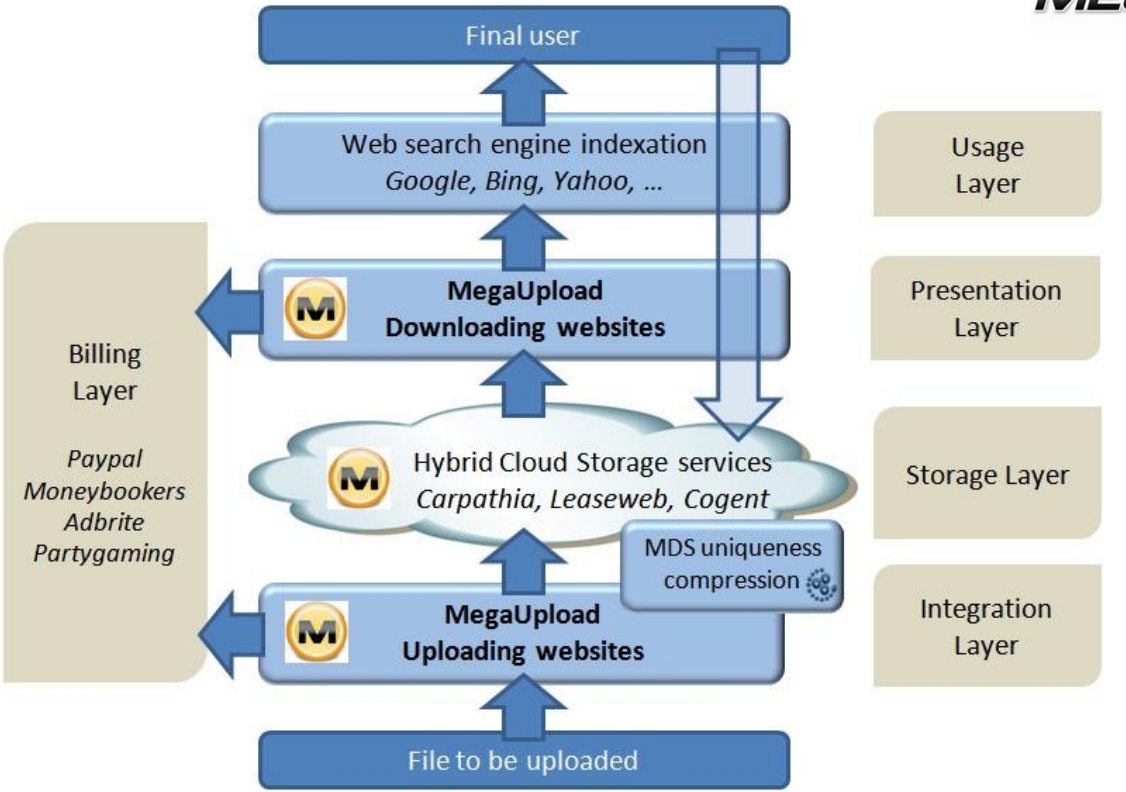
# Caso de estudio: File Sharing


## Cómo lo construirían?

# MegaUpload


- → 25 Petabytes de información (25 000 TB)
- → 13 ranking sitios más visitados
- → 4% Tráfico de internet

# MegaUpload


Final user
Web search engine indexation
Google, Bing, Yahoo, ...
MegaUpload
Downloading websites
Billing
Layer
Paypal
Moneybookers
Adbrite
Partygaming
Hybrid Cloud Storage services
Carpathia, Leaseweb, Cogent
MDS uniqueness
compression
MegaUpload
Uploading websites
File to be uploaded
Usage
Layer
Presentation
Layer
Storage Layer
Integration
Layer

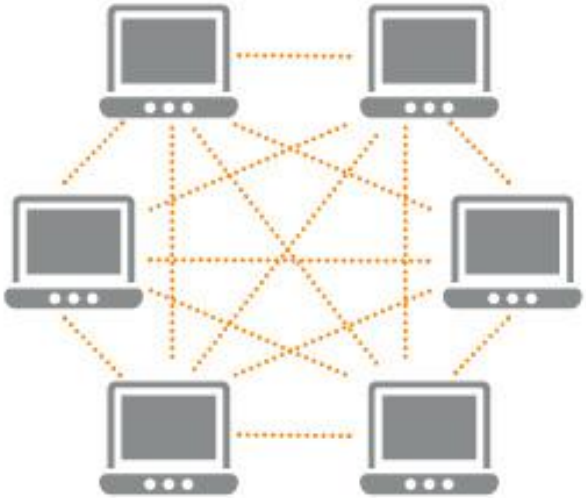
# MegaUpload

Server-Based

# Torrent

BitTorrent

µTorrent



P2P

# RIP MegaUpload

# Introducción

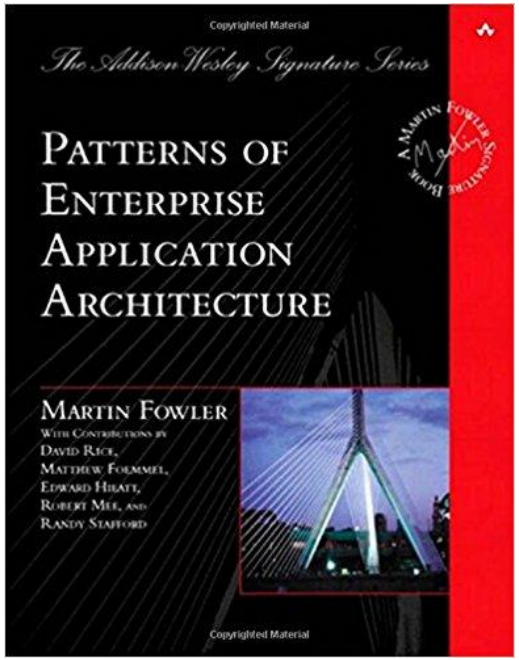
# Qué es la arquitectura de SW?


"the highest level concept of a system in its environment. The architecture of a software system (at a given point in time) is its organization or structure of significant components interacting through interfaces, those components being composed of successively smaller components and interfaces."

# Qué es la arquitectura de SW?

"There is no highest level concept of a system. Customers have a different concept than developers. Customers do not care at all about the structure of significant components. So, perhaps an architecture is the highest level concept that developers have of a system in its environment. Let's forget the developers who just understand their little piece. Architecture is the highest level concept of the expert developers. What makes a component significant? It is significant because the expert developers say so."

# Qué es la arquitectura de SW?




"Decisions that are hard to change"

# Atributos de calidad

## Funcionalidades

habilidad del sistema para hacer el trabajo para el que fue diseñado.

- Subir archivo
- Compartir archivo
- Puntuar archivo
- Comentar archivo

## Atributos

características no funcionales que se consideran deseables en un sistema.

