UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

3. Elosztott rendszerek

Dr. Bilicki Vilmos
Szegedi Tudományegyetem
Informatikai Tanszékcsoport
Szoftverfejlesztés Tanszék

Tartalom

- Bevezető
- Felmerülő problémák
- Alkalmazás architektúrák
- Elosztott rendszer architektúrák
- SOA koncepció
- Virtualizációs szintek
- Összefoglaló

Valós nagy rendszerrel kapcsolatos tapasztalatok (Inktomi)

Összeomlások/Merevlemez hibák	Óránként
Adatbázis frissítések	Naponta
Szoftver frissítések	Hetente/Havonta
OS frissítések	Kétszer
Energetikai hibák	néhány
Hálózat hibák	11
Az egész infrastruktúra fizikai átköltöztetése	2

Motiváció – nem funkcionális követelmények

- Rendelkezésre állás: azon idő amely alatt a komponens képes kielégíteni a funkcionális követelményekben meghatározott funkciókat azon időhöz viszonyítva amikor ezt nem tudja megtenni.
- Kapacitás/Skálázhatóság: A komponens a terhelés növekedésével is képes ellátni a funkcionális követelményekben megfogalmazott feladatokat.
- Párhuzamosság: A komponens a több egymással párhuzamos terhelés hatására is képes ellátni a funkcionális követelményekben megfogalmazott feladatokat.
- Fejleszthetőség: A komponens új funkcionális követelményeket is el tud látni viszonylag szerény fejlesztési ráfordítással.
- Együttműködési képesség: A komponens képes környezetével és a szükséges komponensekkel aránytalan plusz terhelés okozása nélkül is együttműködni.
- Késleltetés: A komponens az adott funkcionális követelményben szereplő szolgáltatását adott terhelés mellett adott időtartamon belül kiszolgálja.
- Karbantarthatóság: A komponensnek támogatnia kell az adott szervezetben definiált karbantartási feladatokat (pl.: backup)
- Menedzselhetőség: A komponensnek támogatnia kell a megfelelő konfigurálhatósági szintet.
- Monitorozhatóság: A komponensnek monitorozhatónak kell lennie.
- Visszaállíthatóság: Hibás adat esetén a komponensnek támogatnia kell a megfelelő állapot visszaállítását.
- Biztonság: A komponensnek a helyi biztonsági előírásoknak megfelelően kell működnie.
- Konzisztencia: A komponens az adatot konzisztens állapotban tartja.

Rendelkezésre állás

- Az alábbi paraméterektől függ
  - Megbízhatóság:
    - Annak az esélye, hogy egy rendszer adott ideig hibamenetesen működik (λ meghibásodási ráta)

$$R(t) = e^{-\lambda t}$$

  - Karbantarthatóság:
    - Annek az esélye, hogy egy meghibásodás után egy adott időtartam alatt visszaáll a működőképes állapot (μ megjavítási ráta)

$$M(t) = 1 - e^{-mt}$$

  - A rendelkezésre állás a rendszer működőképes ideje és a teljes idő hányadosa

$$A = \frac{\text{Up-Time}}{\text{Up-Time + Down-Time}}$$

Fogalmak

- MTTF Mean Time To Fail:

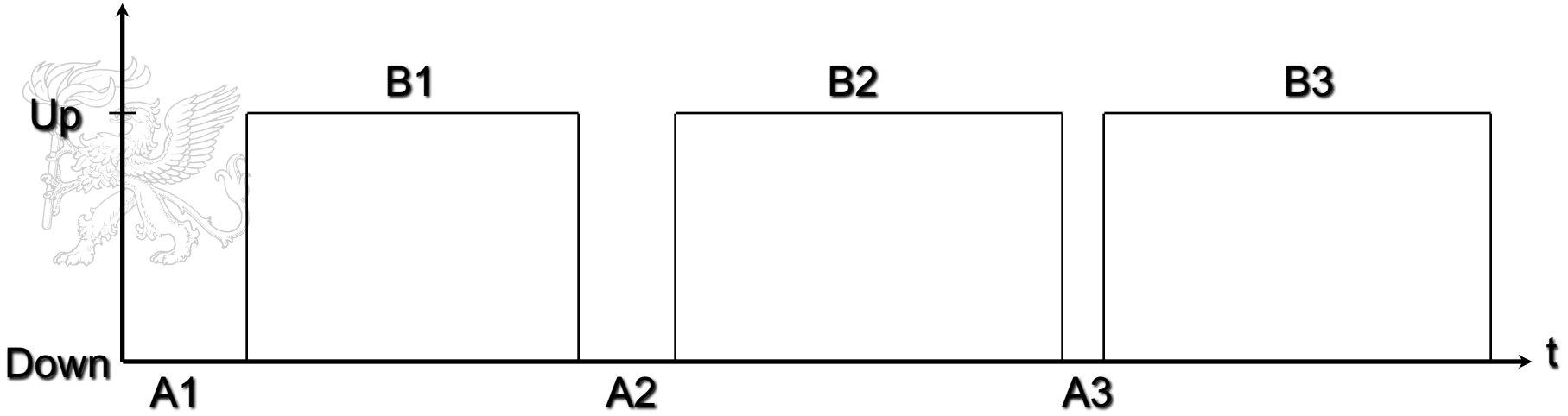
$$MTTF = \frac{B1 + B2 + B3}{3}$$

- MTBF Mean Time Between Failure:

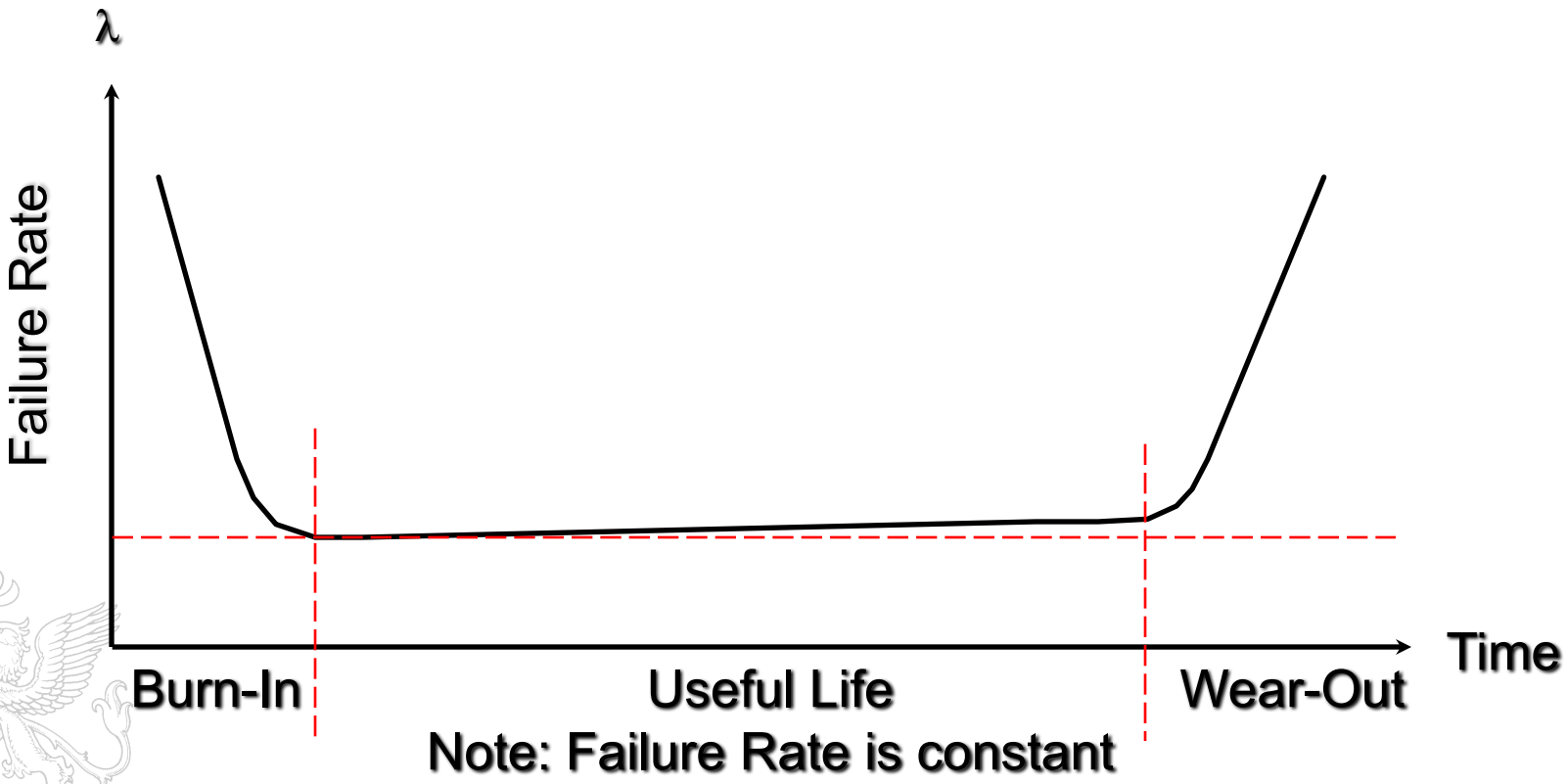
$$MTBF = \frac{(A1 + B1) + (A2 + B2) + (A3 + B3)}{3}$$

