Eötvös Loránd Tudományegyetem
Informatikai Kar

Komponens alapú szoftverfejlesztés

10. előadás

Elosztott alkalmazások architektúrái

Giachetta Roberto

groberto@inf.elte.hu
http://people.inf.elte.hu/groberto

Elosztott alkalmazások architektúrái

Elosztott rendszerek

- *Elosztott rendszerek* (*distributed systems*) olyan szoftverrendszerek, amelyek komponensei a hálózaton fizikailag elkülönítve futnak, és egymással kommunikálnak
  - az egyes komponensek a végrehajtáshoz külön processzort és memóriát használnak (szemben a párhuzamos rendszerekkel, amelyek megosztott memóriát használnak)
  - a komponensek végrehajtási helyei a csomópontok (*node*), amelyek megosztják egymással az erőforrásokat
  - fontos tényezői a párhuzamos feladatvégzés és a skálázhatóság (a szoftver áteresztőképességének növelése)
  - a kommunikáció általában kérés-válasz alapú

Elosztott alkalmazások architektúrái

Elosztott rendszerek

- Elosztott rendszerekkel szemben három elvárás lehetséges:
  1. *következetesség (consistency)*: minden csomópont ugyanazon adatokat látja ugyanabban az időben
  2. *rendelkezésre állás (availability)*: a rendszernek minden időben válaszolnia kell az összes beérkező kérésre
  3. *részleges hibatűrés (partition tolerance)*: a rendszer tovább üzemel akkor is, ha a rendszer egyes részei (partíciói) leállnak, vagy megszakad velük a kapcsolat
- A *CAP tétel* szerint egy elosztott rendszer ebből csak kettőt tud teljesíteni, ennek megfelelően kompromisszumokat kell kötni a képességek terén

Elosztott alkalmazások architektúrái

Kommunikáció

- Elosztott rendszerekben a kommunikáció történhet
  - *eljáráshívási alapon* (*procedural*): a kliens kezdeményezi a szolgáltatás végrehajtását és bevárja annak eredményét
    - a kliens mindig ismeri a szolgáltatás helyét és hívásnak módját
    - kliens oldalon megjelenik a szolgáltatás interfésze, és egyben egy megvalósítása, amelynek feladata a kérés csomagolása (*marshalling*), és továbbítása
    - a szerver fogadja és kicsomagolja (*unmarshalling*) az üzenetet, majd feldolgozza, és ugyanilyen módon visszaküldi a választ
    - általában *távoli eljáráshívás*sal (*remote procedure call*, *RPC*) valósul meg

- pl. webszolgáltatás, CORBA, Remote Method Invocation (RMI, Java)

- *üzenet-alapon* (*message-based*): a kliens egy üzenetet továbbít, és egy válaszüzenetben várja az eredményeket
  - nem igényli a közvetlen kommunikációt, az üzenet továbbítható
  - nem igényli az azonnali feldolgozást, lehetővé teszi az üzenetek sorba állítását (*message queuing*)
    - az üzenetsor (*message queue*) általában kategóriák szerint csoportosítja az üzeneteeket, így a feliratkozók válogathatnak közülük (*publish-and-subscribe*)
    - pl.: Apache ActiveMQ, MSMQ, RabbitMQ

Elosztott alkalmazások architektúrái

Skálázhatóság

- Hardverek vertikálisan és horizontálisan skálázhatóak
- Szoftverek skálázhatósága 3 irányba történhet:
  - *klónozás*: az alkalmazás több példányát futtatjuk, amelyek így megosztják a feladatokat, ám mindegyik egy közös adatforrást használ
    - terheléselosztás (*load balancing*) szükséges a feladatok irányításához
    - biztosítani kell a közös adatok megfelelő elérését (pl. termelő-fogyasztó), gyorsítótárak használatát a teljesítménynövelés érdekében
    - pl. kliens-szerver architektúra

Elosztott alkalmazások architektúrái

Skálázhatóság

- *funkcionális dekompozíció*: az egyes szolgáltatásokat más alkalmazások biztosítják
  - emiatt csak az adatok egy részét kezelik, ám egy feladatkörön belül az összes adatot látják
  - pl.: mikroszolgáltatások, csövek és szűrők architektúra
- *adatparticionálás*: az alkalmazás több példánya fut, ám mindegyik csak egy részét dolgozza fel az adatoknak
  - hatékony adathozzáférést tesz lehetővé (mindenki a lokális adatokon dolgozik), ehhez azonban megfelelő adatelosztási stratégia szükséges
  - pl.: peer-to-peer, megosztásmentes architektúra