- Concurrencia
- Mantenibilidad
- Escalabilidad
- Disponibilidad

# Atributos de calidad

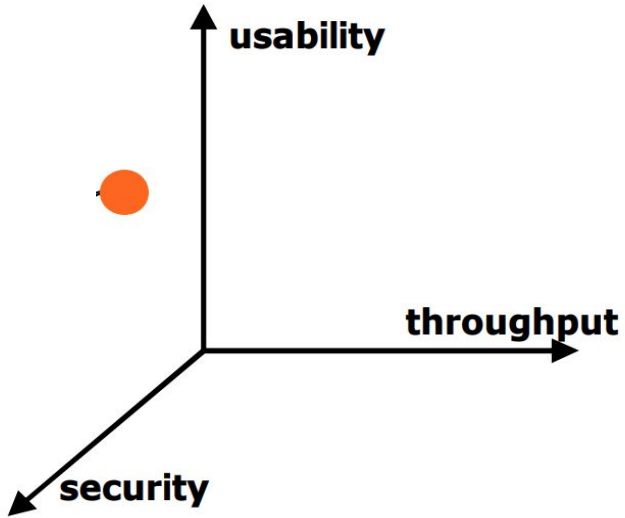
- accessibility
- accountability
- accuracy
- adaptability
- administrability
- affordability
- agility (see Common subsets below)
- auditability
- autonomy [Erl]
- availability
- compatibility
- composability [Erl]
- configurability
- correctness
- credibility
- customizability
- debuggability
- degradability
- determinability
- demonstrability
- dependability (see Common subsets below)
- deployability
- discoverability [Erl]
- distributability
- durability
- effectiveness
- efficiency
- evolvability
- extensibility
- failure transparency
- fault-tolerance
- fidelity
- flexibility
- inspectability
- installability
- integrity
- interchangeability
- interoperability [Erl]
- learnability
- localizability
- maintainability
- manageability
- mobility
- modifiability
- modularity
- observability
- operability
- orthogonality
- portability
- precision
- predictability
- process capabilities
- producibility
- provability
- recoverability
- relevance
- reliability
- repeatability
- reproducibility
- resilience
- responsiveness
- reusability [Erl]
- robustness
- safety
- scalability
- seamlessness
- self-sustainability
- serviceability (a.k.a. supportability)
- securability (see Common subsets below)
- simplicity
- stability
- standards compliance
- survivability
- sustainability
- tailorability
- testability
- timeliness
- traceability
- transparency
- ubiquity
- understandability
- upgradability
- usability
- vulnerability

# Atributos de calidad

- → Muchas veces no están escritos!
- → La seguridad del sistema está implícita?
- → El SIU se va a bancar la carga del sistema de inscripciones.

# Por qué prestarles atención?

Es necesario priorizar entre los diferentes atributos. Home banking: seguridad vs usabilidad.



# Por qué prestarles atención?

Tendencia en sistemas rediseñados no por falta funcional sino por deficiencias en performance.

# Por qué prestarles atención?


→ PHP: Lenguaje de scripting. Tuvo que ser compilado(HHVM)

→ MYSQL: Lento acceso. Información servida principalmente desde Memcached

# Ejercicio

Juntarse con su squad:

- Investigar rápidamente para Explicar a todos el atributo designado. dar 1 ejemplo

| Nro Squad | Atributo |
|---|---|
| 1,7 | Accesibilidad |
| 2,8 | Extensibilidad |
| 3,9 | Disponibilidad (availability) |
| 4,10 | Portabilidad |
| 5,11 | Escalabilidad |
| 6,12 | Resiliencia |

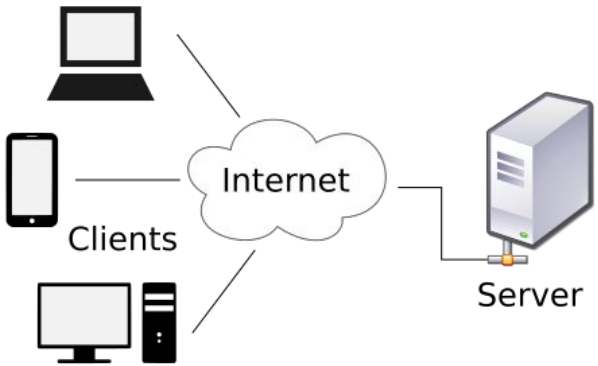
- Bonus track: pensar un segundo atributo que sea des-priorizado cuando priorizan el que les tocó (ejemplo: seguridad vs usabilidad)

Tiempo: 15 minutos.

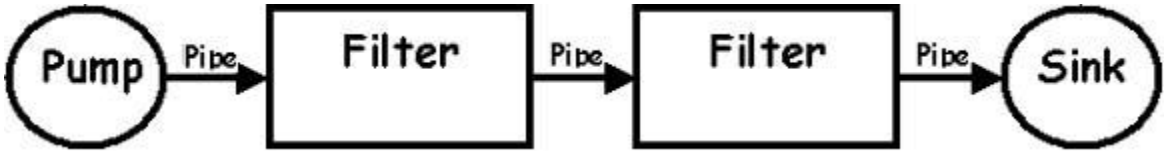
# Patrones y Estilos de arquitecturas

# Cliente - Servidor

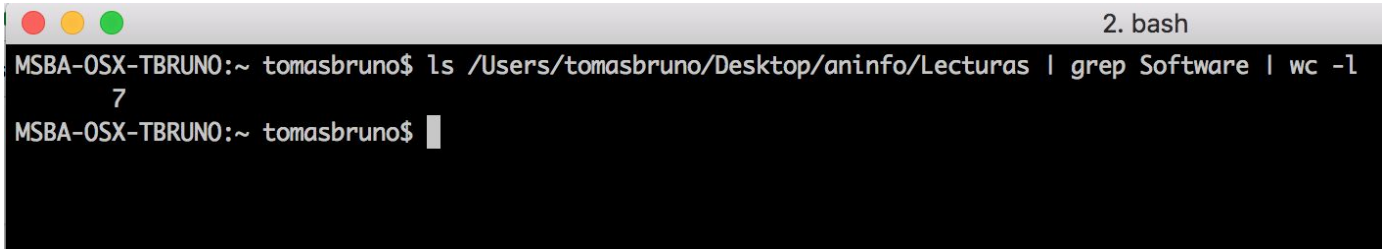
- → Cliente inicia la comunicación
- → Modelo request/response
- → Utilizan un lenguaje común (protocolo)



