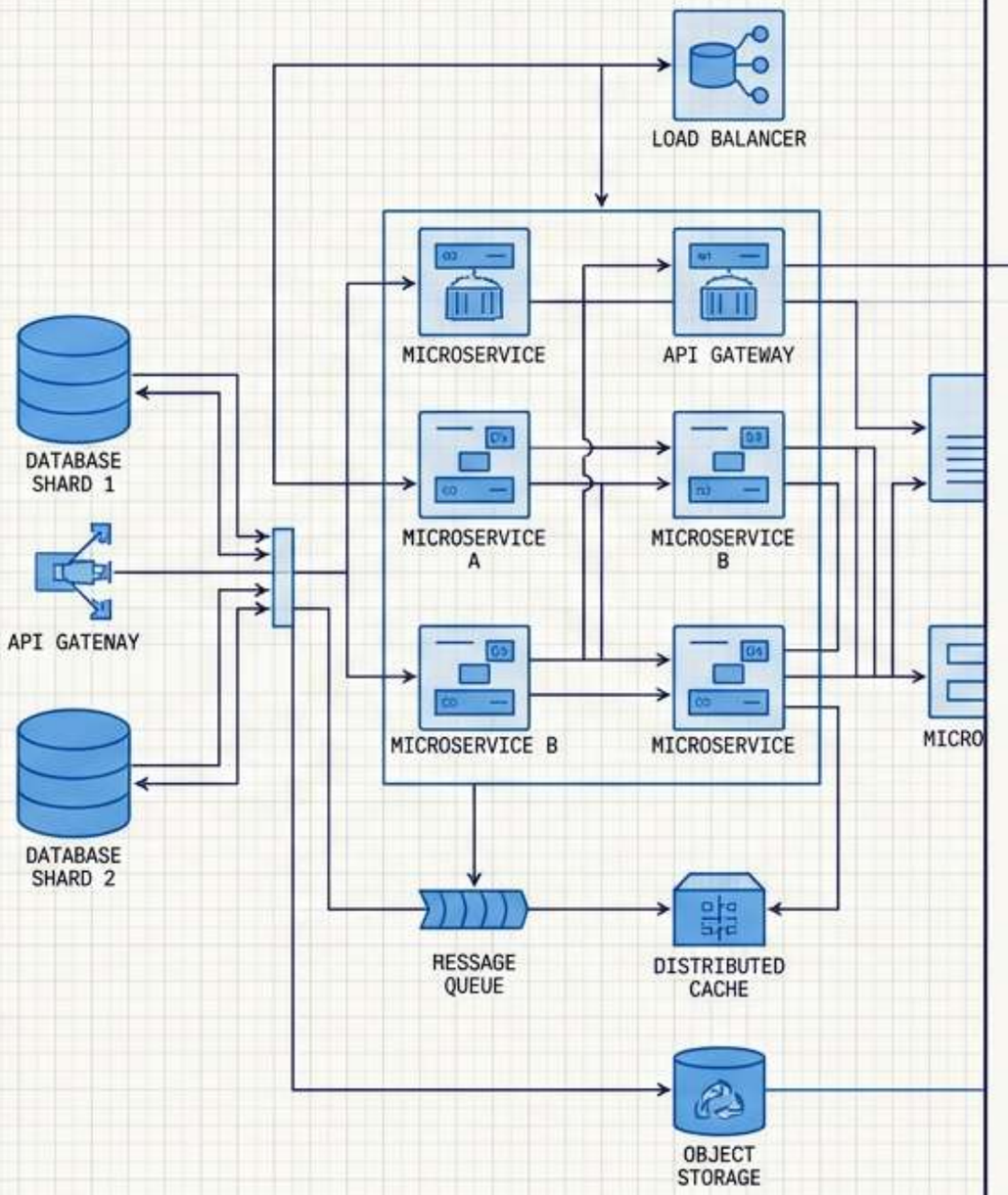
ARCHITEKTÚRA MODERNÝCH CLOUDOVÝCH SYSTÉMOV

Od monolitov k mikroslužbám



Syntéza evolúcie, dizajnových vzorov a prevádzkovej komplexnosti pre softvérových inžinierov a architektov.

Katalyzátor zmeny: Kríza škálovania (2005)

Prípadová štúdia: Yahoo & Amazon

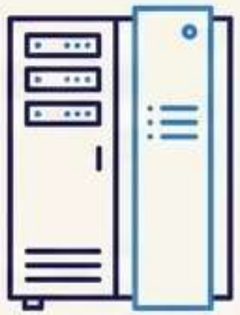
Pravidlo oneskorenia:
Každých 100 ms navyše znamená merateľný pokles výnosov.

Kríza spoločných zdrojov

Zariadenia boli silnejšie, ale limitom sa stala schopnosť jedného bodu zvládnuť paralelný nápor obrovskej masy používateľov.

Evolúcia IT architektúry

1960s



Mainframy

Centralizované

Veľké sálové počítače. Predchodcovia dnešnej virtualizácie a cloudu.

1980s

Client-Server

Decentralizované

Osobné počítače a Unix pracovné stanice. Dáta sa presúvajú na desktopy.

1990s

3-Tier Web

Tenký klient

Klient, aplikačný server a databáza. Neschopné masívneho horizontálneho škálovania.