- MTTR Mean Time To Repair:

$$MTTR = \frac{A1 + A2 + A3}{3}$$



Fürdőkád görbe



Megoldás: redundancia

- Aktív redundancia
  - Elem/komponens duplikálás
  - Rendszer duplikálás
- Passzív redundancia
  - Hot Standby
  - Warm Standby
  - Cold Standby
- Soros rendszer:

$$R_s = p_1 \cdot p_2 \ldots\ldots p_n$$

- Párhuzamos rendszer:

$$R_s = 1 - q_1\, q_2 \ldots\ldots q_m$$

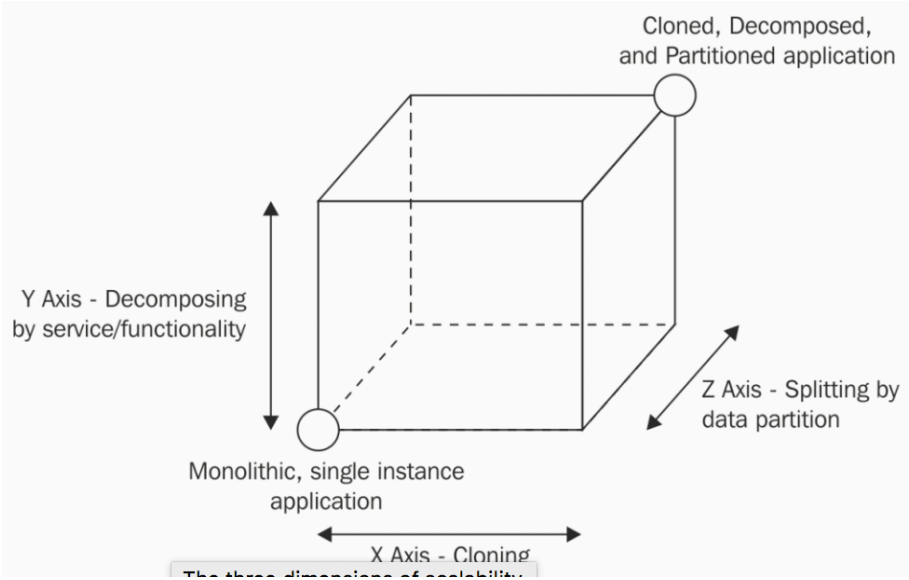
$$q_1 = 1 - p_1$$
$$q_2 = 1 - p_2$$

Példa

- Amazon EC2:
  - 99.95% átlagos rendelkezésre állás egy év alatt
  - Évi 4,38 óra leállás
- Google Apps:
  - 99.9% havi alapon
  - Évi 8,76 óra leállás, havi 0,73 óra leállás
- Serveloft:
  - Hálózat: 99,9%, 4 órás helyreállítási idő
  - Gép: 4 órás helyreállítási idő
- Rackspace:
  - Hálózat: 100% (karbantartáson kívül)
  - Infrastruktúra: 100% havonta (karban tartáson kívül)
  - Gép: 1 órán belül csere

Skálázás kocka

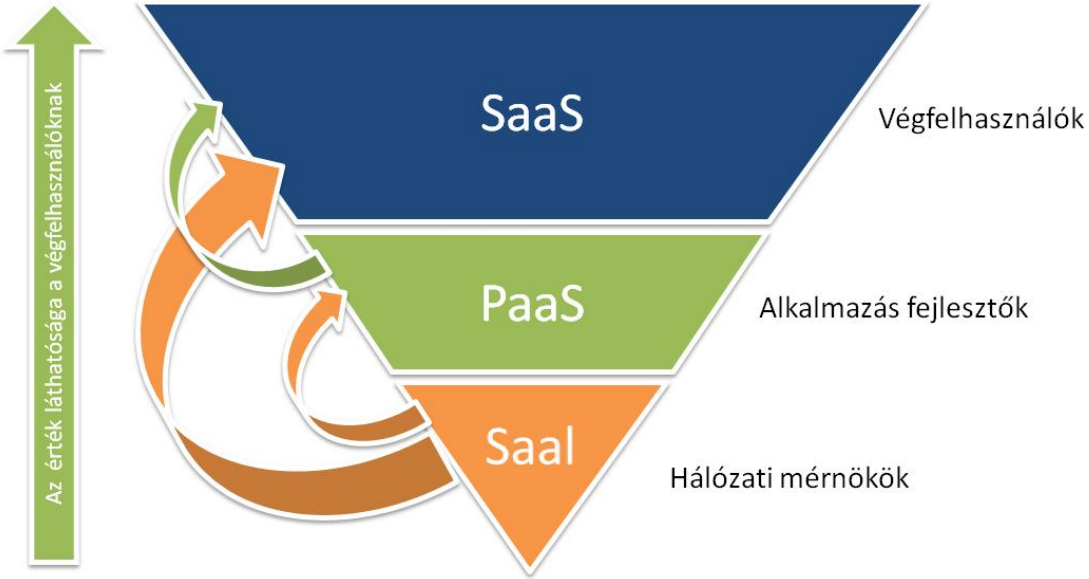
- Dimmenziói
  - Klónozás
  - Dekompozíció (szolgáltatás, funkcionalitás)
  - Adat partícionálás

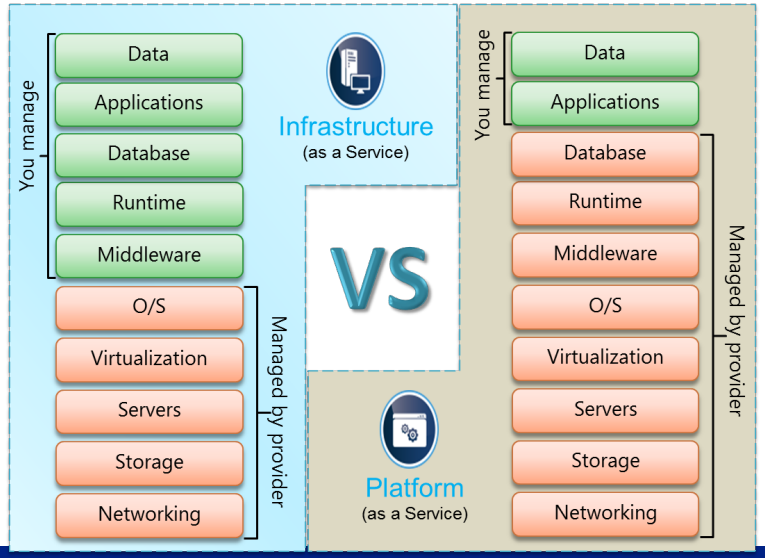


Skálázhatóság

- Adat tárolás/Feldolgozás
  - Vertikális (Scale up)
    - Fizikai korlátai vannak (processzorok száma, JVM szemétgyűjtő, …)
    - Közös memória mint kommunikációs médium
  - Horizontális (Scale out)
    - Elosztott rendszer (Leslie Lamport: „Elosztott rendszer az ahol egy addig teljesen ismeretlen számítógép hibája teszi használhatatlanná a gépünket”)
    - Elvileg korlátlan (elosztottságból fakadóan nem lehet ACID)
    - Távoli hozzáférés problematikája (érték szerint vs. referencia szerint)

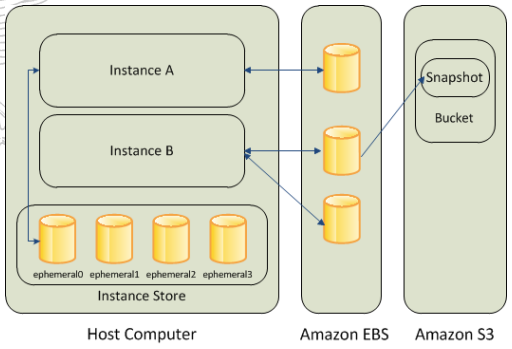
Virtualizációs rétegek

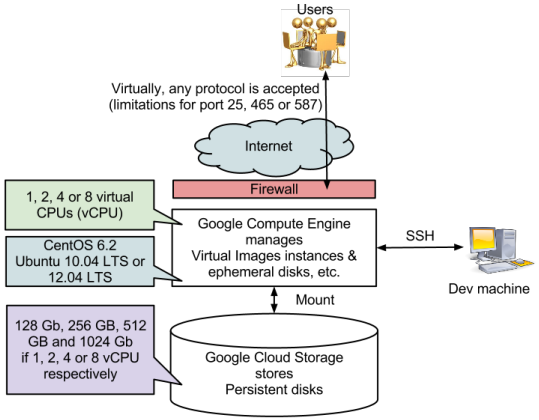




IaaS - Infrastruktúra mint Szolgáltatás

- Valós vagy virtualizált HW
  - Processzor
  - Memória
  - I/O
  - Hálózat
- Függ a fizikai eszközöktől
- Példák:



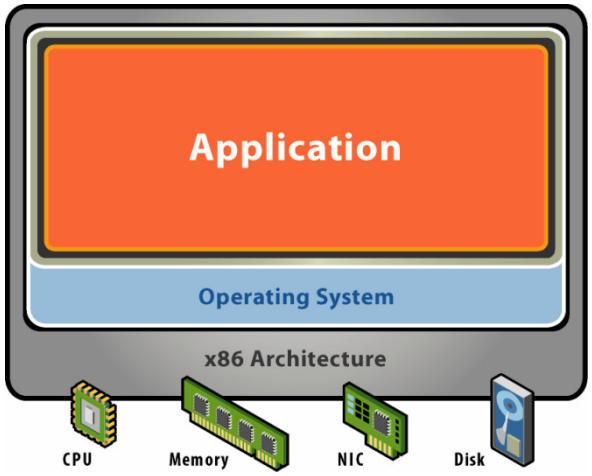


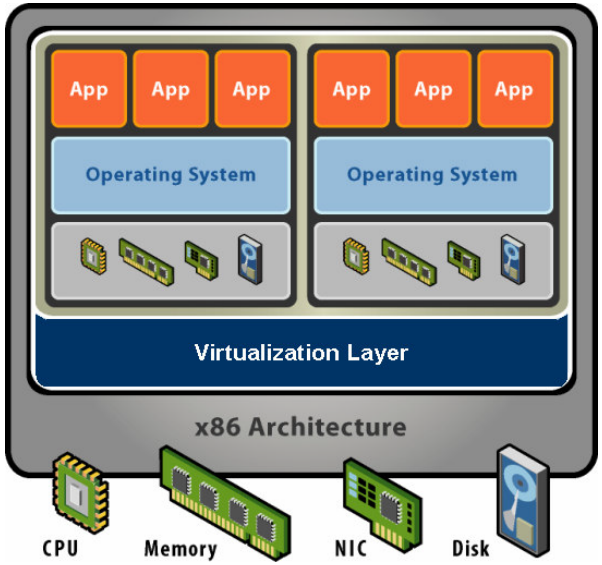
Virtualizáció

- Absztrakt fizikiai komponensek logikai objektumok segítségével
- A fizikai és logikai objektumok dinamikus kötése
- Példa:
  - Hálózat: VLAN, VPN, P2P
  - Tárhely: SAN
  - Számítógép: Virtuális Gép

Fizikai vs. Virtuális gép

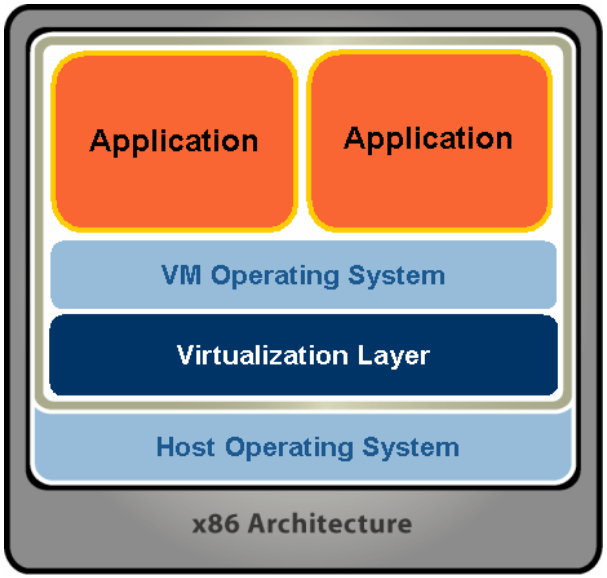
- Fizikai HW:
  - Processzor, memória, chipset, I/O busz
  - A fizikai erőforrások gyakran üresjáratban vannak
- Szoftver:
  - Szorosan kapcsolódik a HW-hez
  - Egy aktív OS kép
  - Az OS vezérli a HW-t
- Hardver szintű absztrakció:
  - Virtuális HW: processzor, memória, chipset, I/O eszközök, …
  - Magában foglalja az OS és alkalmazás állapotot
- Virtualizációs SW:
  - Elválasztja a HW-t és az OS-t
  - A vendég OS-ek felé multiplexálja a HW-t
  - Erős szeparáció
  - Kezeli a fizikai erőforrásokat