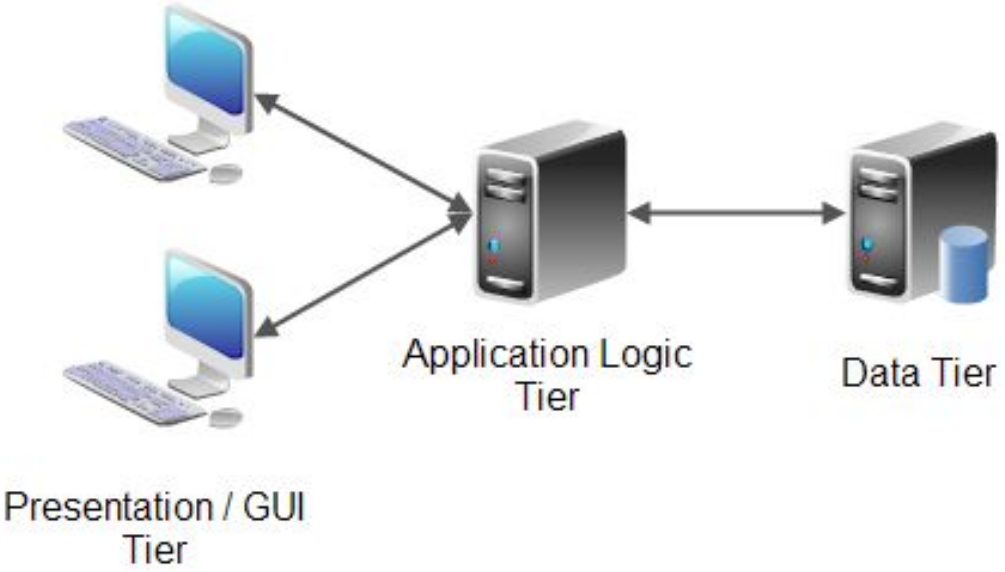
# Pipe and Filter



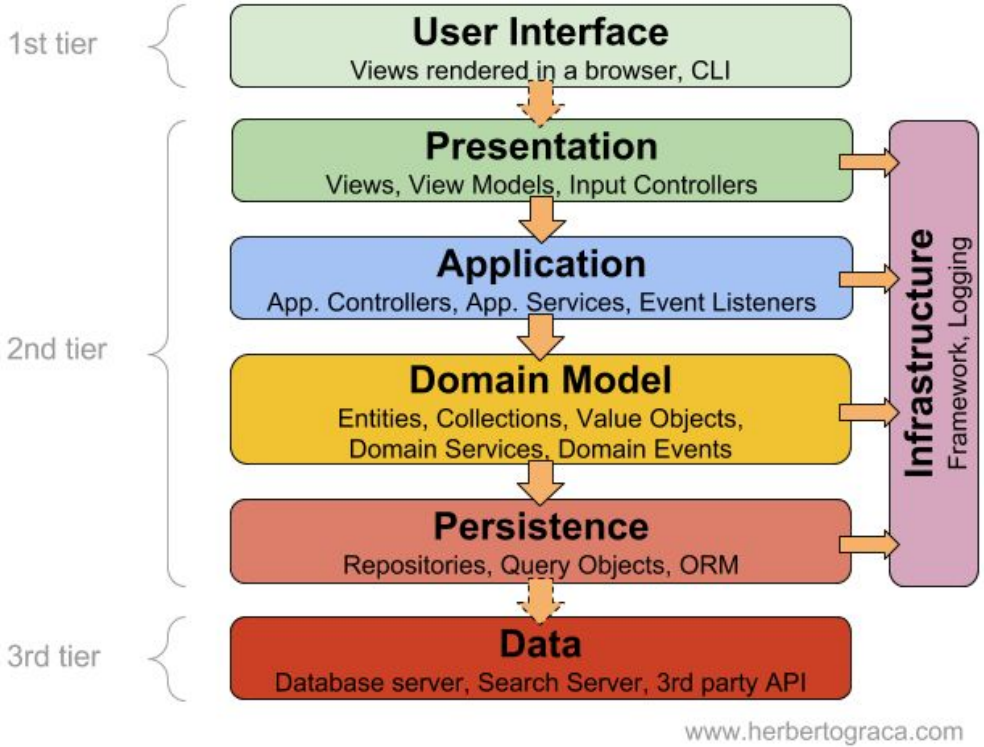

- Componentes conectados a través de conectores que procesan data.
- Ejemplo: comandos unix.



# N-tier: Separación física



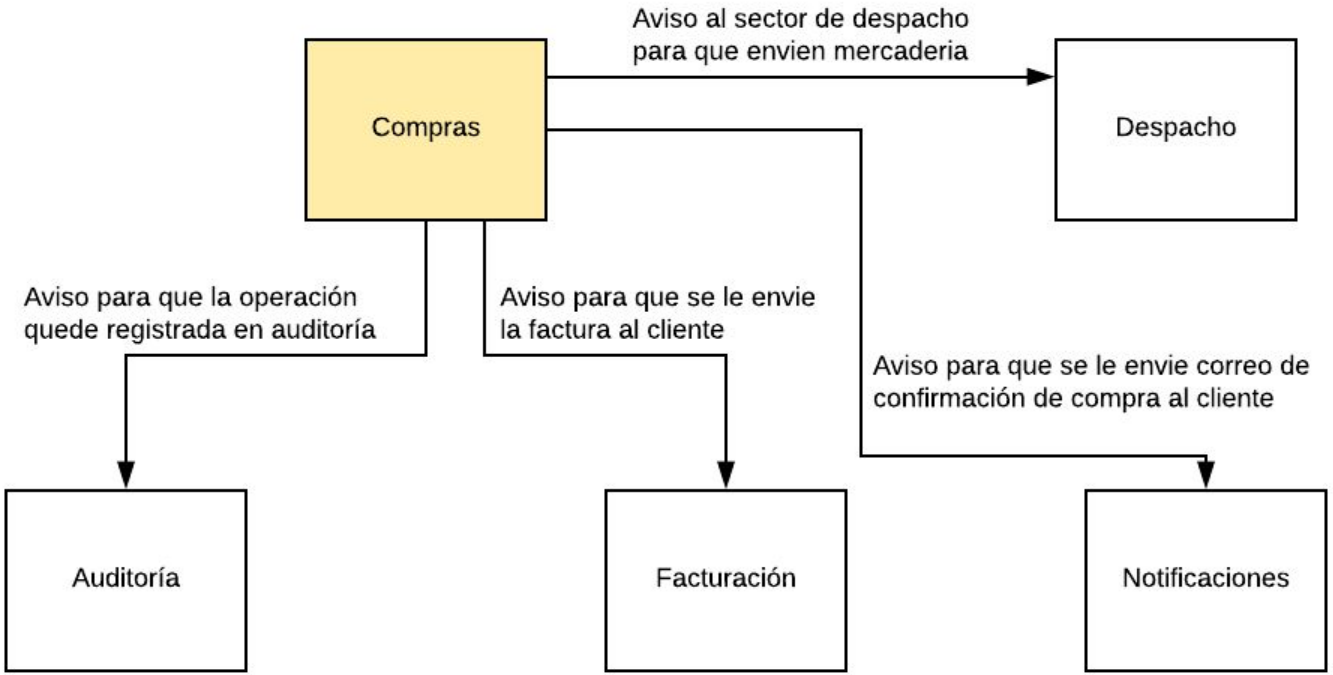
# Capas: Organización Lógica



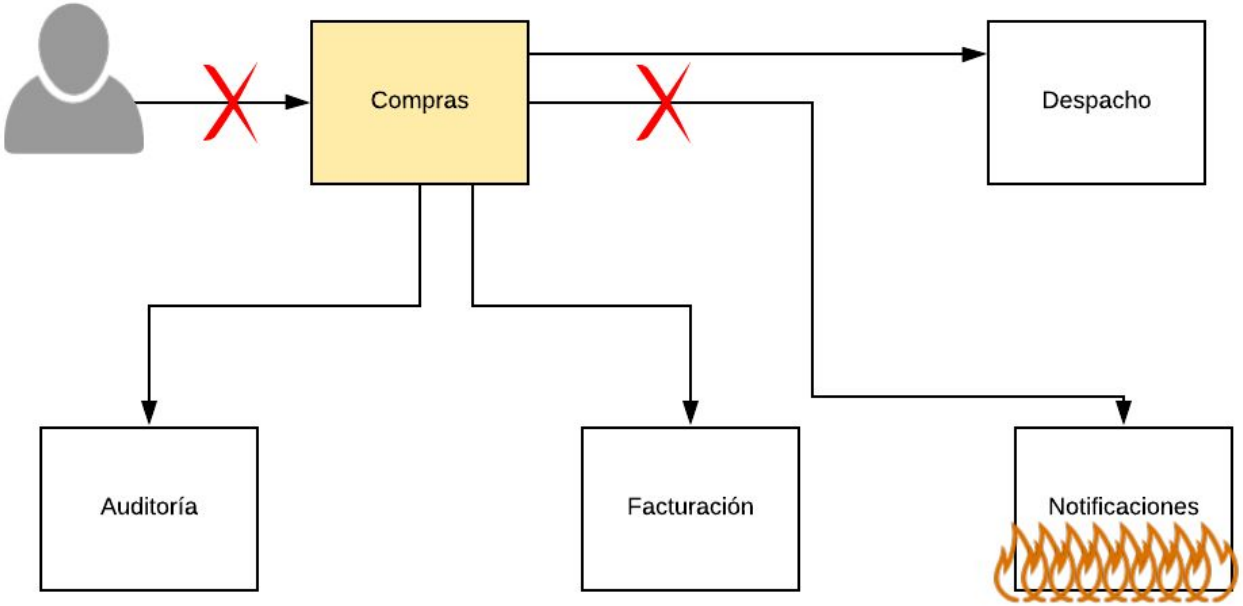
# Orientada a eventos

- Arquitecturas basadas en en la producción, consumición, detección y reacción a eventos.
- Un evento es un cambio en el estado
- Existencia de productores y subscriptores