Elosztott alkalmazások architektúrái

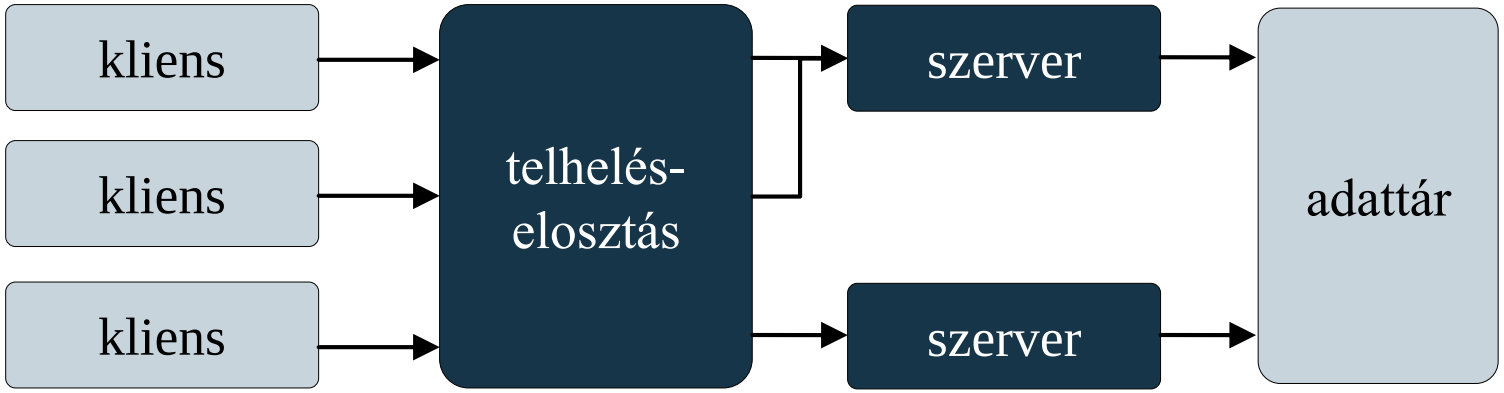
A kliens-szerver architektúra

- A *kliens-szerver* (*client-server*) architektúra két szerepet különböztet meg:
  - a kliens biztosítja a felhasználóval történő interakciót
    - tartalmazhatja csupán a felületet (*vékony kliens*), illetve az egyszerű szolgáltatások megvalósítását (*vastag kliens*)
    - nem ismeri a többi klienst, nem lép vele kapcsolatba
  - a szerver biztosítja a szolgáltatásokat és az adatok tárolását
    - nem ismeri a hozzá csatlakozó klienseket, azok állapotát
    - a klienseknek ismernie kell a szerver elérhetőségét
    - általában erősebb hardver-erőforrásokra támaszkodik (*centralized computing*)

Elosztott alkalmazások architektúrái

A kliens-szerver architektúra terhelésmegosztással

- A kliens-szerver architektúrában minden szerver ugyanazt a funkcionalitást biztosítja, így a skálázhatóság terheléselosztás (*load balancing*) segítségével történik
  - az egyes kliensekhez rendelünk különböző szervereket
  - minden kliens a terheléselosztóhoz csatlakozik, amely ismeri az összes szerver elérhetőségét



Elosztott alkalmazások architektúrái

A kliens-szerver architektúra terhelésmegosztással

- A terhelésmegosztás történhet:
  - ütemezett módon, egy címlistából választva (*round-robin DNS*), vagy delegált módon (*DNS delegation*)
  - kliens oldalon véletlenszerűsítve (a címek átadásával a kliensnek), vagy szerver oldalon (így a kliens nem ismeri a tényleges címeket)
    - a szerver oldali elosztás esetén garantálni kell a magas rendelkezésre állást

- Amennyiben a szerver oldal kezeli a kapcsolat állapotát:
  - a klienst mindig egy dedikált szerverhez kell kapcsolnunk, vagy
  - meg kell osztani az állapotot a szerverek között (*shared database*)