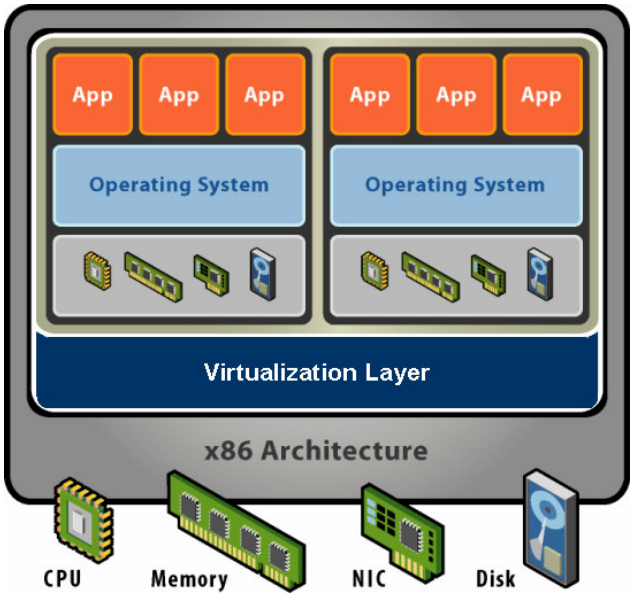




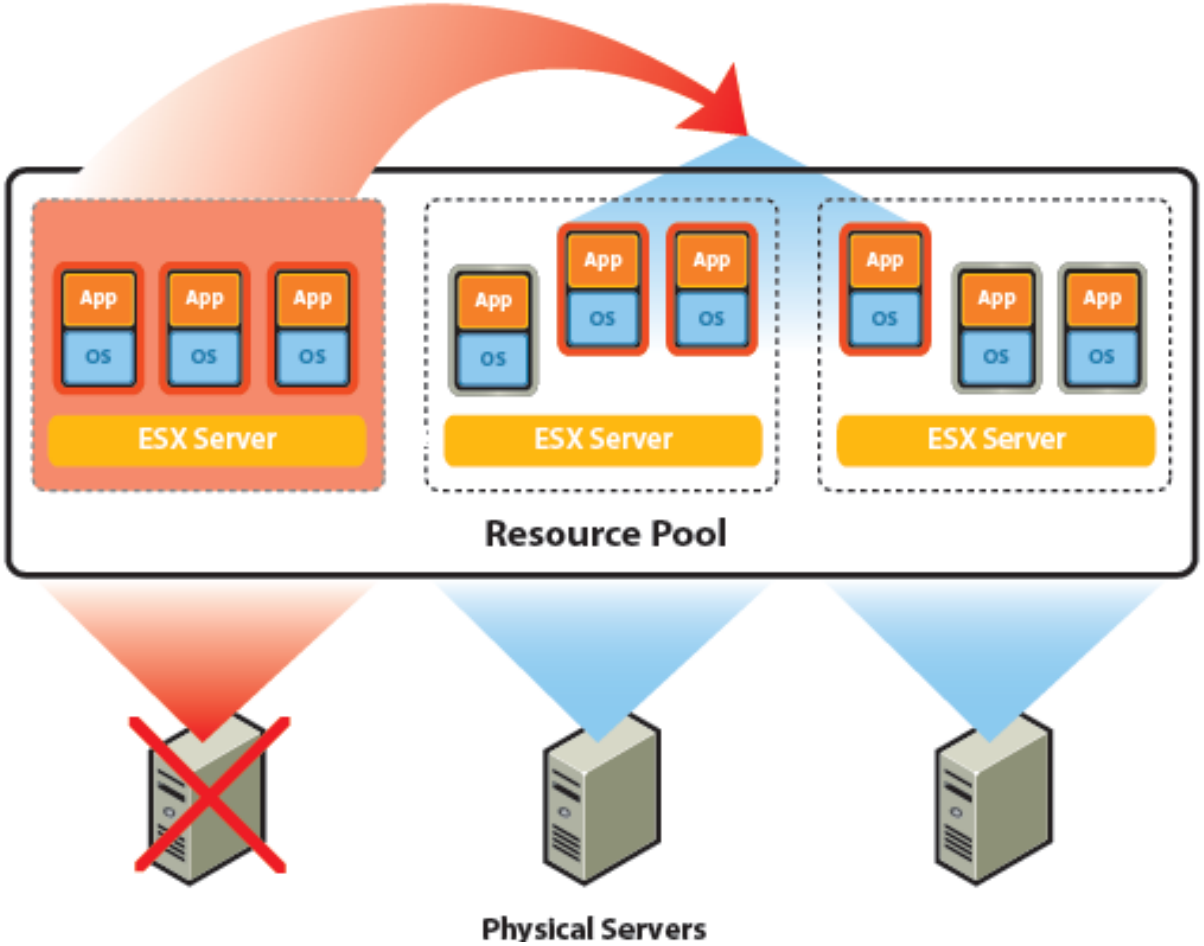
Típusai

- Type 2: Hosztolt architektúra:
  - Befogadó Oprendszer felett van futtatva
  - Kicsi kontextus váltó meghajtó
- Type 1: Csupasz fém architektúra
  - A Hypervisort közvetlenül a HW-re telepítik
- Paravirtualizáció
  - Az operációs rendszer tud a Hyervisorról