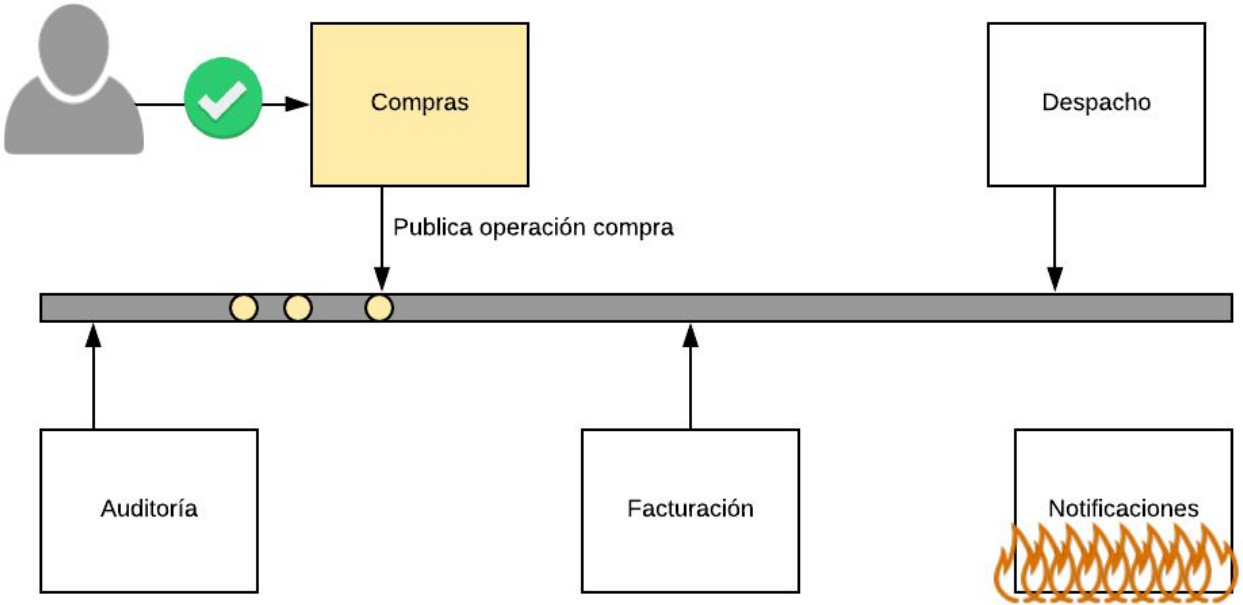
# ~~Orientada a eventos~~ P2P (Punto a punto)



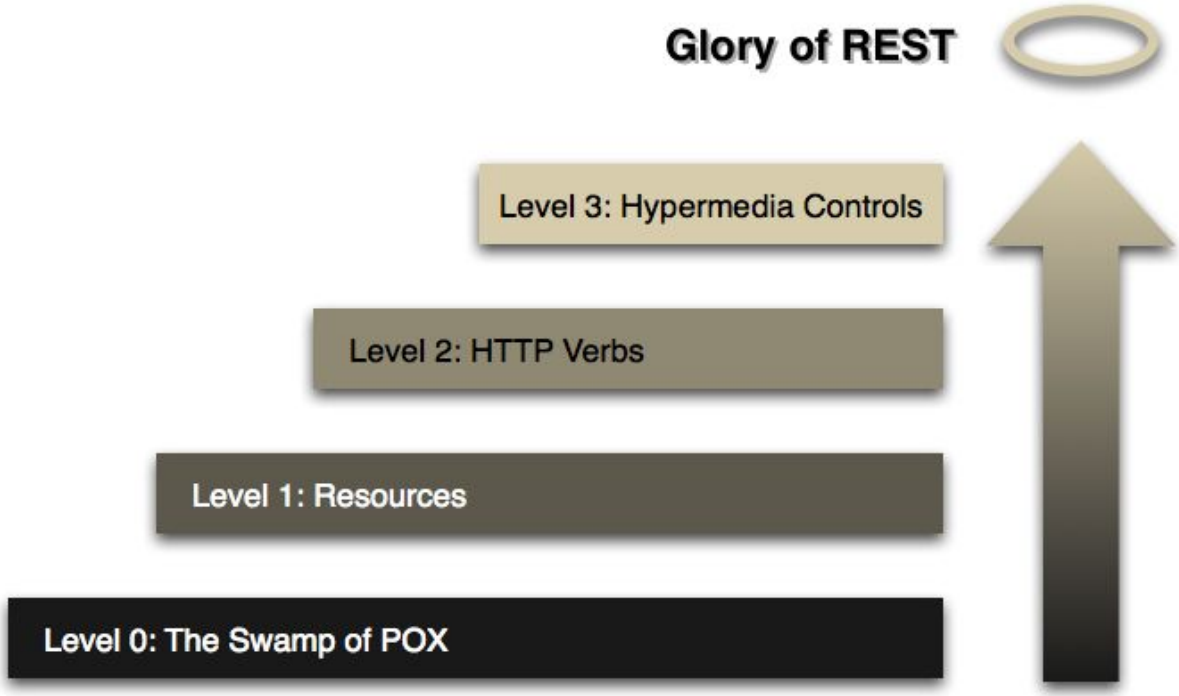
# ~~Orientada a eventos~~ P2P (Punto a punto)



# Orientada a eventos



# REST: Representational State Transfer



# REST - Level 0 - SWAMP of POX (plain old xml)

- → HTTP como mecanismo de transporte
- → HTTP/1.1 protocolo basado en texto (no binario) 
- → Desorden en el uso de los servicios expuestos:
  - Sin recursos definidos
  - Sin correcto uso de verbos 
- → Reglas básicas: Sin extensiones, '_', lowercase

# REST - Level 1 - Recursos

- → Usar diferentes URLS para interactuar con los diferentes recursos de nuestra API.
- → Los recursos de nuestra API suelen corresponde con las entidades de nuestro modelo.
- → Uso de verbos en la uri considerado como mala práctica.
  - Ej: GET http://aninfo.com.ar/**crear**Usuario

# REST - Level 2 - Verbos como sugiere HTTP

→ Seguro: NO modifican recursos

→ Idempotente: puede ser llamado una o 10 veces con el mismo resultado

| Verbo | Idempotente | Seguro |
|---|---|---|
| GET | ✓ | ✓ |
| POST | ✗ | ✗ |
| PUT | ✓ | ✗ |
| DELETE | ✓ | ✗ |

# REST - Level 2 - Verbos - Uso

| **Verbo** | **Acción** |
| --- | --- |
| GET | Recuperar información de un recurso |
| POST | Crear un recurso |
| PATCH | Actualización parcial de un recurso |
| PUT | Reemplazar la información de un recurso |
| DELETE | Eliminación de un recurso |

# REST - Level 2 (ext) - Códigos de respuesta

| Codigo | Significado | Ejemplo |
|---|---|---|
| 2xx | Exito | 201: Creado. |
| 3xx | Redirect | 301: Movido permanentemente. |
| 4xx | Error de cliente | 429: Demasiados requests. |
| 5xx | Error de servidor | 503: Servicio no disponible. |

# REST - Ejemplo API Restful

| **Acción** | **Ejemplo** |
| --- | --- |
| Crear usuario | POST .../usuarios |
| Recuperar un usuario | GET .../usuarios/:usuarioId |
| Recuperar todos los usuarios | GET .../usuarios |
| Reemplazar un usuario | PUT .../usuarios/:usuarioId |
| Actualización parcial de usuario | PATCH .../usuarios/:usuarioId |
| Eliminar un usuario | DELETE .../usuarios/:usuarioId |
| Cargar nota de usuario | POST .../usuarios/:usuarioId/notas |