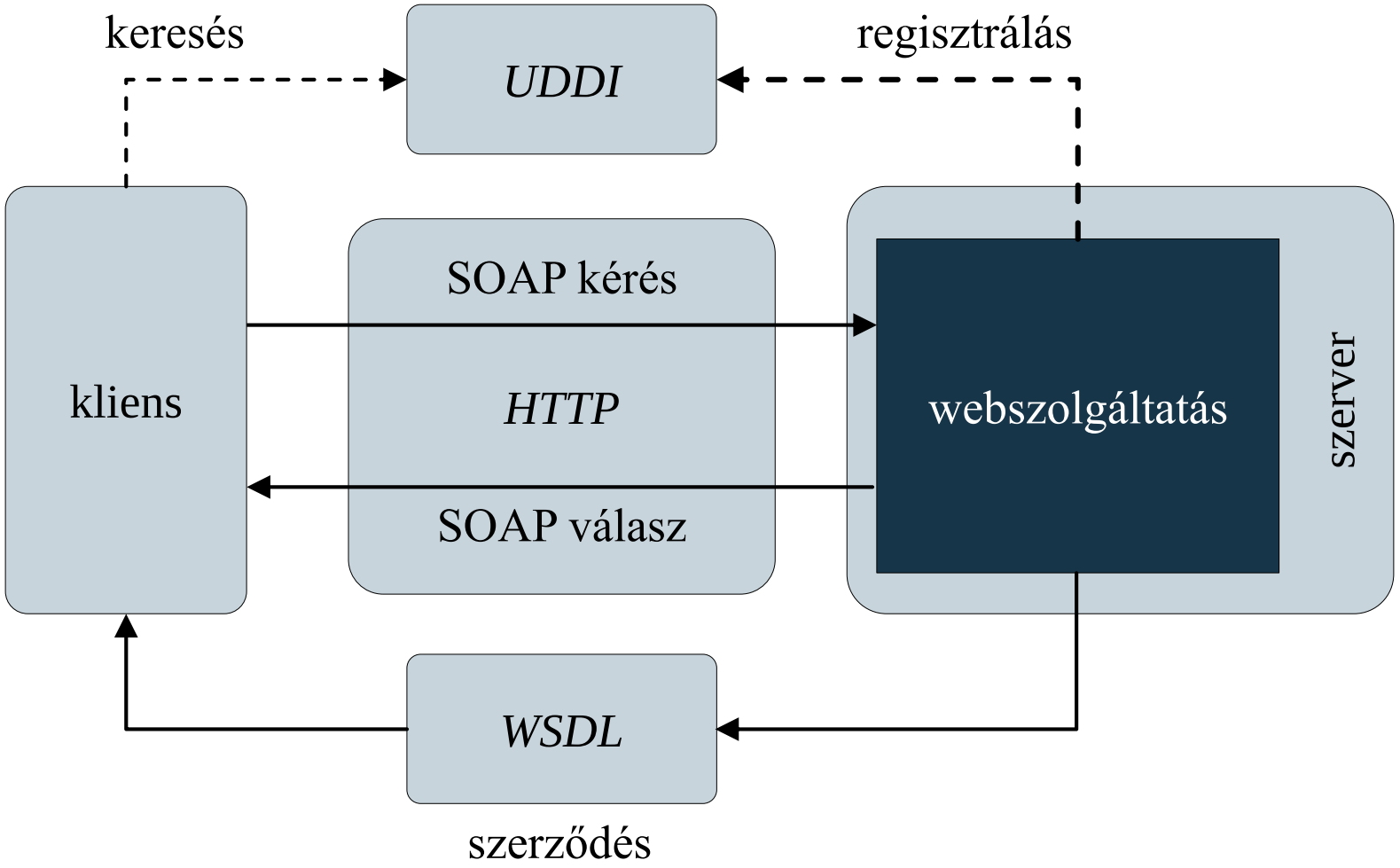
Elosztott alkalmazások architektúrái

A webszolgáltatás

- A *webszolgáltatás* (*Web Service*, *WS*) egy egységes szabványt ad elosztott hálózati alkalmazások közötti kommunikációra
  - a szolgáltatásokat úgynevezett *webmetódusok* (*web method/operation*) formájában érhetőek el távoli eljáráshívás segítségével futtat HTTP protokollon keresztül
  - kommunikációs módszere a *SOAP* (*Simple Object Access Protocol*), egy XML alapú adatközlési csatorna
  - a webszolgáltatáshoz tartozó szerződést *WSDL* (*Web Services Description Language*) formában adhatjuk meg
  - szolgáltatások nyilvántartására szolgál a *UDDI* (*Universal Description Discovery and Integration*)

Elosztott alkalmazások architektúrái

A webszolgáltatás



Elosztott alkalmazások architektúrái

A REST architektúra

- A *REST* (*REpresentational State Transfer*) egy szoftverarchitektúra elsősorban elosztott hipermédia rendszerek számára
  - elsősorban HTTP alapon kommunikál alapvető HTTP utasítások (**`GET`**, **`POST`**, **`PUT`**, **`DELETE`**, …) segítségével, webes erőforrások (*web resource*) elérése céljából
  - egy egységes, egyszerű interfészt vár el, amely biztosítja
    - az erőforrások egyértelmű azonosítását (URL)
    - az erőforrások leírását metaadatok segítségével
    - az erőforrások manipulációjának lehetőségét
    - a komponensek közötti átlátható kommunikációt
  - a támogató szoftverek a *RESTful alkalmazások*

Elosztott alkalmazások architektúrái

A REST architektúra

- A REST alapú rendszereknek feltételei:
  - *kliens-szerver architektúra*, mozgatható komponensekkel
  - *állapotmentes, környezetfüggetlen kommunikáció*: a kliens állapota nem tárolódik a szerveren az üzenet között, így minden üzenetnek kell tartalmaznia a feldolgozásához szükséges valamennyi információt
  - *gyorsítótárazhatóság*: a kliens eltárolhatja a feldolgozás eredményét (amennyiben az üzenet erre lehetőséget ad)
  - *rétegződés*: közvetítők helyezhetők a kliens és a szerver közé (gyorsítótárak, terheléselosztók), amelyek a kliensek számára nem felfedhetőek