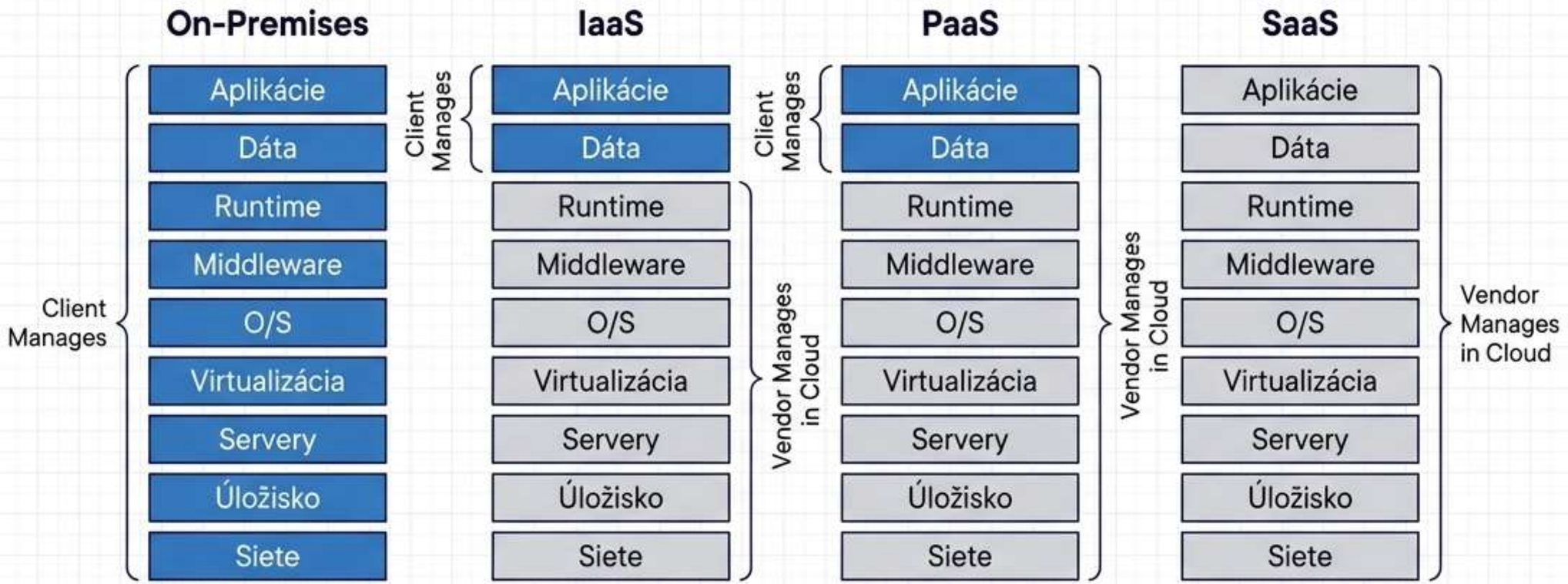
2000s+

Cloud & XaaS

Distribuované

Horizontálne škálovanie, IT továrne, internet služieb (SOA/REST).

Modely cloudových služieb (The XaaS Stack)



IaaS (Infrastructure)

Prenájom surového hardvéru. Najvyššia flexibilita.

PaaS (Platform)

Poskytovateľ spravuje runtime a škálovanie; vývojár iba nasadzuje kód.

SaaS (Software)

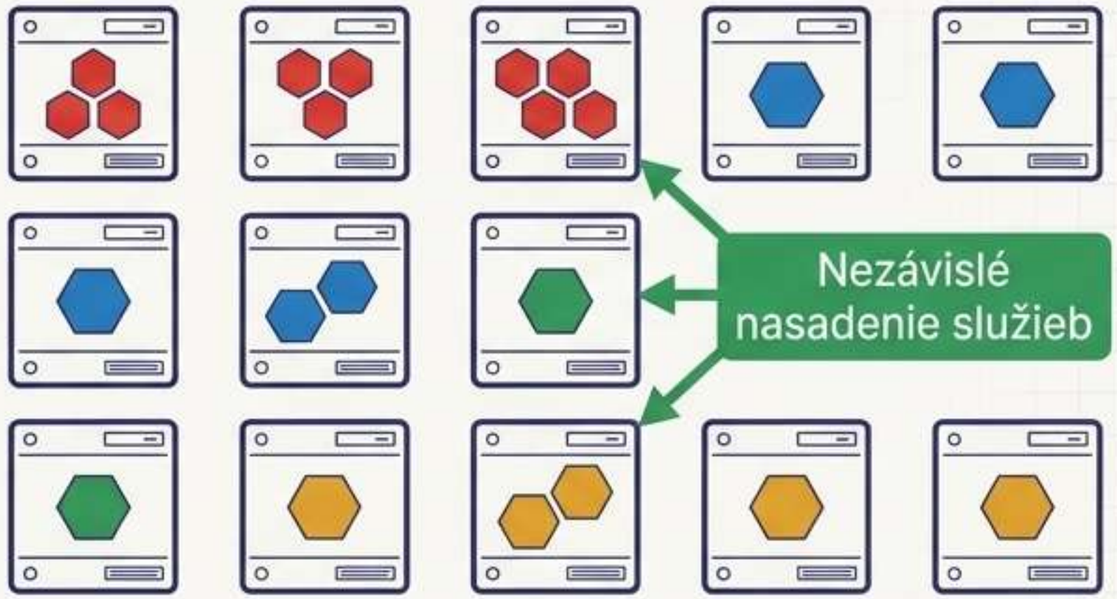
Softvér ako hotová platforma, používateľovi stačí len prehliadač.