# Microservicios

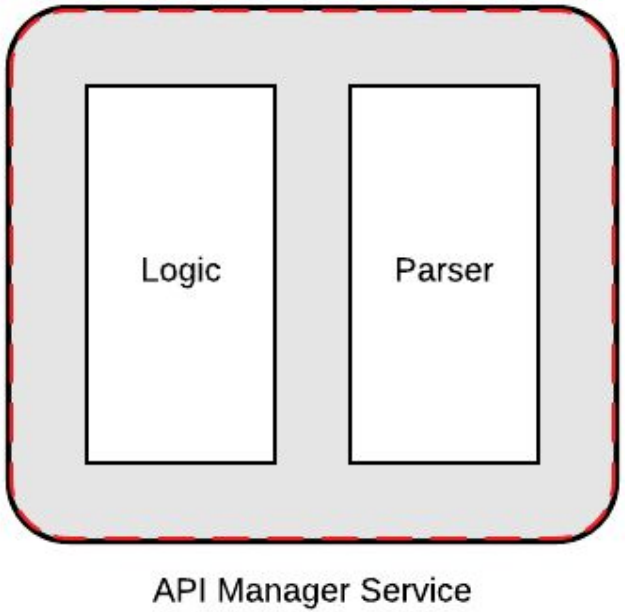
*“A distributed system is one in which the failure of a computer you didn’t even know existed can render you own computer unusable.” Leslie Lamport*

# Microservicios

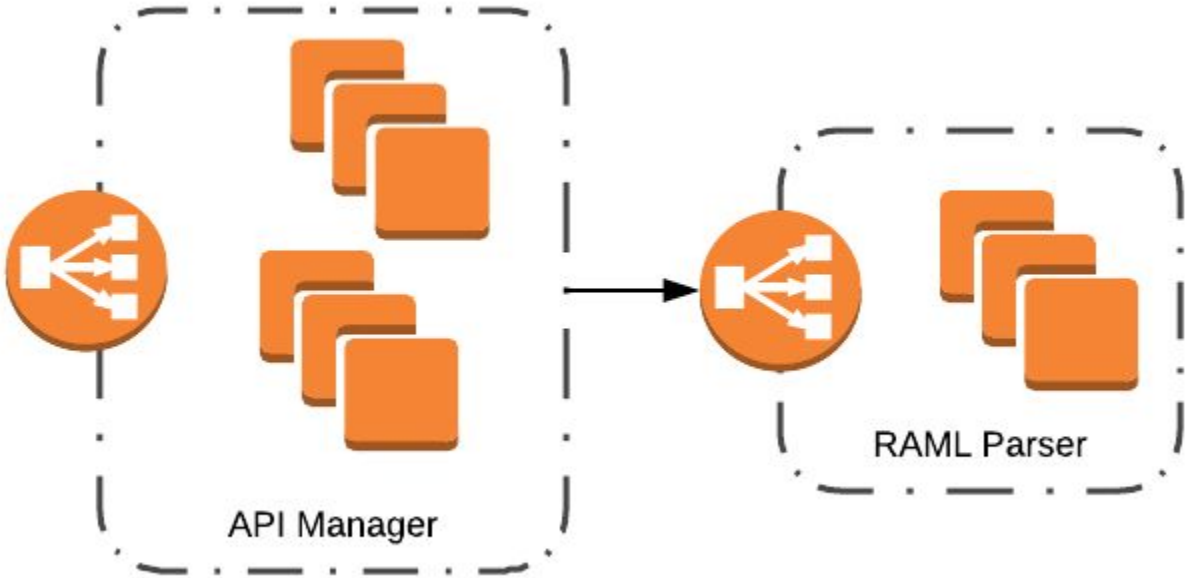
- → Componentización vía servicios (No bibliotecas).
- → Descomposición según negocio (no según función).
- → Producto (no proyecto). "You build it, you run it"
- → Gobierno descentralizado (y sin única tecnología estandarizada)
- → Design for failure
- → Automatización de la infraestructura
- → Diseño evolutivo

# Microservicios: Ejemplo

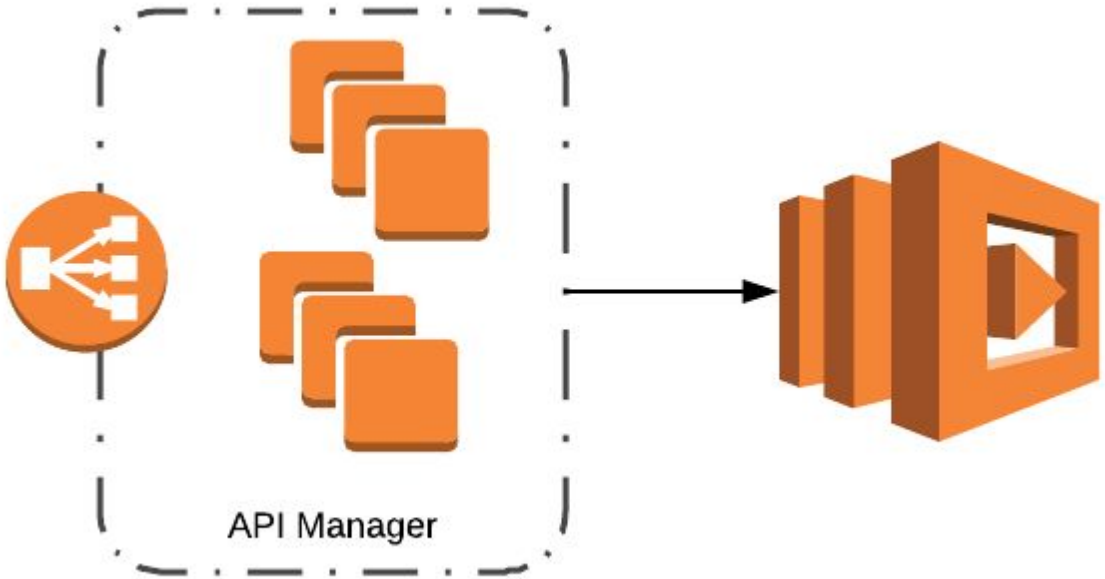
- → Servicio desarrollado en nodeJS
- → Parser embebido como dependencia del servicio
- → Parseo operación intensiva CPU
- → Parseo operación sincrónica