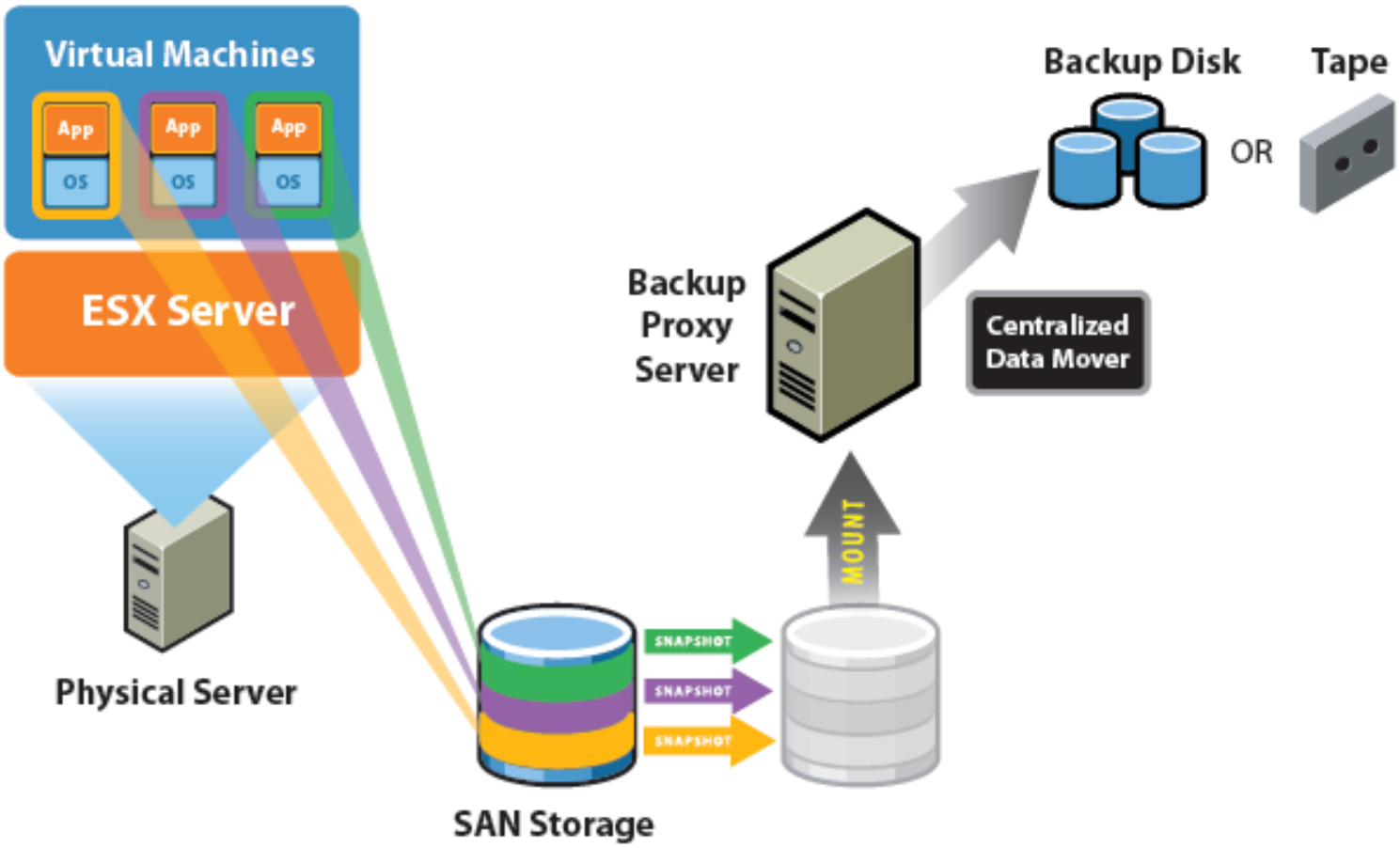




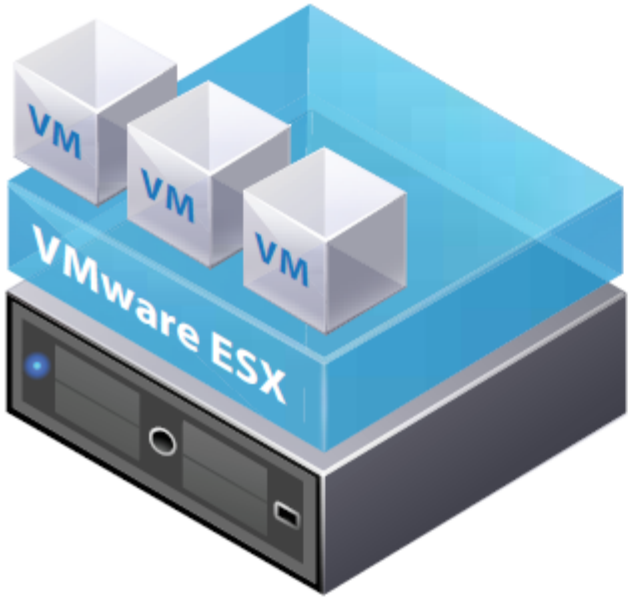
Magas rendelkezésre állás

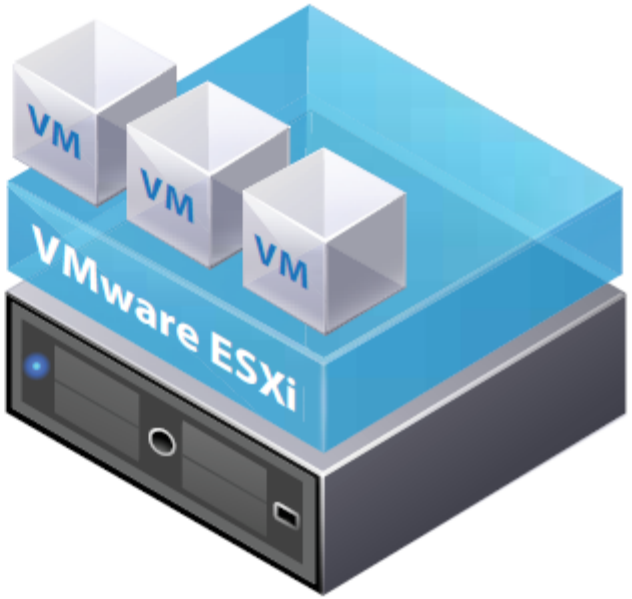


Konszolidált mentés

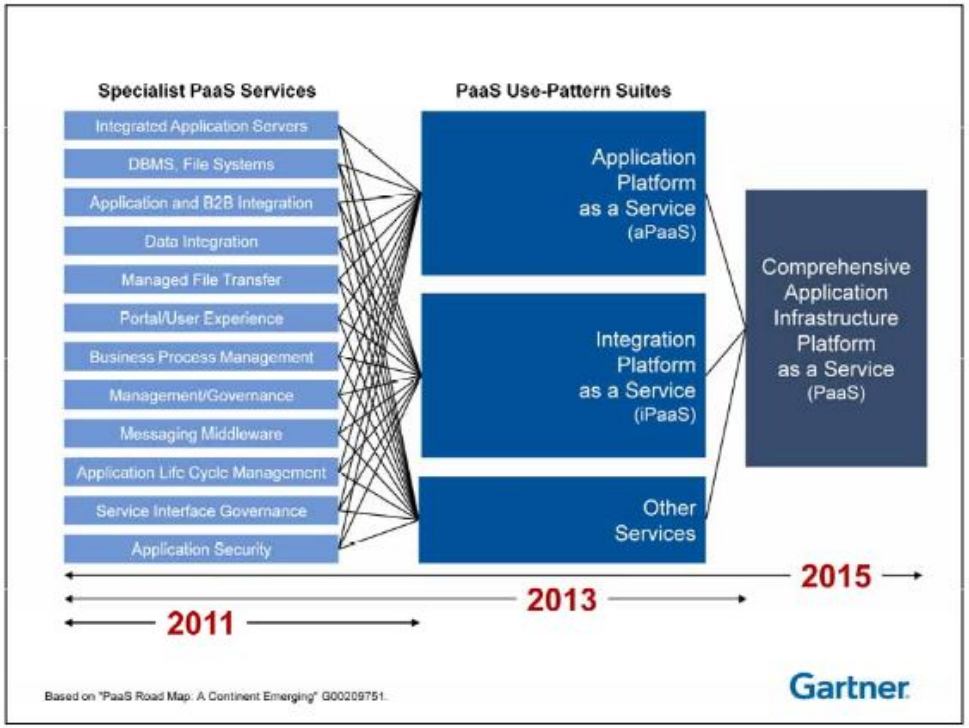


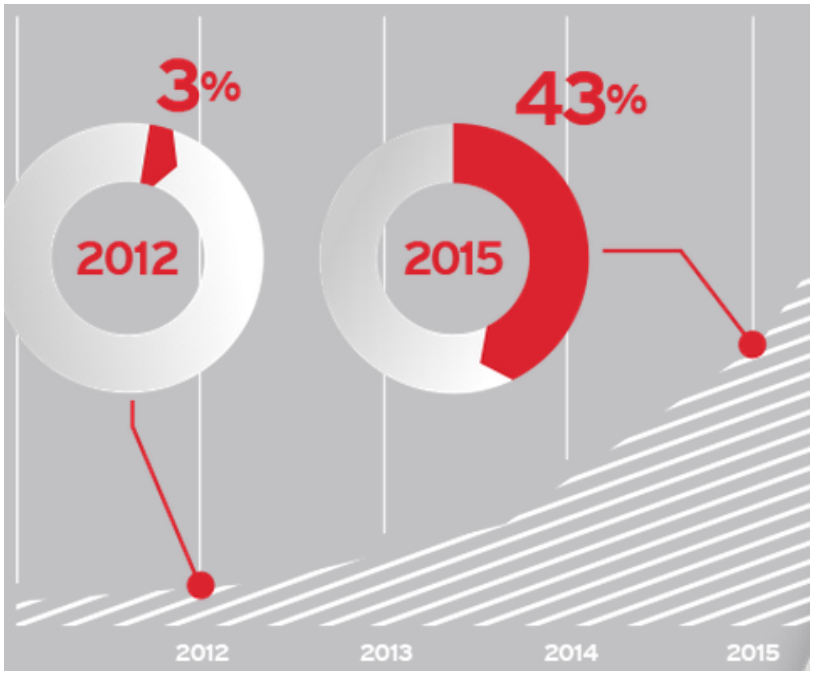
Probléma:





PaaS - Gartner





PaaS Felmerülő problémák

- Leslie Lamport egy igen frappáns definíciót adott az eloszotott rendszerekre:
- "*A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable*" azaz „*Elosztott rendszer az, ahol egy olyan számítógép hibája teszi használhatatlanná a saját gépünket, amiről azt sem tudtuk, hogy létezik.*"
- Egy szabatosabb definíció Wolfgang Emmerichtől: „*Elosztott rendszernek nevezzük az autonóm, egymással hálózaton együttműködő számítógépeket, amelyek közös együttműködésén alapuló szolgáltatását a felhasználó úgy érzékeli, mintha azt egy integrált eszköz szolgáltatta volna.*"

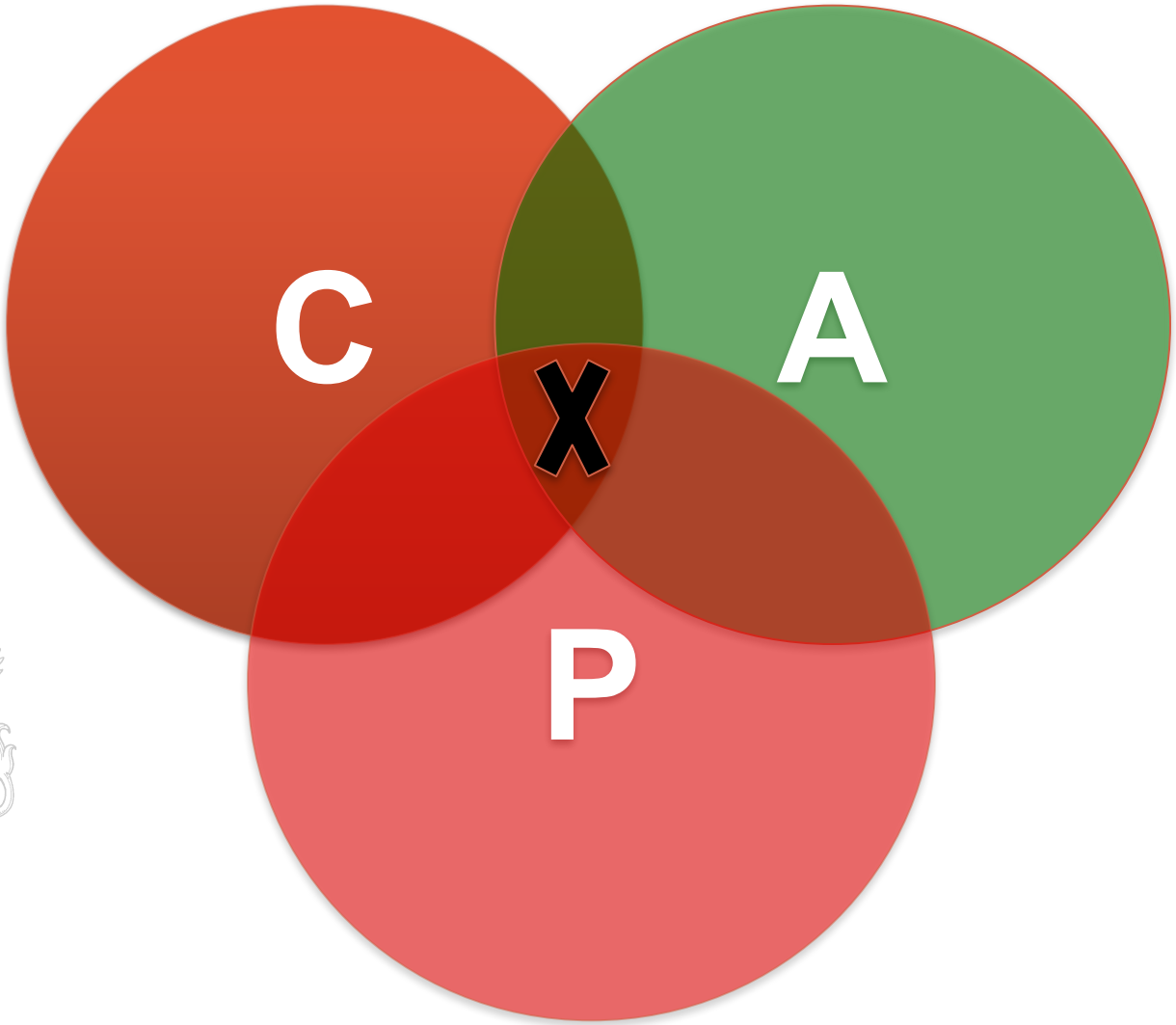
CAP Tétel

- Prof. Eric Brewer sejtése 2000
- Elosztott rendszer tervezési döntési pontok
- Nem lehet egyszerr mindhármat kielégíteni:
  - **Konzisztencia**
  - **Rendlkezésre állás**
  - **Partíció tűrés**

CAP Tétel

- **K**onzisztencia:
  - Adott időpillanatban minden csomópont ugyanazon adatot látja
- **R**endelkezésre álás:
  - A kieső csomópontok nem akadályozzák a működőket a működésben
- **P**artíció tűrés:
  - A rendszer partíciók esetén istovább működik
- Egy elosztott rendszer ezek közül csak tettőt tud teljesíteni **mindhármat nem**