Zmena paradigmy: Od monolitov k mikroslužbám

Monolitický prístup

- **Štruktúra:** Všetka funkcionalita je pevne zviazaná v jednom procese (hrubozrnná hustota).
- **Nasadenie:** Pri každej zmene je nutné nasadiť celú obrovskú aplikáciu znova.
- **Škálovanie:** Klonovanie celej aplikácie na nové servery (VM). Neefektívne pre úzke hrdlá.

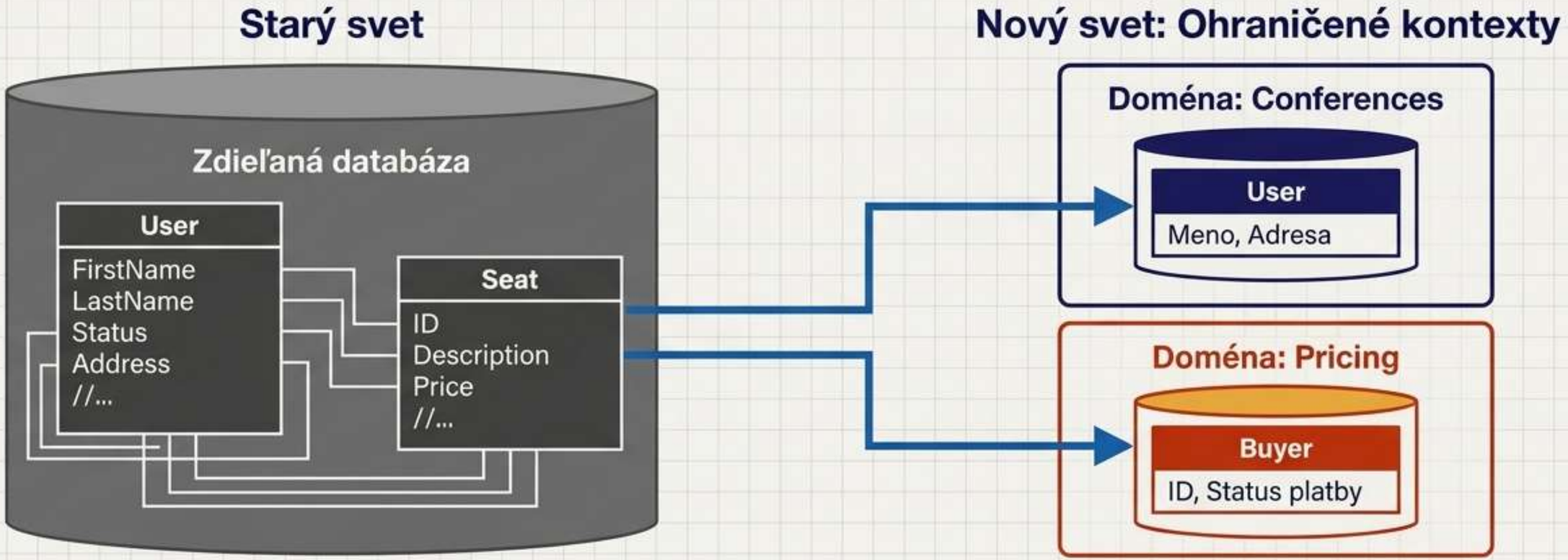
Architektúra mikroslužieb



- **Štruktúra:** Funkcionalita je segregovaná do malých, nezávislých služieb (jemnozrnná hustota).
- **Nasadenie:** Každá služba je nasadzovaná a vyvíjaná úplne nezávisle (ako Docker kontajner).
- **Škálovanie:** Nezávislé horizontálne škálovanie iba pre tie služby, ktoré to práve potrebujú.