Elosztott alkalmazások architektúrái

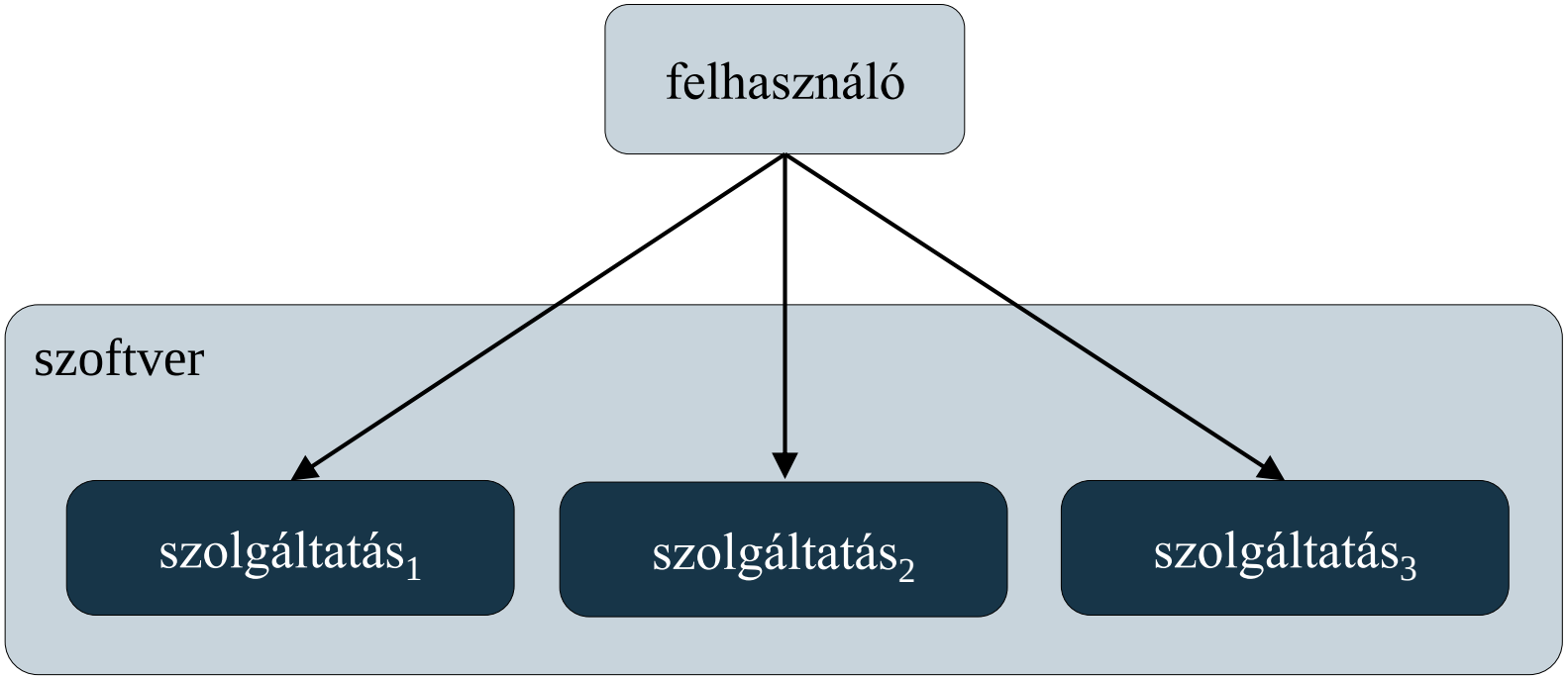
Mikroszolgáltatások architektúra

- A szoftver szolgáltatási szempontjából felbontva a monolitikus felépítést kapjuk a *mikroszolgáltatások architektúrát* (*microservices architecture*)
  - a szolgáltatásokat (vagy szolgáltatások csoportját) külön programegységek biztosítják, amelyek egymástól függetlenül hajtják végre tevékenységeiket
  - ennek megfelelően a szolgáltatások külön fejleszthetőek, tesztelhetőek, és könnyen telepíthető új szolgáltatás
  - a szolgáltatások a megoldás érdekében korlátozottan kommunikálhatnak egymással (szabványos csatornán), esetleg közös adatforrásokkal
  - a szolgáltatások rendelkezhetnek közös hozzáférési ponttal

Elosztott alkalmazások architektúrái

Mikroszolgáltatások architektúra

- a szoftver könnyen skálázható, mivel az egyes szolgáltatásokat külön platform láthatja el



Elosztott alkalmazások architektúrái

Mikroszolgáltatások architektúra

- Előnyei:
  - a komponensek jól leválaszthatóak, külön fejleszthetők, telepíthetőek
  - a szolgáltatások mentén jól skálázható
- Hátrányai:
  - a szolgáltatások közötti felelősségátadás nagyban megnehezedik
  - a szolgáltatások közötti kommunikáció korlátozott, problémákhoz vezethet (és magasabb fokú hálózati igénybevételhez)