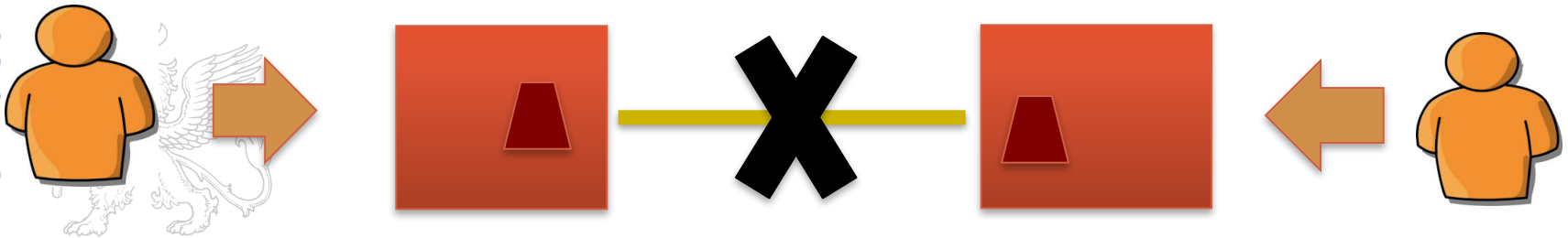
CAP Tétel



CAP Tétel

- Példa

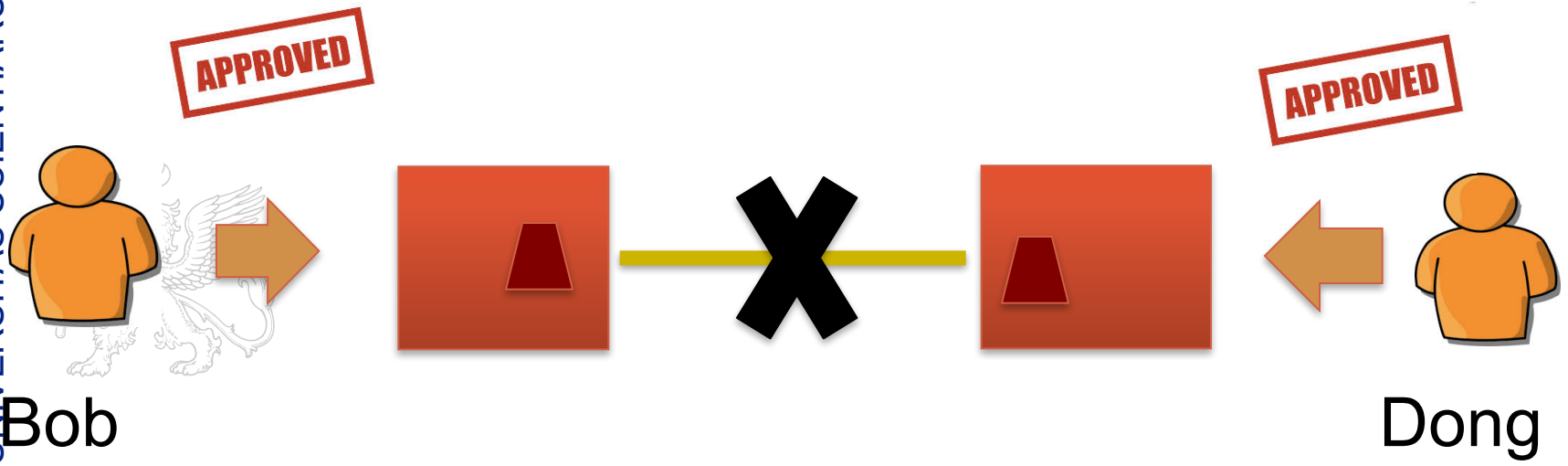
Hotel Foglalás: Lefoglaljuk ugyanazt kétszer?



CAP Tétel

- Példa

Hotel foglalás: Lefoglaljuk ugyanazt kétszer?

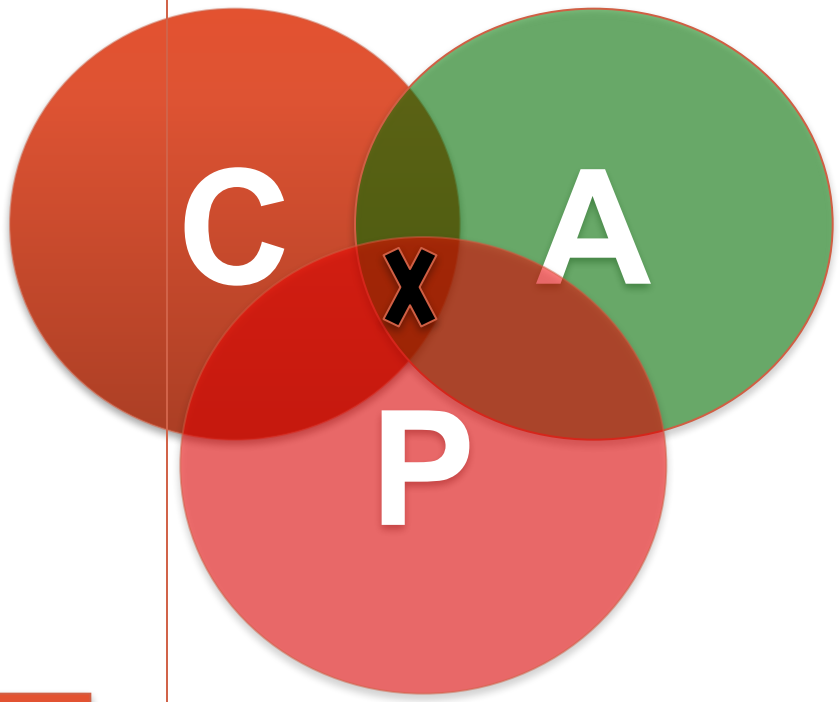


Miért fontos?

- Az adatbázis a jövőben **elosztottak** lesznek (Big Data Trend, etc.)
- CAP tétel megadja a döntési pontokat
- Meg kell értenünk a lehetőségeket és a veszélyeket

A CAP tétel újragodolása

- Ezek közül kettő teljesül:
  - Consistency
  - Availability
  - Partition tolerance
- Válasszunk kettőt
- A fentiek alapján ezek lehetségesek:
  - CP
  - AP
  - CA



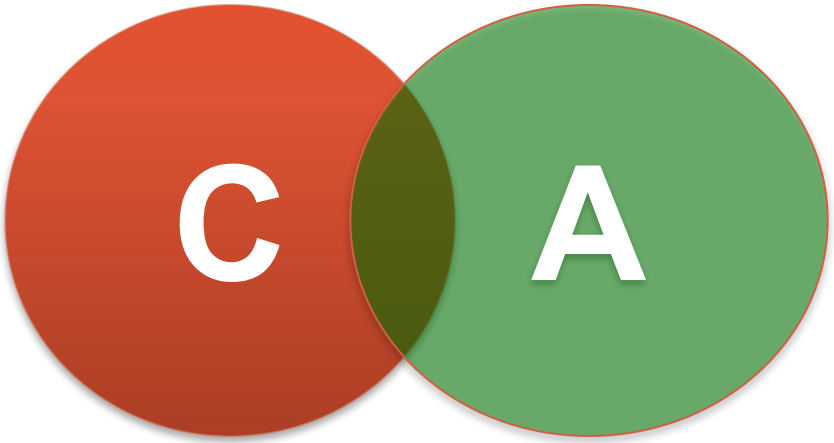
Bármilyen probléma?

Relációs adatbázis skálázás

- A relációs adatbázisok az **ACID** (Atomicity, Consistency, Isolation, Durability) képességeket szeretnék teljesíteni
- A fentiek alapján egy igazán elosztott relációs adatbázisnak mindhárom paramétert teljesíteni kelene, Ez lehetetlen.

Egy népszerű hiba:

- Mi lenne a CA?
- Lehet egy elosztott rendszer konzisztens és rendelkezésre álló hibás hálózat esetén?



Néhány vélemény

- Coda Hale, Yammer mérnök:
  - “Of the CAP theorem’s Consistency, Availability, and Partition Tolerance, **Partition Tolerance is mandatory in distributed systems**. You cannot not choose it.”

http://codahale.com/you-cant-sacrifice-partition-tolerance/

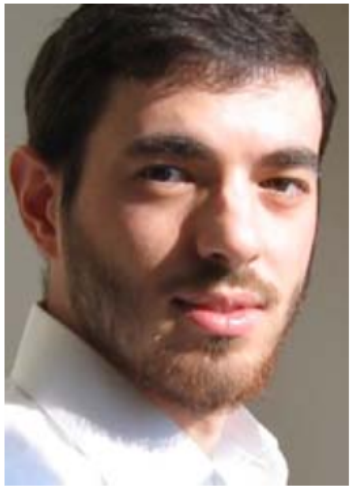
A few witnesses

- Werner Vogels, Amazon CTO
  - "An important observation is that in larger distributed-scale systems, network partitions are a given; therefore, **consistency and availability cannot be achieved at the same time**."

http://www.allthingsdistributed.com/2008/12/eventually_consistent.html

A few witnesses

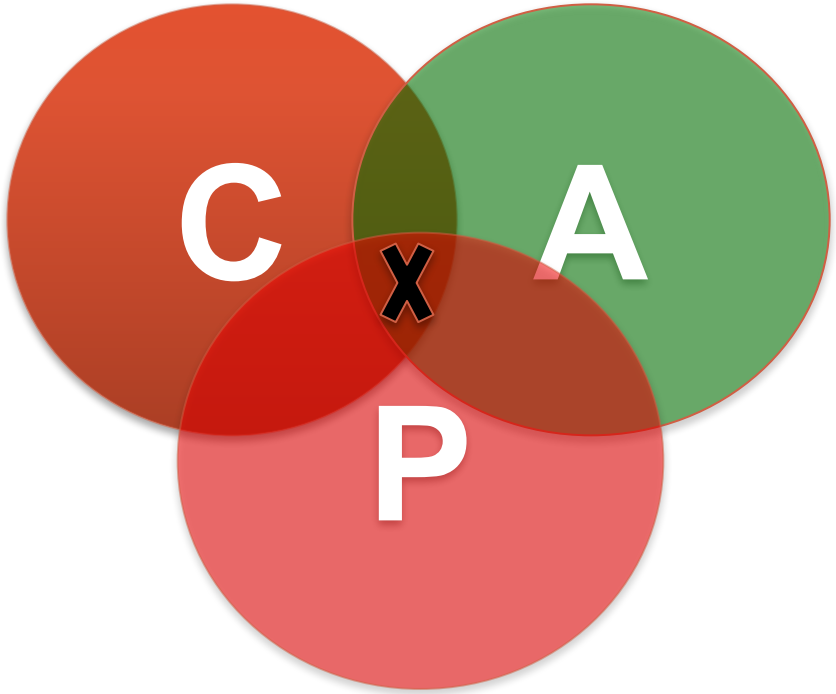
- Daneil Abadi, Co-founder of Hadapt
  - So in reality, there are only two types of systems ... I.e., if there is a partition, **does the system give up availability or consistency?**



http://dbmsmusings.blogspot.com/2010/04/problems-with-cap-and-yahoos-little.h

Konzisztencia vagy rendelkezésre állás

- Nem bináris döntés
- AP csökkenti a konzisztencia követelményt, de nem szünteti azt meg
- CP csökkenti a rendelkezésre állás követelményt, de nem szünteti azt meg
- A CP és az AP is lehet köztes megoldás