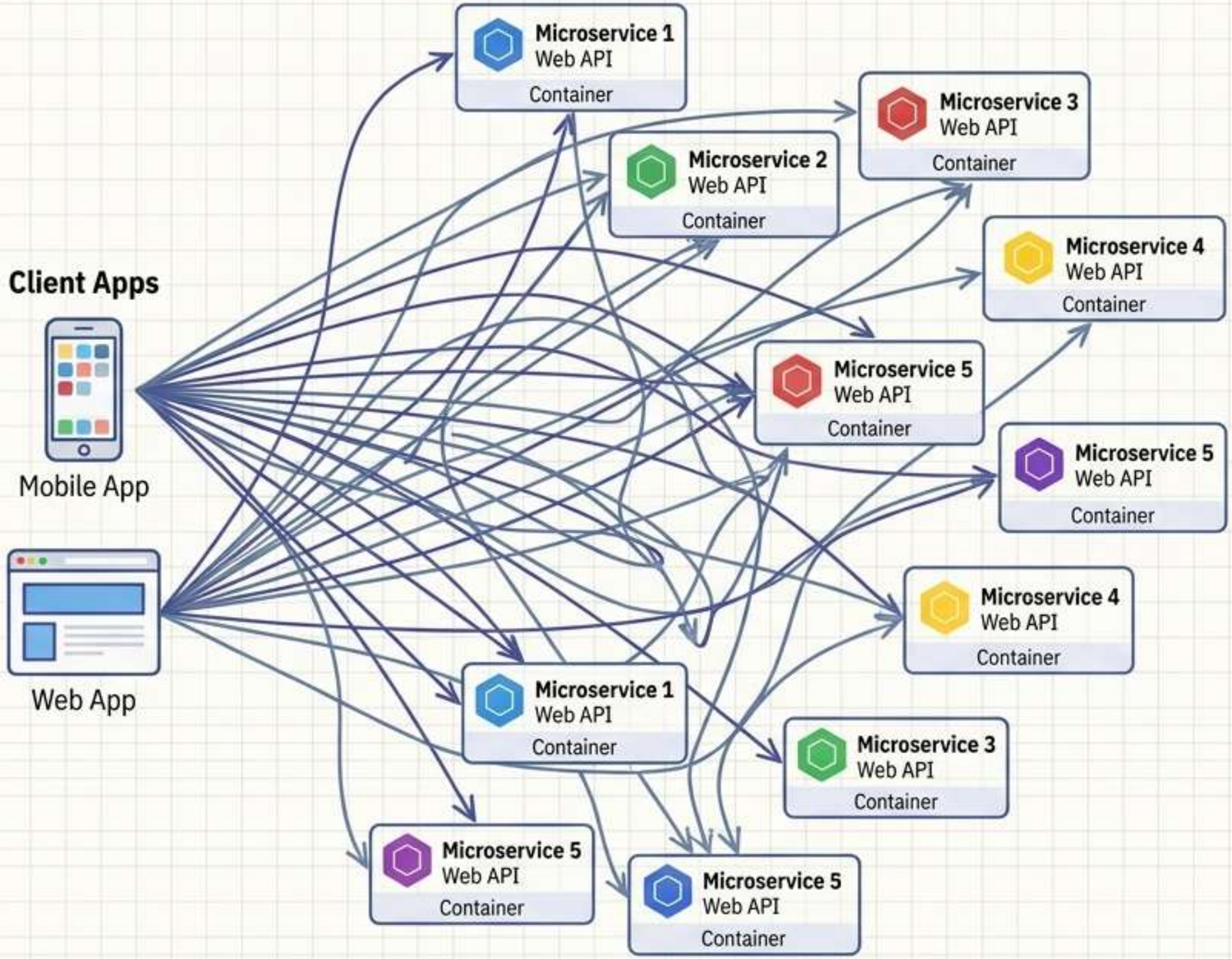
Dekompozícia dát (Domain-Driven Design)

Každá mikroslužba musí exkluzívne vlastniť svoj **dátový model** a databázu.



Tento prístup eliminuje zdieľané databázové zámky a umožňuje nezávislé technologické voľby.

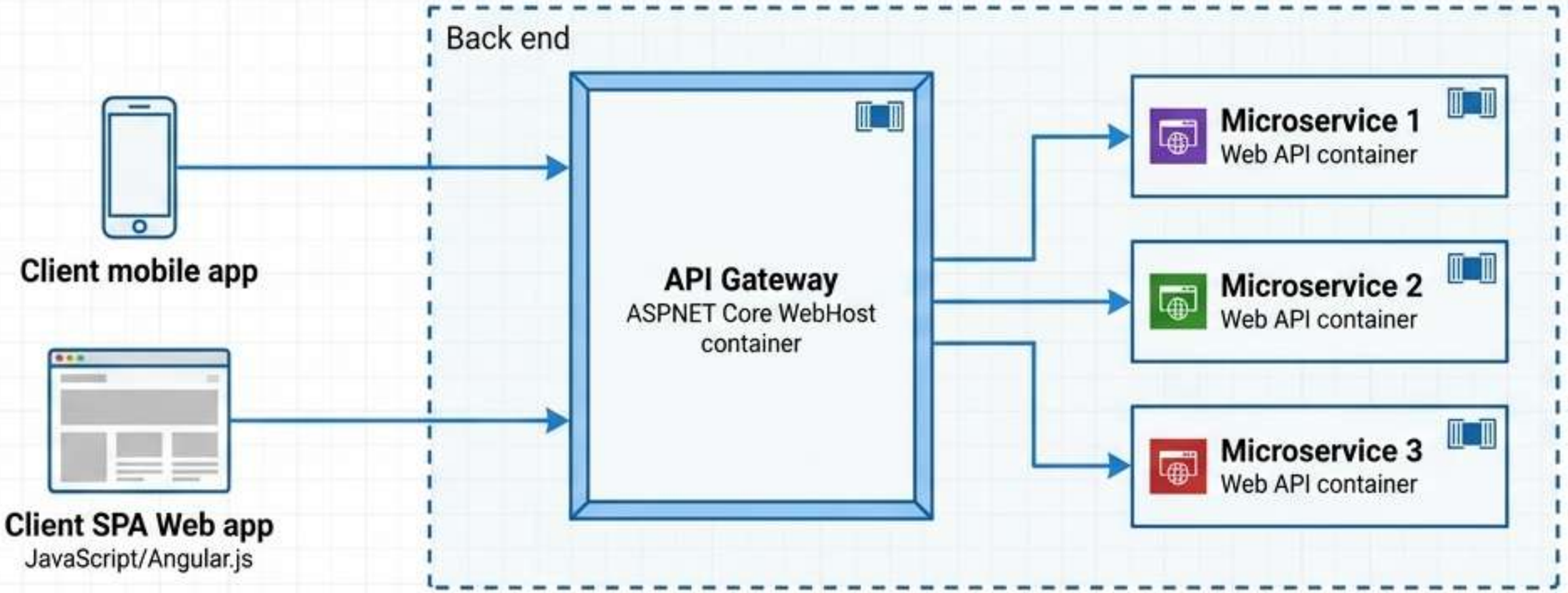
Architektonický chaos: Problém priamej komunikácie