Elosztott alkalmazások architektúrái

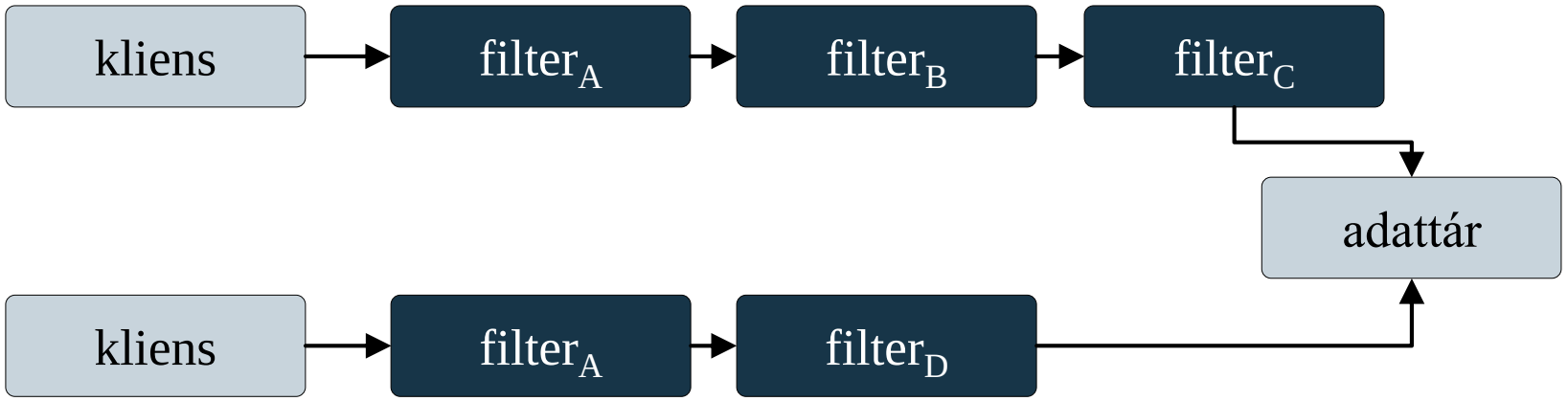
Megosztásmentes architektúra

- A *megosztásmentes architektúrá*ban (*shared nothing architecture*, SN) az egyes csomópontok egymástól függetlenek, és önfenntartóak
  - nincs központi erőforrás, az adatok elosztásra, vagy lemásolásra kerülnek az egyes csomópontok között (*sharding*), így lineárisan skálázható, és nincs meghibásodási pontja
  - a módosítások is a helyi csomópontra íródnak vissza, ezért nem támogatja a központosított adatkezelést
  - általában biztosítja az egyes csomópontok leállásmentes frissítését
  - a rendszer része lehet egy vezérlő, amely megfelelően elirányítja, elosztja az adatokat, ekkor azonban lehetőséget kell adni más csomópontoknak is a vezérlő szerepének átvételét

Elosztott alkalmazások architektúrái

Csövek és szűrők

- Munkafolyamatok elosztott végrehajtását teszi lehetővé a *csövek és szűrők* (*pipes and filters*) architektúráját
  - akkor alkalmazható, ha a munkafolyamat felbontható egymástól független lépésekre, amelyet az egyes komponensek (*filter*) végrehajtanak, az eredményt pedig továbbítják a megfelelő csatornán (*pipe*)



Elosztott alkalmazások architektúrái

Csövek és szűrők

- Előnyei:
  - a részfeladatok egymástól függetlenül skálázhatóak
  - egy szűrő példány kiesése esetén a feladat átirányítható más példányhoz
- Hátrányai:
  - nem alkalmazható, ha a folyamat nem lineáris, vagy a részfeladatok között összefüggések vannak
  - elveszett, vagy sérült üzenet esetén a teljes folyamat leállhat
  - amennyiben a részfeladat elvégzését nem megfelelő időpillanatban jelzik, duplikált munkavégzés alakulhat ki

Elosztott alkalmazások architektúrái

A MapReduce architektúra

- A *MapReduce* programozási modell a csövek és szűrök architektúra egy egyszerű megvalósítása, amely két részre bontja a feldolgozás
  - a Map lépés a részadatok feldolgozását végzi, a Reduce lépést a különböző Map által adott eredmények összefuttatását végzi
  - számos Map, illetve Reduce lépés futhat párhuzamosan, amelyek különböző adatokkal dolgoznak
  - az adatok kulcs/érték párok formájában kezeli, amelyek beazonosítják a megfelelő Map és Reduce folyamatokat, amelyekhez az adatokat el kell juttatni
  - a végrehajtás nagyban múlik a két folyamat párhuzamosításának kihasználásán