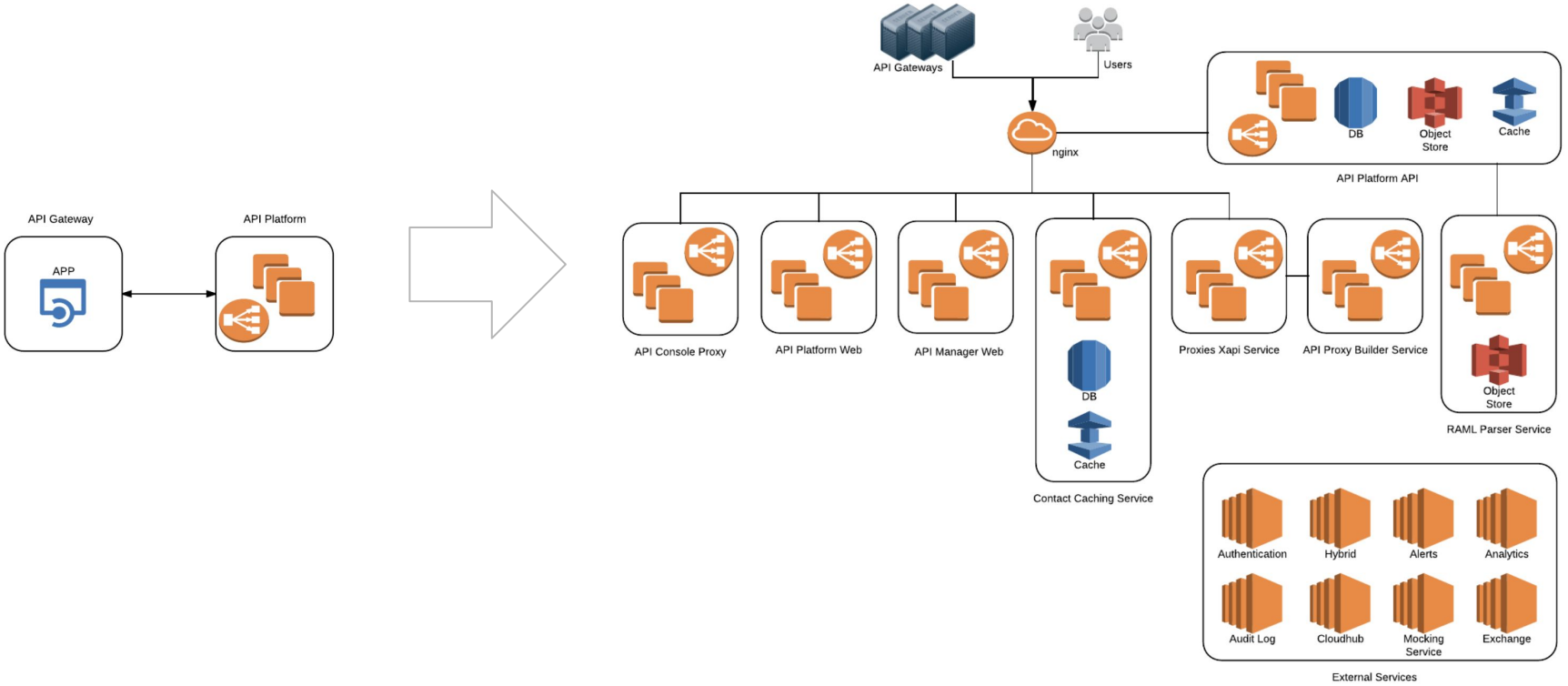
# Microservicios: Descomposición



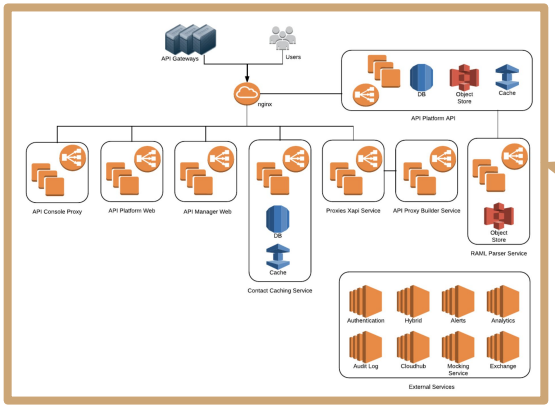
# Microservicios: Descomposición a FaaS

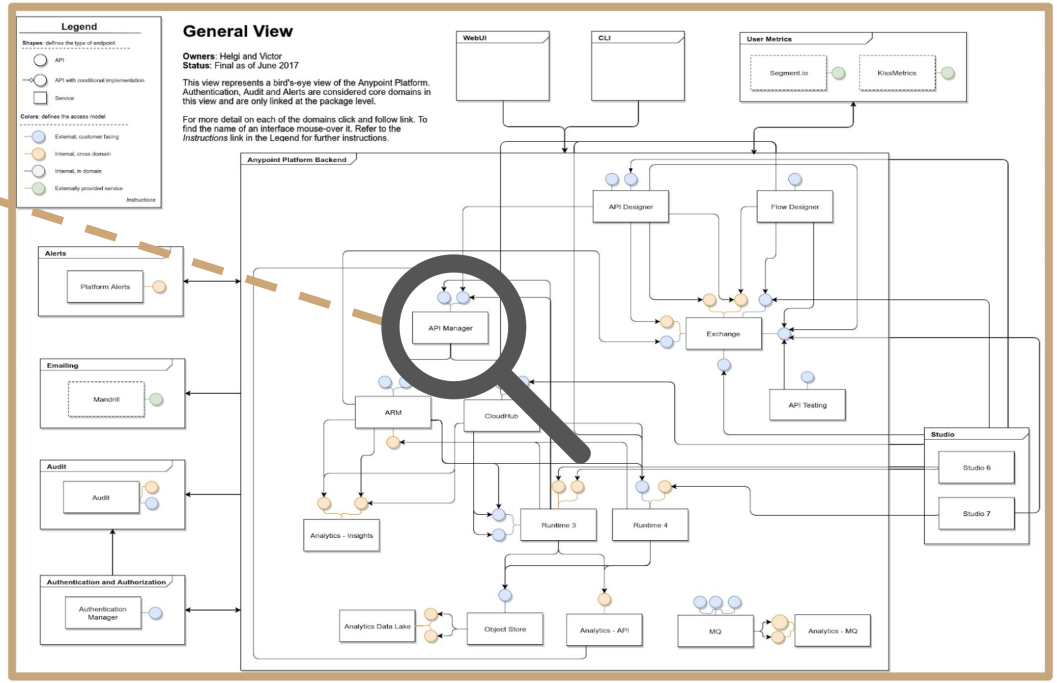


# Microservicios: Diseño evolutivo

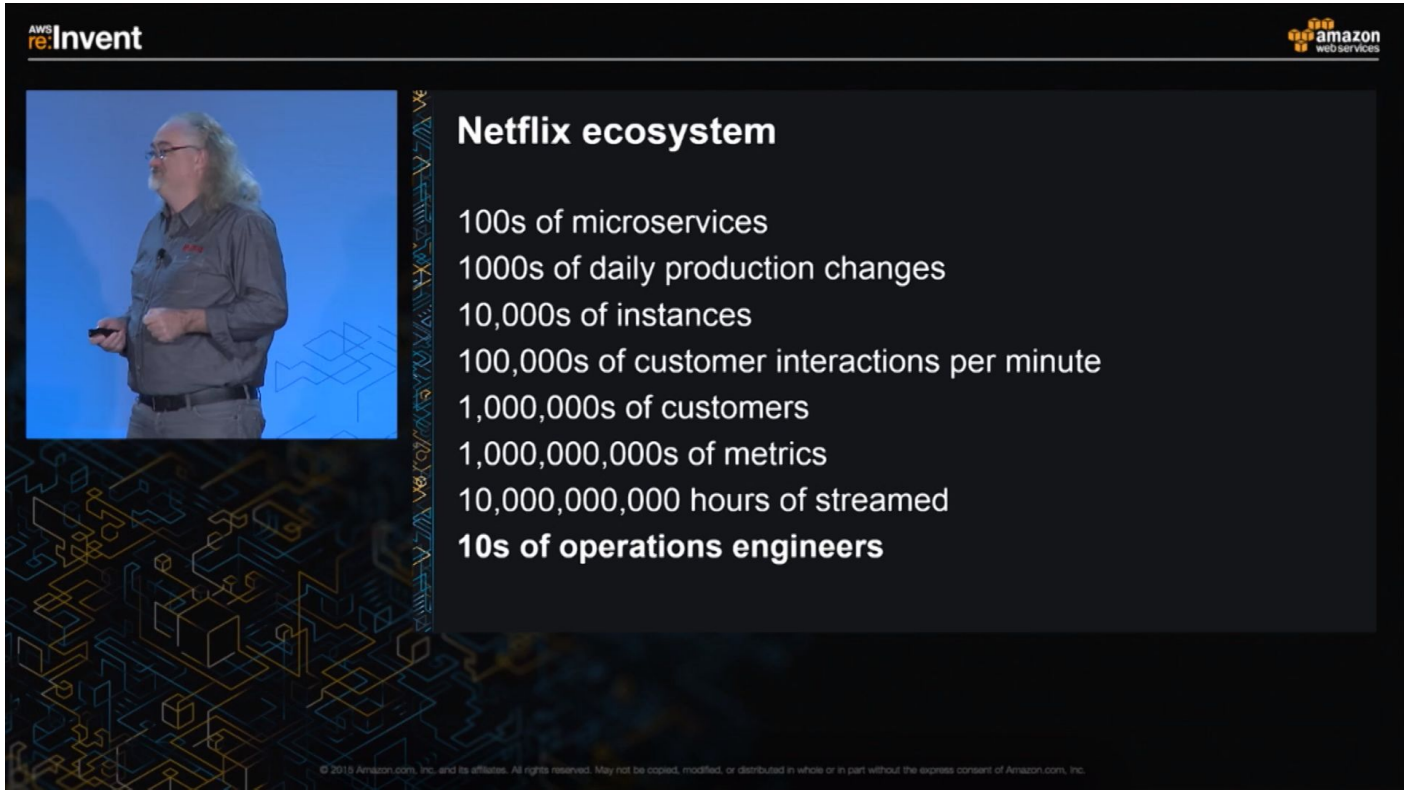


# Microservicios: Producto parte de una plataforma



# Microservicios: Netflix

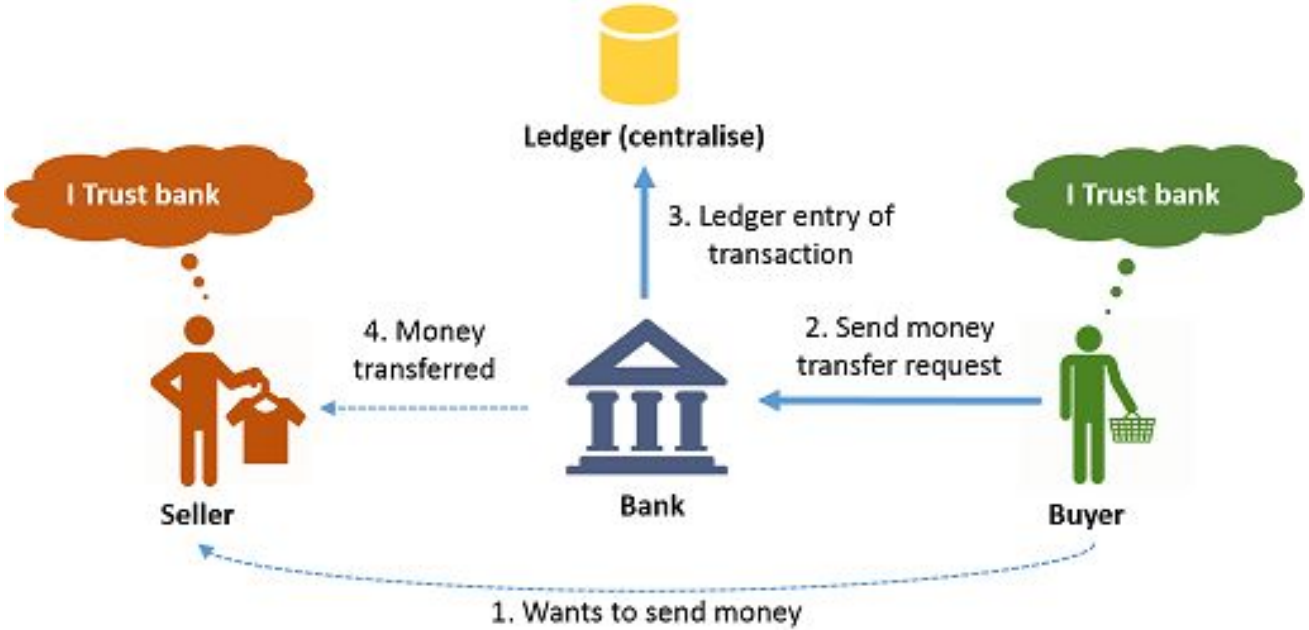