Zlyhania tohto prístupu:

1. **Príliš veľa sieťových volaní:** Jedna obrazovka mobilnej aplikácie si vyžaduje 50 dopytov. Odozva drasticky klesá.
2. **Pevná väzba (Coupling):** Klient musí poznať vnútornú mapu siete a adresy všetkých mikroslužieb.
3. **Bezpečnostné riziko:** Každá mikroslužba je vystavená verejnému internetu – obrovská útočná plocha.
4. **Prierezové problémy:** Každá služba si musí sama riešiť SSL, autorizáciu a šifrovanie.

Riešenie agregácie: Vzor API Gateway



1. Reverzný proxy server

Tvorí jednotný vstupný bod (URL) pre klientov a maskuje vnútornú infraštruktúru.

2. Agregátor požiadaviek

Spojí dáta z viacerých vnútorných služieb a vráti ich klientovi v jednom JSON balíčku (znižuje zahltenie siete).

3. Bezpečnostný firewall

Centrálne rieši SSL certifikáty, autorizáciu, throttling a load balancing, čím odľahčuje mikroslužby.

Vnútrofiremná pošta:
Ako si služby **vymieňajú dáta**

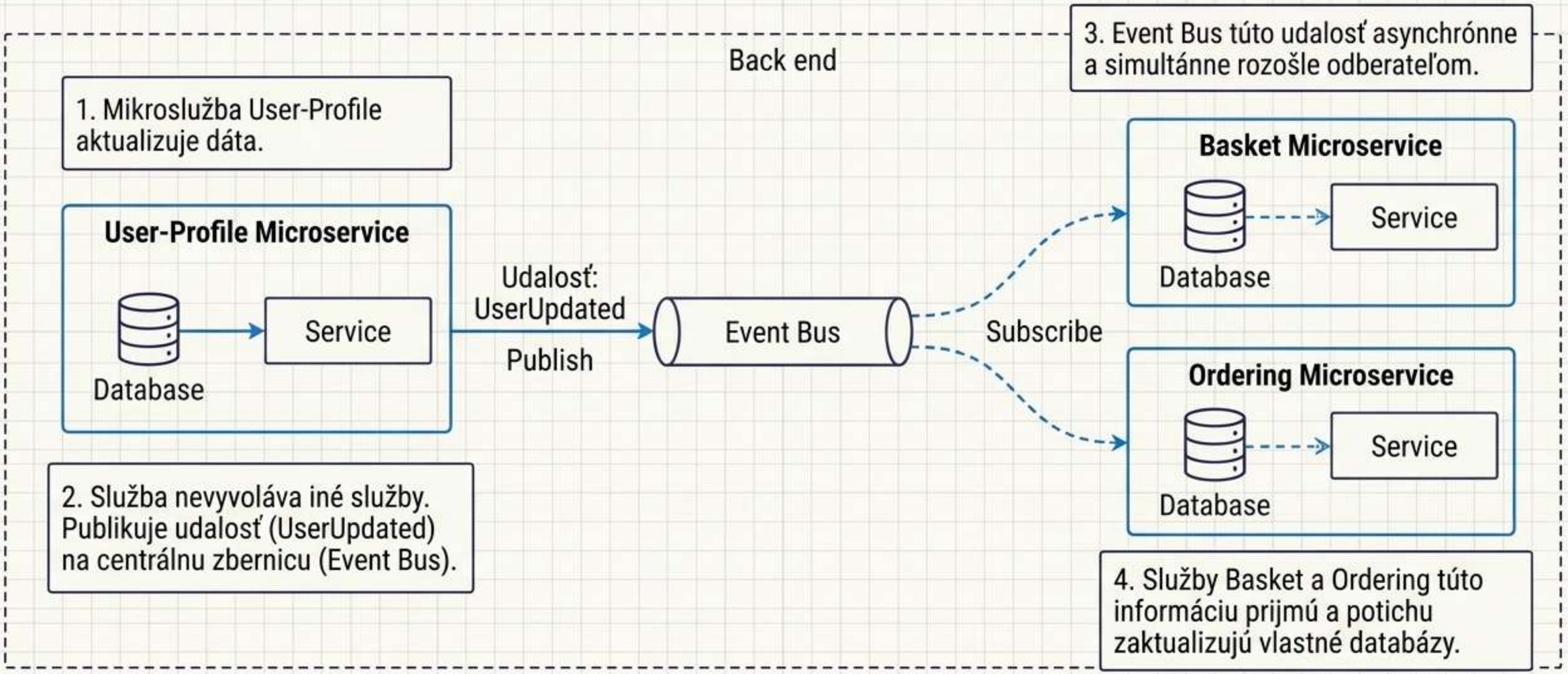
1 Synchrónna komunikácia (HTTP Request/Response)

Proces čaká na odpoveď (blocking). Ak zlyhá jedna služba v reťazci, celá operácia zlyhá (kaskádové zlyhanie). Anti-pattern pre internú komunikáciu.