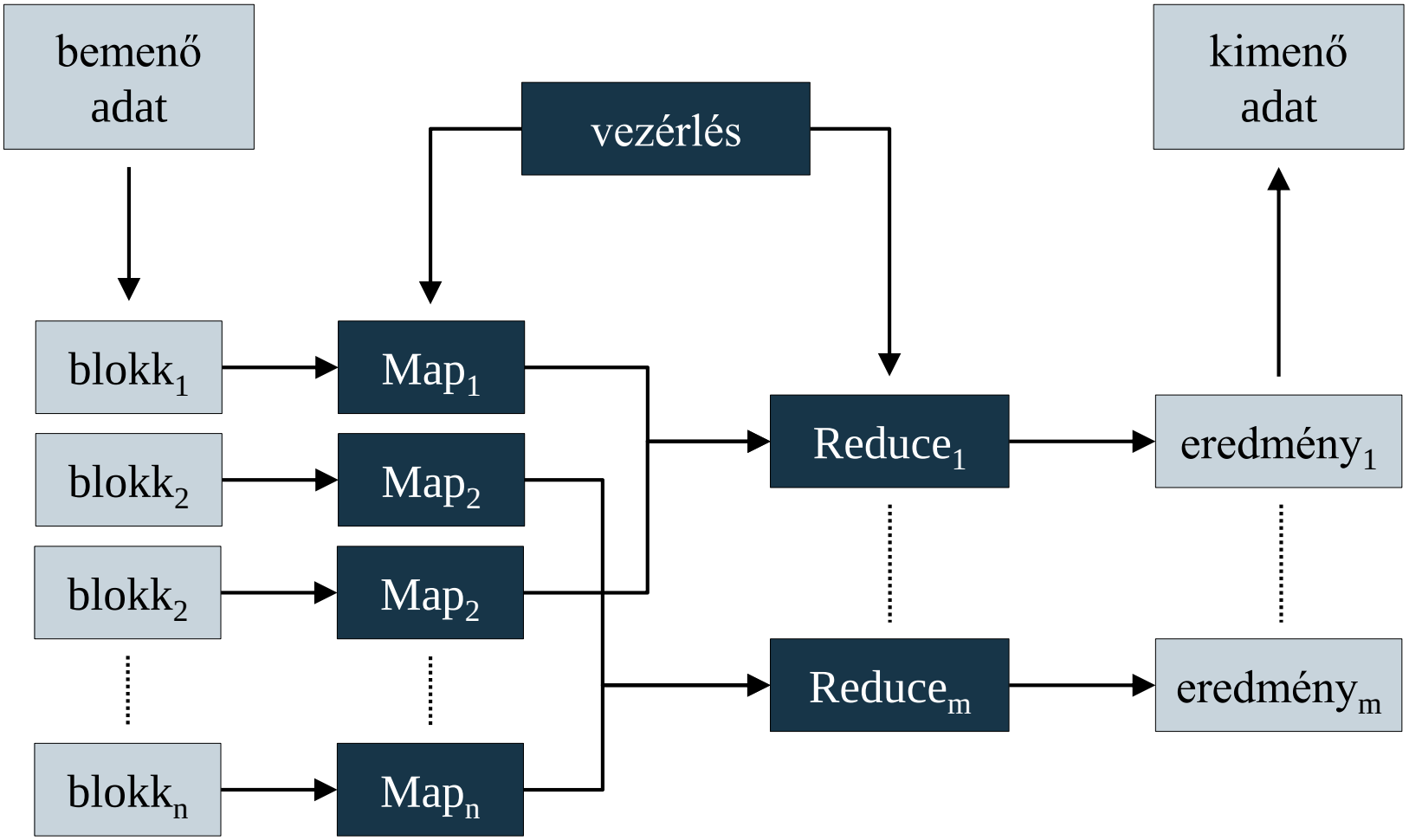
Elosztott alkalmazások architektúrái

A MapReduce architektúra

- Egy teljes munkafolyamat a MapReduce modellben 5 lépésből áll:
  1. *Prepare*: felosztja a beérkező adatokat azonos méretű blokkokra, mindegyikhez egy kulcsot rendel
  2. *Map*: a kapott kulcs/érték párokat párhuzamosan transzformája (szűri, rendezi, …) köztes eredményekké
  3. *Shuffle*: a köztes eredményeket átrendezi, csoportosítja kulcs szerint és újabb kulccsal látja el
  4. *Reduce*: a kapott kulcs/érték párokat párhuzamosan kiértékeli
  5. *Produce*: az eredményeket begyűjti és összesíti

Elosztott alkalmazások architektúrái

A MapReduce architektúra



Elosztott alkalmazások architektúrái

A MapReduce architektúra

- Pl. szeretnénk megszámolni dokumentumok gyűjteményében minden szónak az előfordulási számát:
  1. felosztjuk a dokumentumokat, minden blokk egy dokumentumot kap (a kulcs a dokumentum neve)