AP: Legjobb szándék szerinti konzisztencia

- Példa:
  - Web Gyorstár
  - DNS
- Jellemzők:
  - Optimista
  - Expiration/Time-to-live
  - Konfliktus feloldás

CP: Legjobb szándék szerinti rendelkezésre állás

- Példa:
  - Többségi protokollok
  - Elosztott zárolások (Google Chubby Lock service)
- Jellemzők:
  - Pesszimista zárolás
  - A kissebségi partíció nem működik

A konzisztencia típusai

- Erős konzisztencia
  - Az írás után **bármely ezt követő** hozzáférés ugyanezt az eredményt adja vissza
- Gyenge konzisztencia
  - **Nem garantált** az, hogy az írást követő olvasások ugyanazt az értéket kapják
- **Végső soron konzisztens**
  - A gyenge konzisztencia egy típusa
  - Garantált, hogy **amennyiben nincs több írás akkor előbb utóbb** minden hozzáférés az utolsó eredményt fogja kapni

Végső soron konzisztens variációk

- Kauzális konzisztenica
  - A kauzális viszonyban lévő folyamatok konzisztens adatot fognak látni
- Olvasd a saját írásod konzistencia
  - A folyamat a saját írsát mindég visszakapja
- Viszony konzisztencia
  - Amíg a viszony tart addig a rendszer garantája az olvasd a saját írásodta konzisztenciát

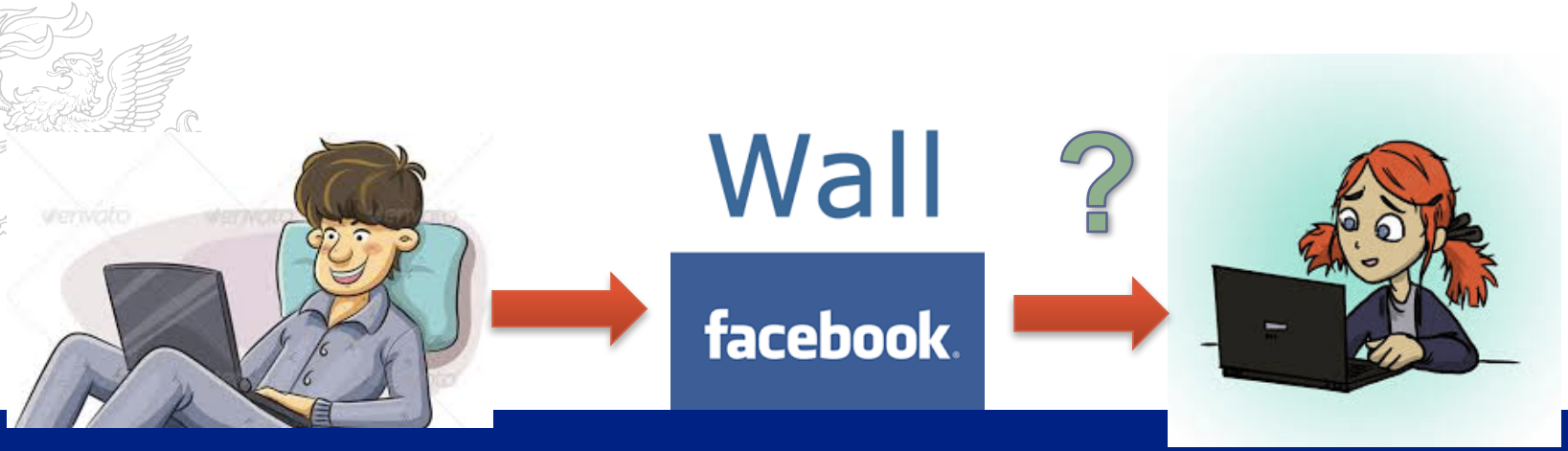
Végső soron konzisztens variációk

- Monoton olvasás konzisztenica
  - Amennyiben egy folyamat már egy értéket látott attól régebbit nem láthat
- Monoton írás konzisztencia
  - A rendszer az írásokat folyamat szinten sorosítja
- A gyakorlatban
  - Több típus is kombinálható
  - Monoton olvasás és olvasd az írásod javasolt

Facebook

- Bob egy érdekes történetet ost meg Alizzal
- Jelzi ezt Aliznak
- Alíz belép de:

  - Semmit sem talál!



Végsősoron konzisztens

- Bob szól, hogy egy kicsit később nézze meg
- Alíz vár és újra megnézi

- Most már látja a Bob által megosztott történetet

Végsősoron konzisztens

- Megoldás: azért van így mert a Facebook a **végsősoron konzisztens modellt használja**
- Mirt nem az erős konzisztenciát?
  - Több mint 1 millárd felhasználó
  - Nem triviális ezt erős konzisztenciával kezelni
  - A végsősoron konzisztens megoldás **csökkenti a terhelést és növeli a rendelkezésre állást**

Dinamikus választás C és A között

- Egy repülő társaság foglaló rendszer
  - Amikor sok az üres szék akkor a konziszetncia kevésbé fontos, fontosabb a rendelkezésre állás
  - Amikor a gép majdnem tele van akkor a konzisztenica fontosabb mint a rendelkezésre állás

Mi van akkor ha nincs partíció?

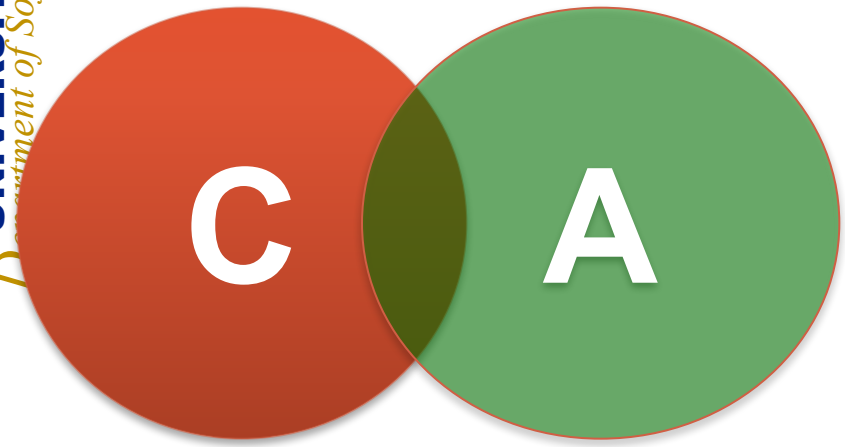
- Döntenünk kell a **Konzisztencia** és a **Késleltetés** között:
- **a hiba lehetősége**
  - Magas rendelkezésre állás -> replikát adat -> konzisztenica probléma
- Alap ötlet:
  - A rendelkezésre állás és a késleltetés ugyanaz: : nem áll rendelkezésre -> extrém nagy késleltetés

CAP -> PACELC

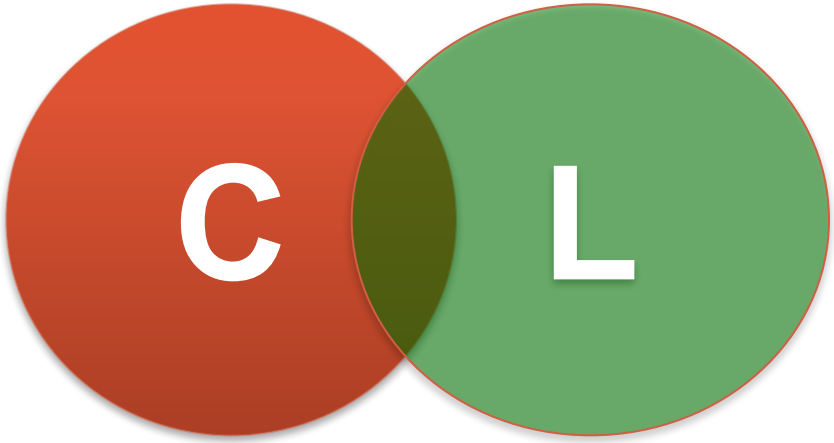
- Egy jóval gyakorlat közelibb megközelítésmód:
  - Amennyiben van **partíció (P)**, akkor döntenünk kell a **rendelkezésre állás és a konzisztencia között (A and C)**; **egyébként (E)**, amikor minden rendben van akkor **késleltetés (L) vagy konzisztencia(C)**?

Abadi, Daniel J. "Consistency tradeoffs in modern distributed database system design." Computer-IEEE Computer Magazine 45.2 (2012): 37.

PACELC



Partícionál
t



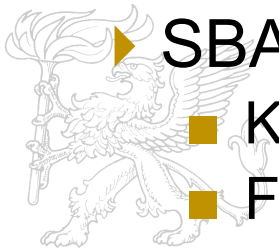
Normál

Példák

- **PA/EL rendszer:** Adjuk fel a C-t a rendelkezésre állásért és a konzisztenciáért
  - Dynamo, Cassandra, Riak
- **PC/EC rendszer:** Nem adjuk fel a konzisztenciát így a rendelkezésre állással és a késleltetéssel fizetünk
  - BigTable, Hbase, VoltDB/H-Store
- **PA/EC rendszer:** Partíció esetén adjuk fel a konzisztenciát, egyébként adjuk fel a válaszidőt
  - MongoDB
- **PC/EL rendszer:** Konzisztencia ha partíció van, de adjuk fel a konzisztenciát ha normál üzem van
  - Yahoo! PNUTS

Elosztott rendszer architektúrák

- CS – Kliens szerver
- P2P
- N rétegű architektúra
- Szorosan csatolt rendszer
- Lazán csatolt rendszer (EDA)
  - Esemény
  - ESP
  - CEP
- SBA (Space Base Architecture) (PU alapú)
  - Klaszter
  - Felhő
  - Rács
  - Elosztott Objektum



Kliens - Szerver

- Szerepkörök alapján határozza meg a rendszerben résztvevő entitások helyzetét.
- A szerver oldal birtokolja az erőforrásokat,
- Kliens oldal használja azokat.
- Ezen elvek mentén működnek napjaink legnépeszerűbb protokolljai (HTTP, SMTP, DNS, stb.).
- A megközelítés előnyei
  - könnyebb karbantarthatóság, és a központosítás miatt elvileg könyebb biztonságos rendszert készíteni.
- Hátránya
  - a skálázhatóság és a robosztusság kérdése, mivel ez csak a szerver oldalon múlik.