2 Asynchrónna komunikácia (Message Broker / AMQP)

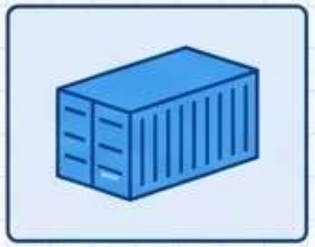
Služba odošle správu do rady a ihneď pokračuje (non-blocking). Odolné voči výpadkom – ak je príjemca offline, správa počká v rade. Voľná väzba (loose coupling).

Reakcia na udalosti: Event-Driven Architecture



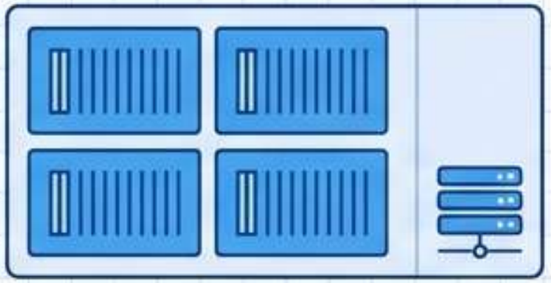
Základný prínos (Open/Closed Principle): Do systému môžeme pridať ďalších 10 služieb bez zásahu do kódu pôvodnej služby.

Nasadenie a orchestrácia: Docker & Kubernetes



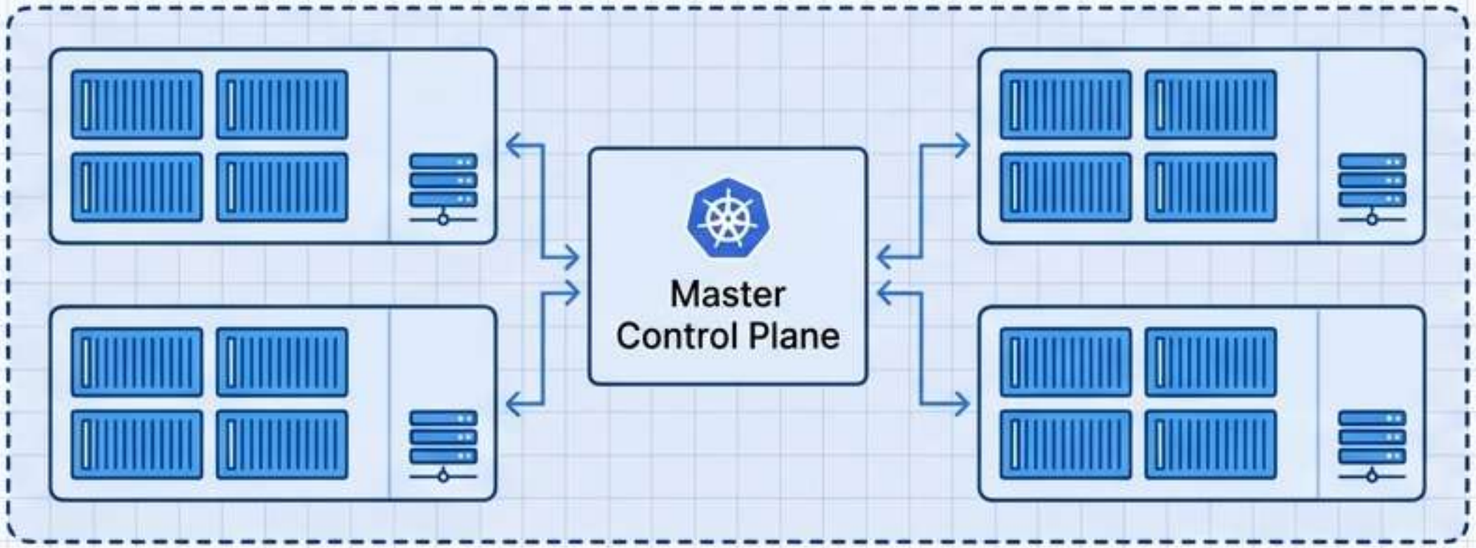
Úroveň 1: Kontajner (Docker)

Jednotka nasadenia. Obsahuje aplikáciu a všetky jej závislosti. Izolované prostredie.



Úroveň 2: Pod / Uzol (Virtuálny server)

Jeden virtuálny server bežiaci na Linuxe, ktorý nesie desiatky rôznych kontajnerov.