# Microservicios: Design for failure

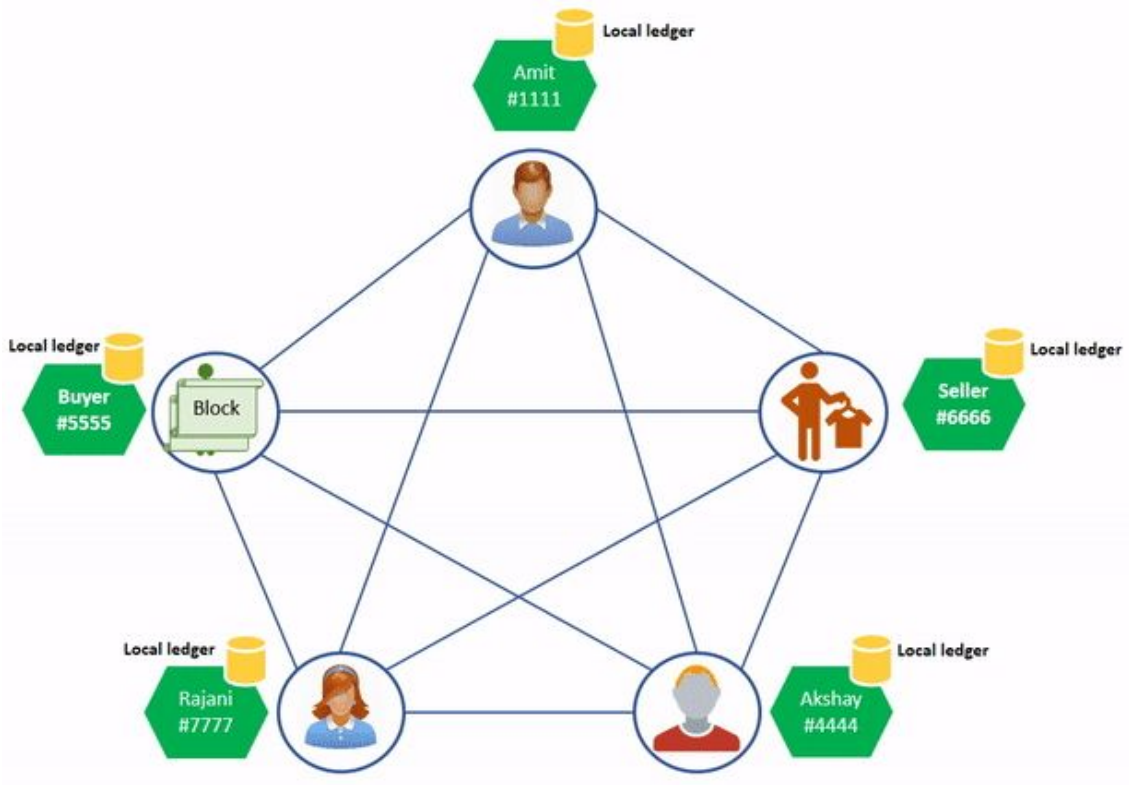
# Blockchain

*“Whereas most technologies tend to automate workers on the periphery doing menial tasks, blockchains automate away the center. Instead of putting the taxi driver out of a job, blockchain puts Uber out of a job and lets the taxi drivers work with the customer directly.”*
*Vitalik Buterin*

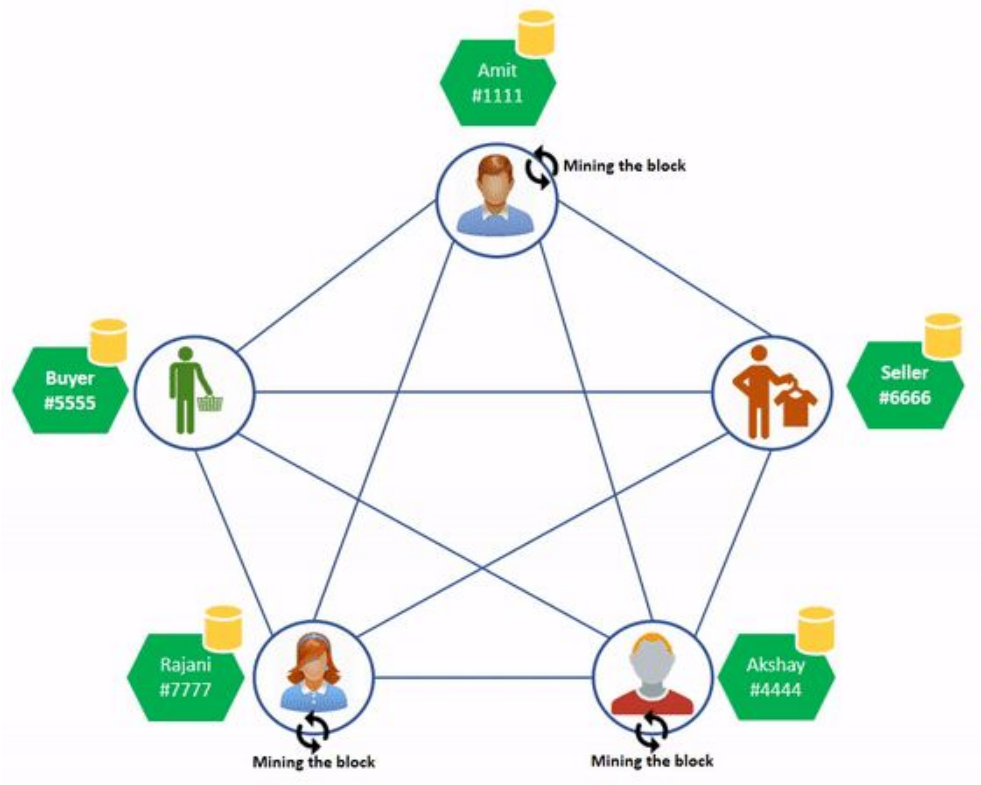
# Blockchain: Confianza?