P2P

- A kliens szerver architektúra ellentétje a P2P architektúra, ahol egyenrangú vagy nem egyenrangú, de az erőforrások megosztásában egyaránt részt vevő entitások alkotják a rendszert.
- Skype vagy a legtöbb fájlcserélő protokoll ezen az elven működik.
- A kliens-szerver megoldással szemben a hátránya a komplexitásában és elosztottságában rejlik, viszont cserébe extrém skálázhatóságot nyújt.
- Típusai:
  - Elosztott kivonat tábla (pl.: Kademina)
  - Kicsi világ (pl.: Gnutella)
- Szolgáltatás primitívek:
  - Store
  - Load
  - Search

N-rétegű architektúra

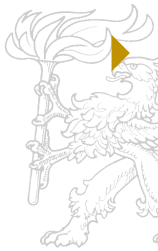
- *Megjelenítés*: a társ rendszer felé nyújt interfészt (gyakran ez a felhasználói interfész), és az ehhez kapcsolódó eseményeket kezeli le. Ennek a leggyakoribb megvalósítása a weboldal és az előállító logika.
- *Üzleti logika*: ezen réteg feladata az üzleti folyamatok futtatása, hosszú életű tranzakciók kezelése. Itt kerül sor az adatok szélesebb hatókört, több adattípust átölelő szabályainak kikényszerítése is.
- *Perzisztencia*: az adatok tartós tárolásával foglalkozik. Itt történik meg a szűkebb hatókörrel rendelkező adatszabályok kikényszerítése és gyakran az objektum és relációs adatmodellek közötti leképezés is.

Szorosan csatolt rendszerek

- A szorosan csatolt rendszerekben a komponensek (vagy ha ilyenek nincsenek, akkor az osztályok, objektumok) szinkron módon kommunikálnak egymással, szorosan követve a klasszikus függvényhívás szemantikáját.
- Ebbe a kategóriába tartoznak a távoli eljárásokon alapuló rendszerek is.
- A megközelítés előnye az, hogy a rendszer a klasszikus, egy processzen belül is megtalálható interkaciós paradigmákra épít, egyszerűvé téve ezzel a megvalósítást.
- Ezzel a megoldással viszont nehéz robosztus, skálázható rendszereket létrehozni.

Lazán csatolt rendszerek

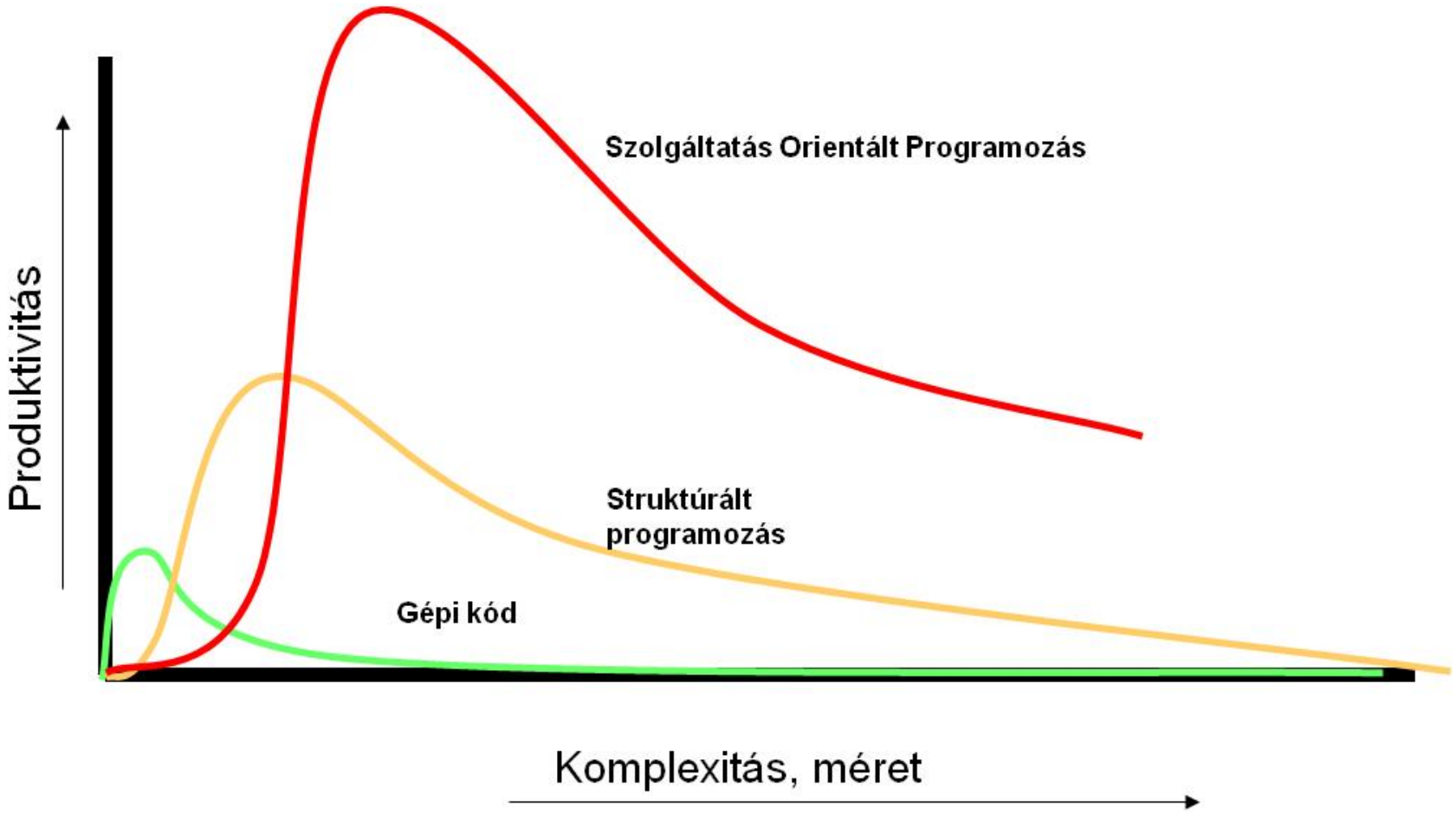
- *Egyszerű események feldolgozása*: ez esetben egy atomi esemény beérkezése indít el egy feldolgozási folyamatot. Egy példa erre a különböző felhasználói interakciókat támogató keretrendszerek eseménykezelése (pl. SWING JSF, stb.)
- *Eseményfolyamok feldolgozása (Event Stream Processing – ESP)*: ezesetben eseményfolyamokat szűrnek egyszerű feltételek alapján, és az érdekes eseményekre feliratkozott vevőknek küldik ki a szűrt eseményeket. Példa lehet erre, amikor a naplózásnál csak adott szintet elérő eseményeket továbbítunk.
- *Komplex esemény feldolgozás (Complex Event Processing - CEP):* ez esetben a valós idejű eseményfolyamon értékelünk ki olyan szabályokat, melyek figyelembe vehetik az események sorrendiségét, bekövetkeztét, számosságát és számos egyéb tulajdonságát. Erre a későbbiekben majd látunk példát a Drools-szal kapcsolatban.