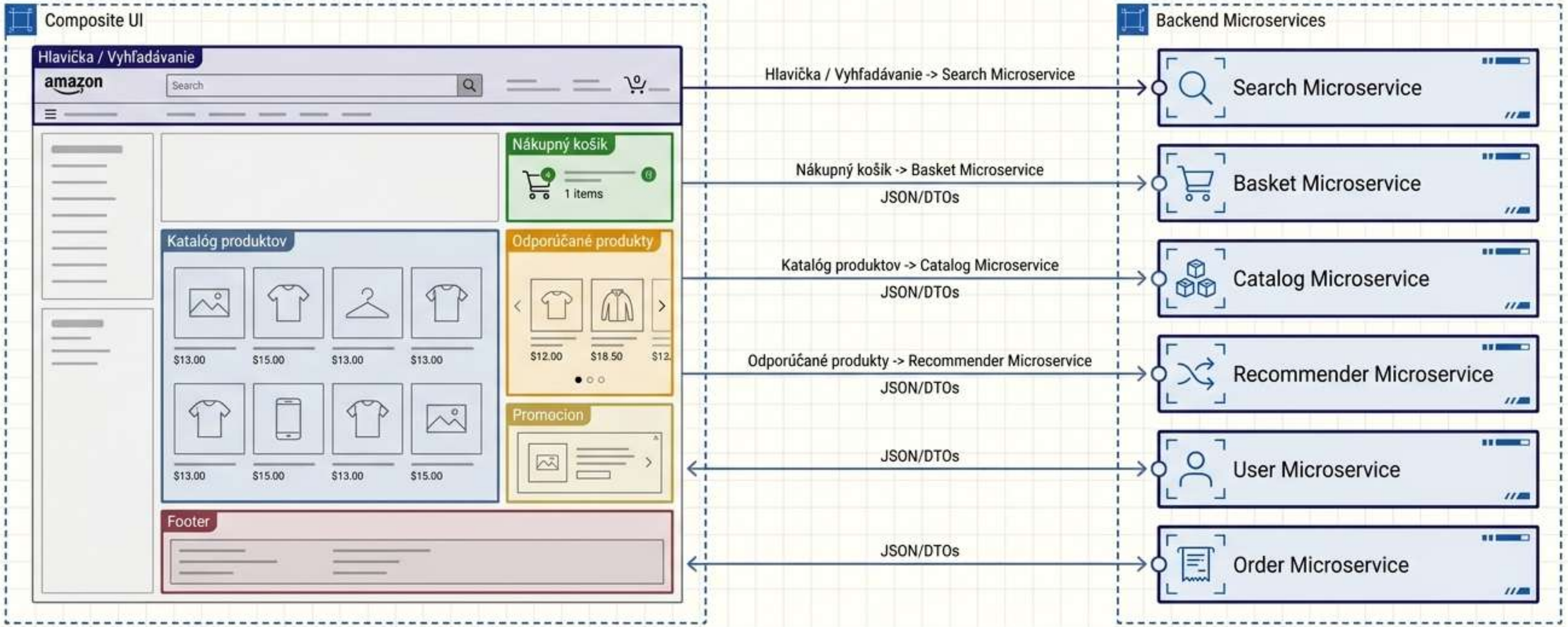


Úroveň 3: Klaster a Orchestrátor (Kubernetes)

Manažér klastra. Automaticky sleduje zdravie ✅ ⚠️ (Liveness/Readiness) kontajnerov a systémovú záťaž. ✅

⚠️ **Dynamika systému (Elasticita):**
Orchestrátor automaticky namnoží kontajnery pri vysokej záťaži a zlikviduje ich, keď záťaž klesne.

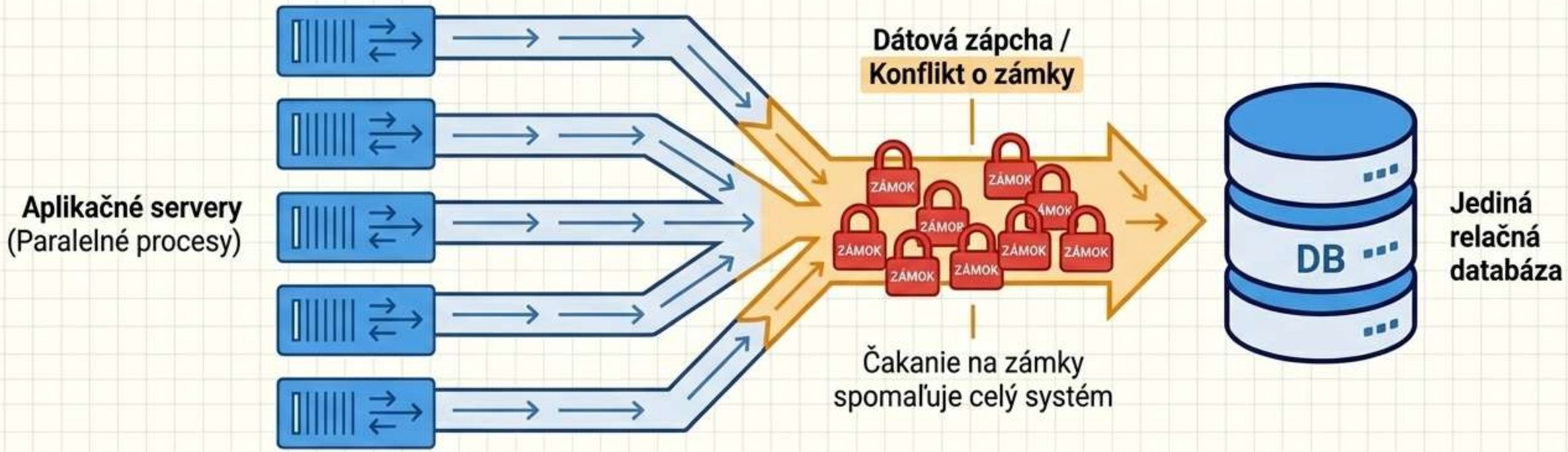
Ilúzia jednotnosti: Kompozitné používateľské rozhrania



Monolitické používateľské rozhranie iba vizuálne agreguje oddelené, autonómne systémy bežiace v cloude.