  2. a dokumentumban minden megtalált szóra egy 1-es értéket adunk (a kulcs a szó, az érték 1), ez a *Map* lépés
  3. csoportosítunk a szó szerint, minden szóhoz egészek egy sorozata tartozik
  4. összeadjuk a sorozat értékeit, így megkapjuk minden szóra az összes előfordulás számát, ez a *Reduce* lépés
  5. az eredményeket összesítjük (szavanként)

Elosztott alkalmazások architektúrái

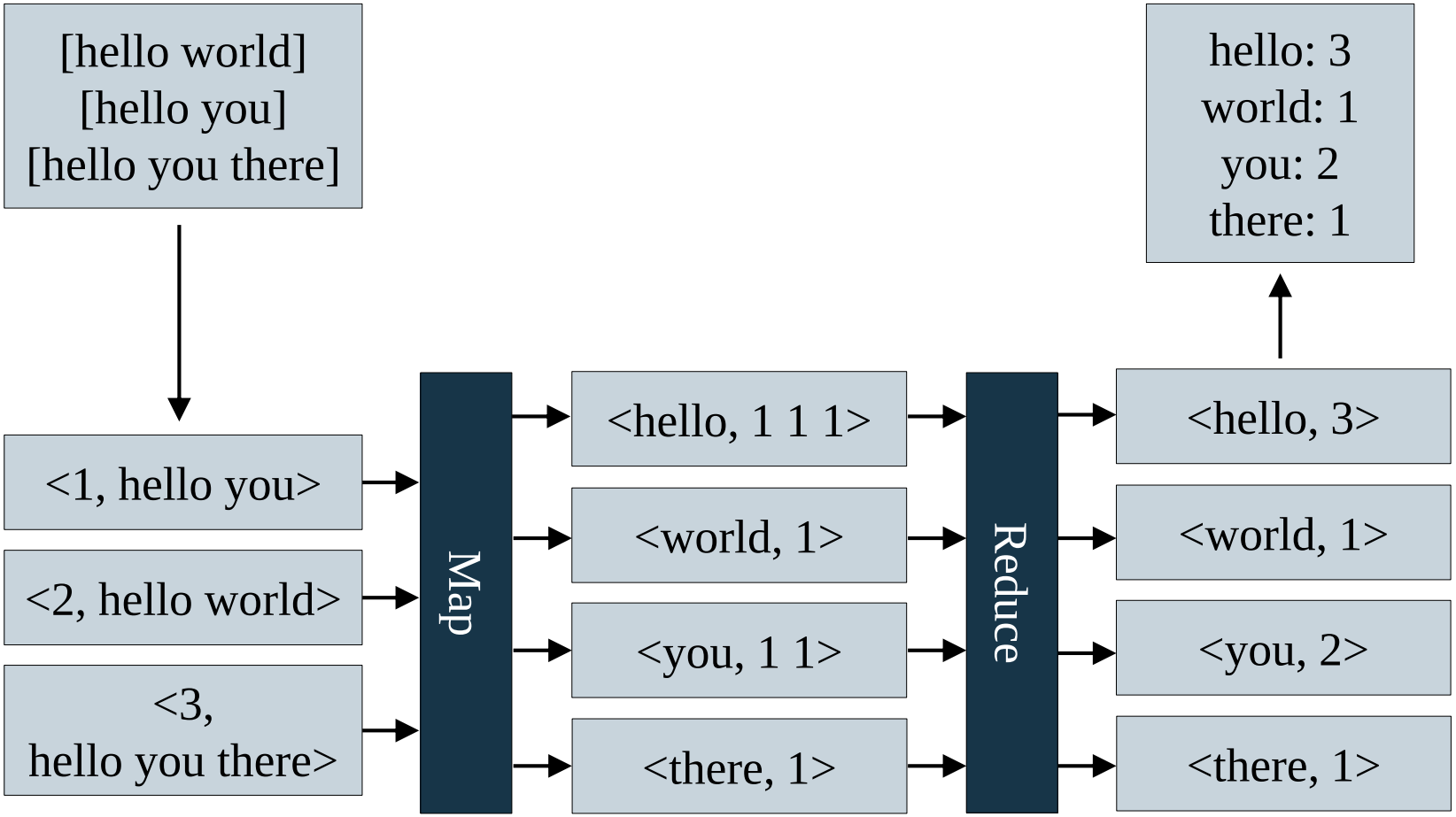
A MapReduce architektúra

```
Map(String key, String value):
   // key: dokumentum neve
   // value: dokumentum tartalma
   foreach (String word in value):
      EmitIntermediate(w, "1");
      // a kulcs a szó, az érték 1 lesz

Reduce(String key, Enumerable<String> values):
   // key: a szó
   // values: a szóra kapott értékek sorozata
   int result = 0;
   foreach (String v in values):
      result += AsInt(v);
   Emit(AsString(result)); // szövegesen adjuk meg
```