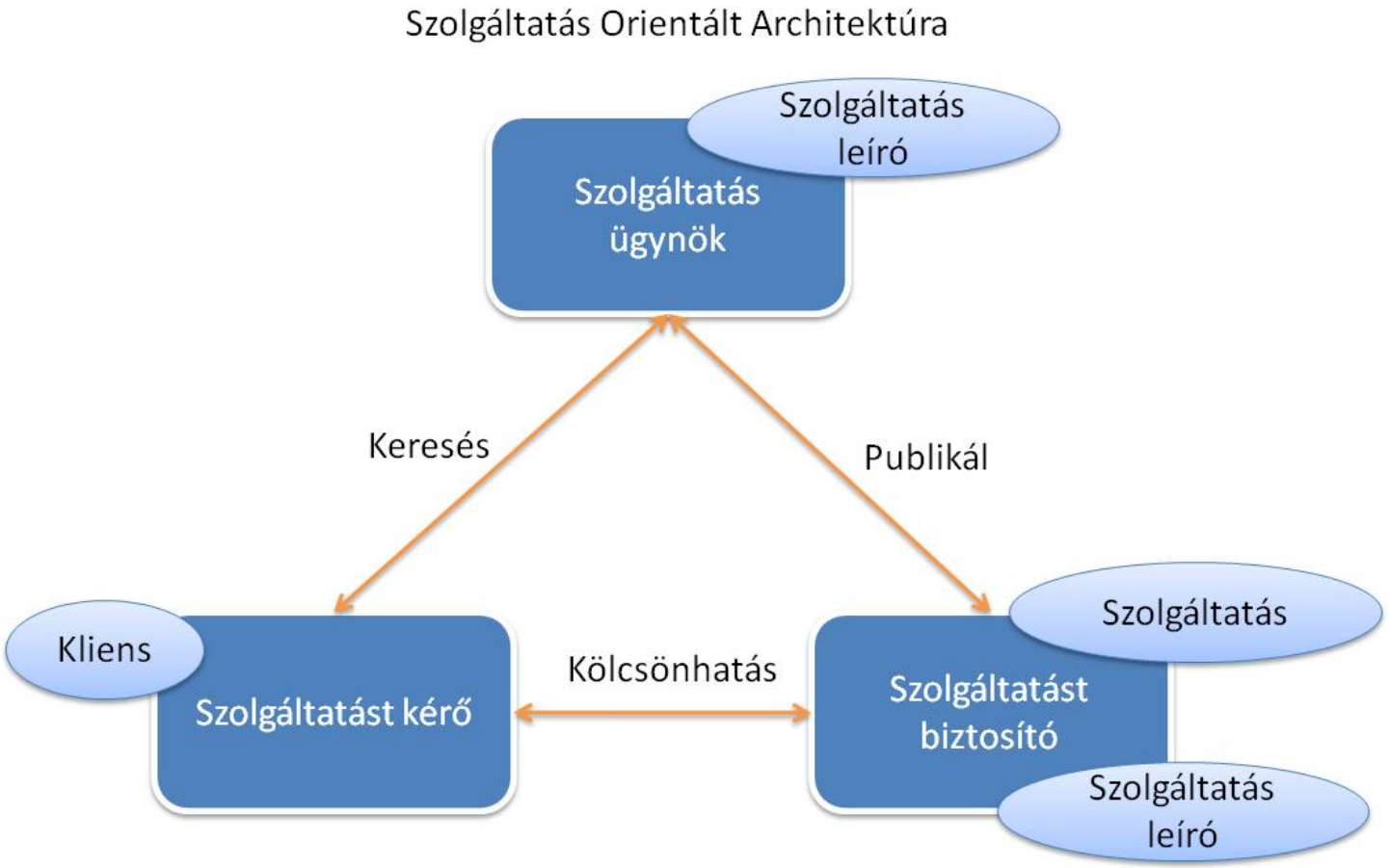
SBA rendszerek

- *Klaszter alapú (Cluster):* a feldolgozásra koncentrál, az adattárolásra különböző tervezési minták vannak, a tipikusan olvasás jellegű alkalmazásoknál igen jó skálázódást tud elérni a CAP kompromisszumai nélkül. Le tudja kezelni az írási ütközést is. Példák erre a különböző alkalmazásszerverek által nyújtott klaszterezési lehetőségek (pl. JBoss Cluster)
- *Felhő alapú (Cloud):* a CAP paradigma mentén a skálázható adattárolást is biztosítja, tipikusan a konzisztencia rovására. Példa erre a Google App Engine.
- *Rács alapú (Grid):* főleg a feldologzásra koncentrál, nincs igazán írási ütközés, így a feldolgozás tetszőlegesen párhuzamosítható. Ezt leginkább munkafolyamatok (**Job**) formájában szokták megtenni.
- *Elosztott objektum alapú (Distributed Objects):* az eloszott objektumok gyakran a fenti architektúrák alapjait képezik. Ekkor egy-egy objektum vagy replikánsa (**replikált objektum** – replicated object) megjelenik azokon a helyeken, ahol használják, és a háttérben egy keretrendszer (köztes réteg) gondoskodik arról, hogy konzisztens maradjon, vagy pedig egy helyen van tárolva (**élő objektum** – live object), és azokon a helyszíneken, ahol szükséges megfelelő proxy objektumok képviselik az objektumot. A megosztott objektumok gyakran **objektumterekbe (object spaces)** vannak szervezve, melyet valamilyen köztes réteg tart karban, virtualizál (pl. Java Spaces).

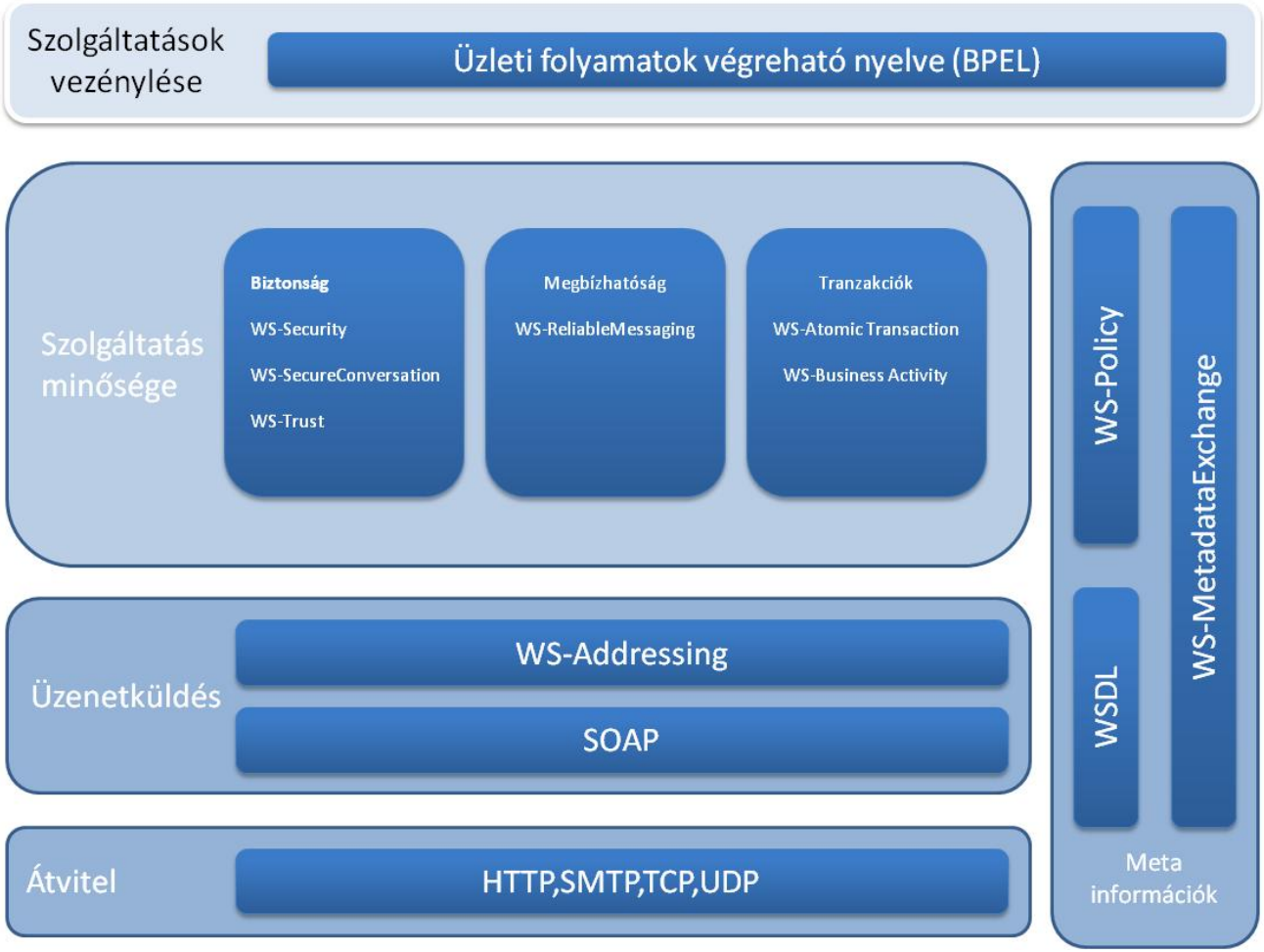
SOA koncepció



SOA háromszög



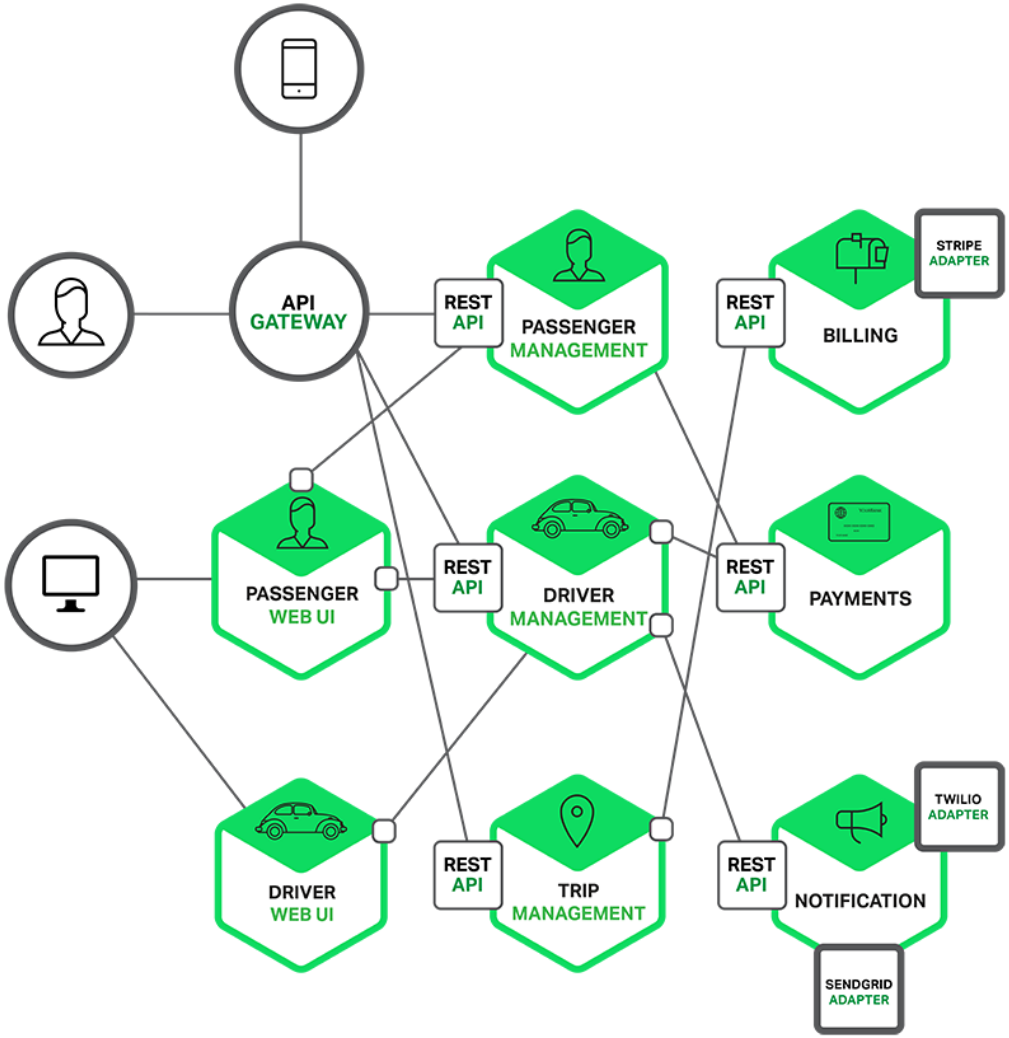
WS világ


Szolgáltatások vezénylése
Üzleti folyamatok végrehajtó nyelve (BPEL)
Szolgáltatás minősége
Biztonság
WS-Security
WS-SecureConversation
WS-Trust
Megbízhatóság
WS-ReliableMessaging
Tranzakciók
WS-Atomic Transaction
WS-Business Activity
Üzenetküldés
WS-Addressing
SOAP
Átvitel
HTTP,SMTP,TCP,UDP
WS-Policy
WSDL
WS-MetadataExchange
Meta információk

Mikroszolgáltatások

- Írjunk nagy alkalmazásokat: írjunk kicsiket (egy nagy alkalmazás kicsikből álljon)
- Egy architektúrlális típus:
  - Egy alkalmazás több önnáló könnyűsúlyú processz
  - Egymással HTTP-n kommunikál
  - Automatizált telepítés
  - Központosított menedzsment

Mikroszolgáltatások



Összefoglaló

- Bevezető
- Felmerülő problémák
- Alkalmazás architektúrák
- Elosztott rendszer architektúrák
- SOA koncepció
- Virtualizációs szintek
- Összefoglaló

A következő előadás tartalma

- Kontextus
- Biztonsági kontextus
- Tranzakciós kontextus
- Perzisztencia kontextus