Fyzika systémov: Teória konfliktov a zámkov (Lock Contention)

Vedecký základ:
J. Gray et al., 1996 (The dangers of replication and a solution)



Pokus o zrýchlenie:

Pridávanie ďalších aplikačných serverov k jednej relačnej databáze zlyháva.

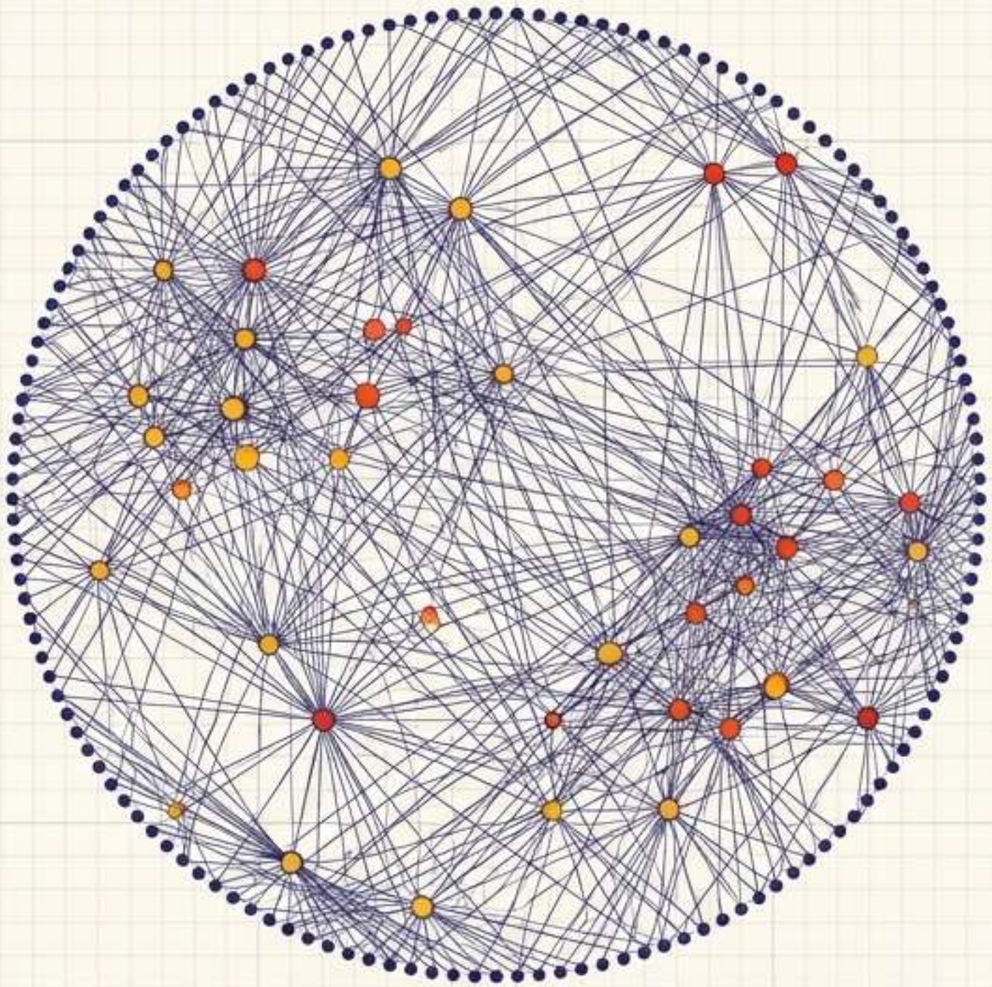
Paradox narodenín:

Čím viac paralelných procesov k databáze pristupuje, tým exponenciálne stúpa šanca, že chcú upraviť rovnaký objekt.

Umelá zápcha (Zámky):

Pridanie serverov len zhorší zápchu. Všetci stoja v rade na zámok. Jediné riešenie je systém rozsekať (Divide and Conquer).

Temná stránka: Komplexnosť a kaskádové zlyhania



Kritické riziká:

Kaskádové porušenia QoS

Jedna pomalá mikroslužba môže spôsobiť pretečenie radov, reštarty a dominový efekt v celej architektúre (Performance unpredictability).

Kritické riziká:

Náročnosť ladenia (Debugging)

Izolovať koreňovú príčinu problému, keď jedna požiadavka prejde stovkou mikroslužieb v asynchrónnom režime, si vyžaduje masívne nástroje na distribuovaný tracing.

Presunuli sme zložitosť z vnútra kódu (monolit) do vzájomnej sieťovej komunikácie.

Biznisová realita: Riziko Vendor Lock-in

Prípadová štúdia

Dropbox

Od 2007 do 2016 Dropbox bežal kompletne na úložisku Amazon S3.

Riziko: Ak by poskytovateľ zdvojnásobil ceny alebo platformu vypol, Dropbox by skolaboval. Svoju kľúčovú infraštruktúru nevlastnili.

Vyústenie: V roku 2016 Dropbox masívne investoval do migrácie z cloudu na vlastný hardvér.

Záverečný paradox cloudu:

Moderný cloud dáva firmám bezprecedentnú agilitu, škálovateľnosť a rýchlosť uvedenia na trh.

Zároveň však vytvára tvrdú technologickú závislosť na API jedného poskytovateľa. Architektúra musí s týmto strategickým rizikom od začiatku počítať.