# Blockchain: Confianza? No

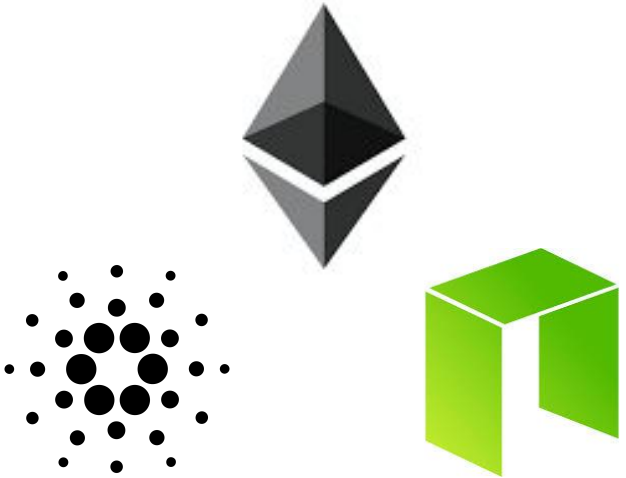


# Blockchain: Mineros



# Blockchain: tipos

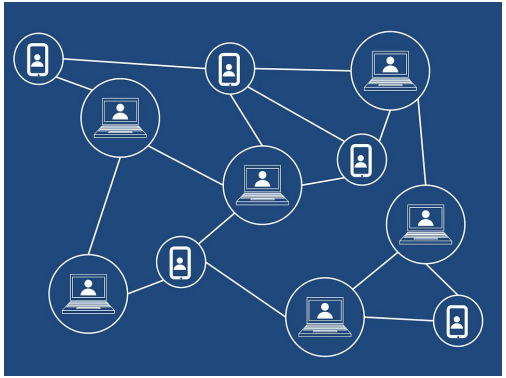
Monedas



Plataformas de aplicaciones

# Blockchain: Caso de uso - Información académica

Seguridad



Protección

# Architectural Kata

*“How do we get great designers? Great designers design,  of course." -- Fred Brooks*

*"So how are we supposed to get great architects, if  they only get the chance  to architect fewer than a half-dozen times in their career?"  --Ted Neward*

# All Staff, no cruft

Conference organizer needs a management system for the conferences he runs.

**Requirements**: attendees can access speaking schedule online, including room assignments; speakers can manage talks (enter, edit, modify); attendees "vote up/down" talks; organizer can notify attendees of schedule changes up-to-the-minute (if attendees opt in); each conference (being a different subject) can be branded independently; speaker slides are accessible online only to attendees; evaluation system via web page, email, SMS, or phone

**Users**: hundreds of speakers, dozens of event staff, thousands of attendees

# Bibliografía

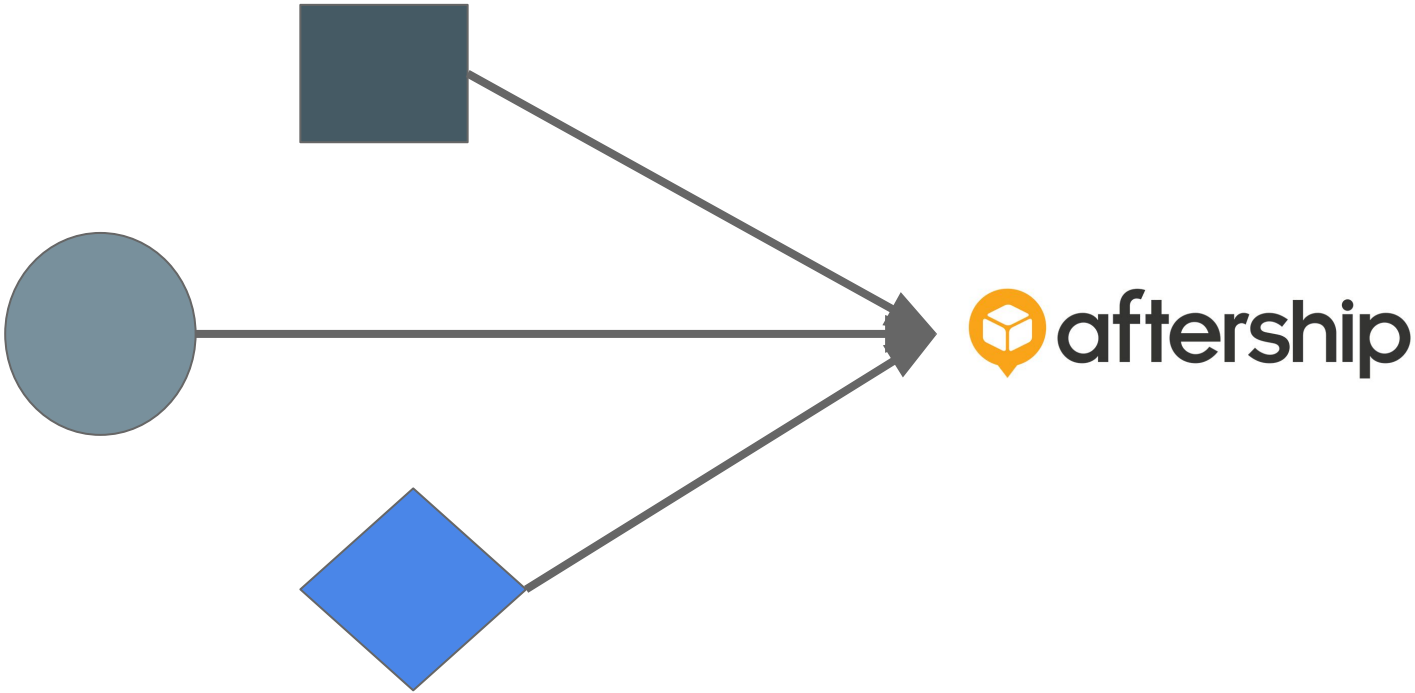

- “Patterns of Enterprise Architecture”. Martin Fowler
- “Introduction to Software Architecture”, George Fairbanks, http://www.georgefairbanks.com/assets/pdf/Intro-to-SA-CUBoulder-2012-09-18.pdf
- “Who needs an Architect?” Martin Fowler
- “Microservices Guide”. Martin Fowler. https://martinfowler.com/articles/microservices.html.
- Architectural Katas. https://archkatas.herokuapp.com/kata.html
- Starbucks does not use Two Phase Commit: https://www.enterpriseintegrationpatterns.com/ramblings/18_starbucks.html

# Ejemplo: Tracking as a Service

Cuál es el estado de tracking del paquete?

aftership

# Ejemplo: Implementando monetización



# Ejemplo: Alternativas

- → Código desarrollado dentro del servicio.
- → Código extraído a un módulo.
- → Utilización de un gateway para el chequeo del pago