Elosztott alkalmazások architektúrái

A MapReduce architektúra



Elosztott alkalmazások architektúrái

Hadoop

- Az *Apache Hadoop* egy, a megosztás-mentes architektúrára és a MapReduce programozási modellre épülő szoftver keretrendszer
  - 3 fő modulból alkotja:
    - *Hadoop MapReduce*: adatfeldolgozó
    - *Hadoop Distributed File System* (*HDFS*): elosztott fájlrendszer, amely felel az adatok elosztásáért az egyes node-ok között
    - *Hadoop YARN*: erőforrás kezelő és folyamat ütemező
  - Java-ban íródott, de alkalmas bármilyen nyelvű, MapReduce alapú alkalmazás futtatására (*Hadoop Streaming*, *REST API*)
  - számos rendszer alapjául szolgál (*Pig*, *Hive*, *HBase*, *Spark*, …)

Elosztott alkalmazások architektúrái

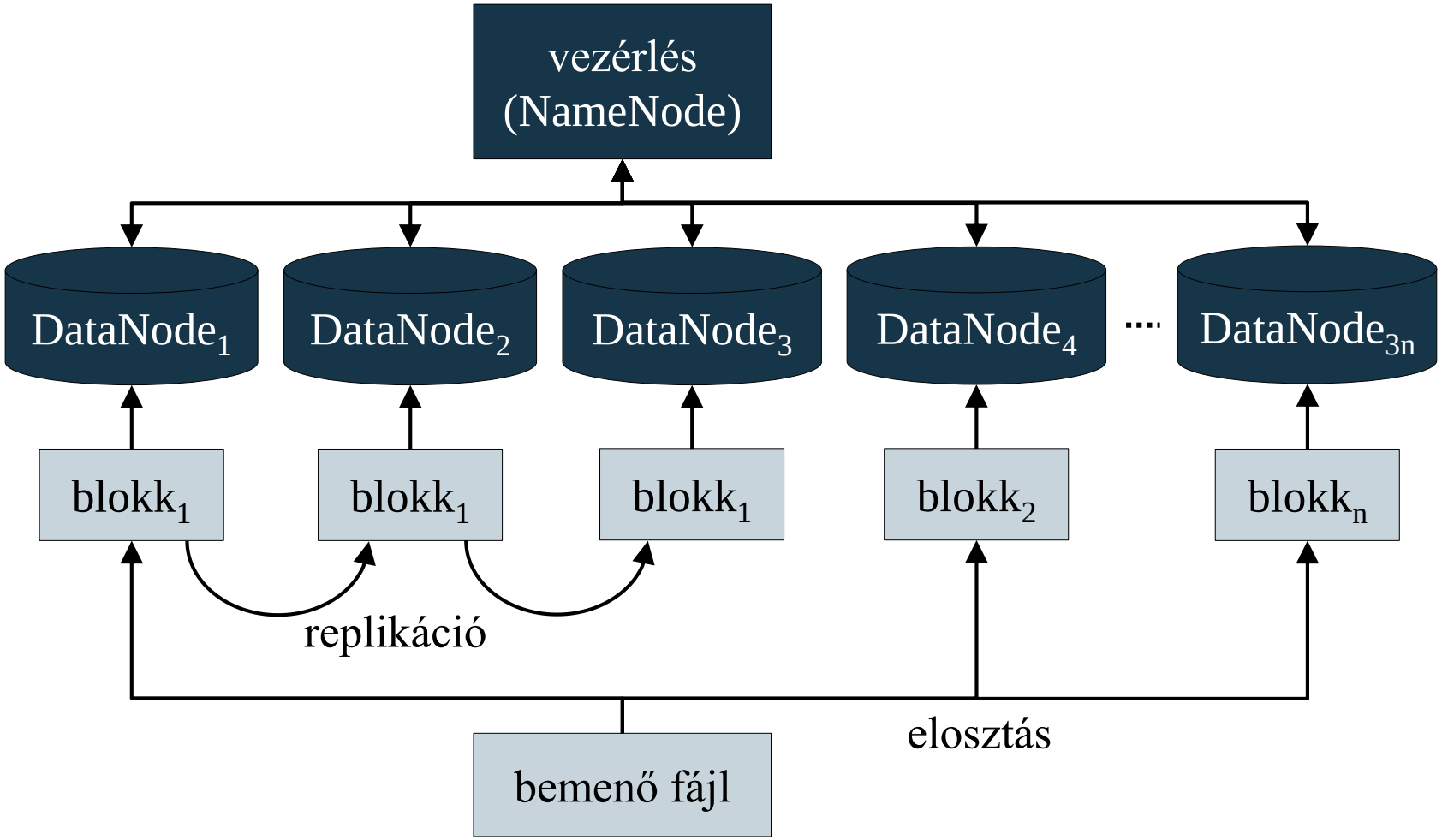
Hadoop

- A Hadoop nagyon jól skálázható, akár több ezer node alkotta rendszert is felépíthetünk

- A HDFS fájlrendszer célja a Hadoop architektúra kiszolgálása, ügyelve a rendelkezésre állásra
  - hatékonyan kezeli a nagy, homogén szerkezetű fájlokat, amelyeket azonos méretű blokkokra bont (pl. 256 MB)
  - olvasásra optimalizál (*write once, read many times*)
  - képes kezelni az egyes gépek kiesését, az egyes fájlrendszer csomópontok (*DataNode*) folyamatosan jelzik jelenlétüket
  - replikálja a tartalmat (alapértelmezetten 3 példány), így kerüli el az adatvesztést

Elosztott alkalmazások architektúrái

Hadoop



Elosztott alkalmazások architektúrái

A P2P architektúra

- A peer-to-peer (P2P) egy olyan decentralizált architektúra, amelyben a rendszert azonos alkalmazások alkotják, amelyek elosztják a feladatokat egymás között
  - minden csomópont külön felel a feldolgozásért, az adattárolásért, a kommunikációért (beleértve a többi csomópont elérését)
  - a központi adattárolással szemben az információkat elosztottan kezeli a hálózat tagjai között
  - a tényleges hálózatot legtöbb esetben elfedi egy virtuális *fedőhálózat* (*overlay network*), amelynek feladata a csomópontok kezelése (indexelés, felfedezés, irányítás)
    - független a fizikai hálózattól, az ismeretségi reláció definiálja

Elosztott alkalmazások architektúrái

A P2P architektúra

- A P2P hálózat lehet:
  - *strukturálatlan* (pl. Gnutella, Gossip), amelyben nincs rögzített topológia, a csomópontok véletlenszerűen kapcsolódnak egymáshoz
    - minden csomópont ugyanazt a szerepet tölti be, így nagyon rugalmasan módosítható a tényleges felépítés
    - a kéréseket minden (adott pontból elérhető) összes csomópont megkapja, feldolgozza, esetleg továbbítja, ami lassítja a működést (*message flooding*), és nem garantálja, hogy a kérés sikeres lesz
  - *strukturált* (pl. Kad), amelyben a csomópont rögzített topológia szerint, általában elosztott hash-tábla alapján épülnek fel

Elosztott alkalmazások architektúrái

A P2P architektúra

- Előnyei:
  - nagyfokú rendelkezésre állást és hibatűrést tud biztosítani
  - könnyen skálázható, hatékonyan használja a hálózati kapacitást
- Hátrányai:
  - az egyes csomópontoknak minden funkcionalitást magukban kell tartalmaznia
  - nem biztosít központosított adatkezelést
  - jelentős többletet okozhat feldolgozásban és kommunikációban (pl. hálózat tagjainak lokalizálása)

Elosztott alkalmazások architektúrái

A P2P architektúra

- A tisztán peer-to-peer architektúra mellett felépíthetőek hibrid modellek, amelyben a kliens és szerver szerepe megkülönböztethető
  - a szerver egy központi regisztrációja a klienseknek, amely nyilvántartja a klienseket, illetve a kliensek által nyújtott szolgáltatásokat (pl. Napster)
  - több szerver is rendelkezésre állhat, amelyek elosztják a feladatokat, így egy kliens megtalálása több szerver közbeiktatásával történik (pl. Skype)
    - kliens is válthat szerverré
  - a kommunikáció felgyorsítása érdekében a rendszer több forrásból is igénybe vehet részadatokat (pl. BitTorrent)

Elosztott alkalmazások architektúrái

Elosztott hash táblák

- Az *elosztott hash-tábla* (*distributed hash-table*, *DHT*) egy olyan hálózatelosztási modell, ahol a kérések irányítása hash-tábla alapon működik
  - az adatok kulcs/érték párként tároltak, ahol a kulcs az érték hash-függvény által generált értéke (pl. *SHA-1*, amely 160 bites kulcsokat állít elő)
  - a kulcsok alapján azonosítják a csomópontot, ahol az adatot tárolják, ez ideális esetben $O(1)$ lépést jelent
  - a hatékonyság érdekében a csomópontok csak a hálózat egy részével állnak kapcsolatban (*routing table*)
    - általában N csomópont esetén $\log N$ szomszéd ismert, így a kérés célba érkezése maximum $O(\log N)$ lépésben megtörténik

Elosztott alkalmazások architektúrái

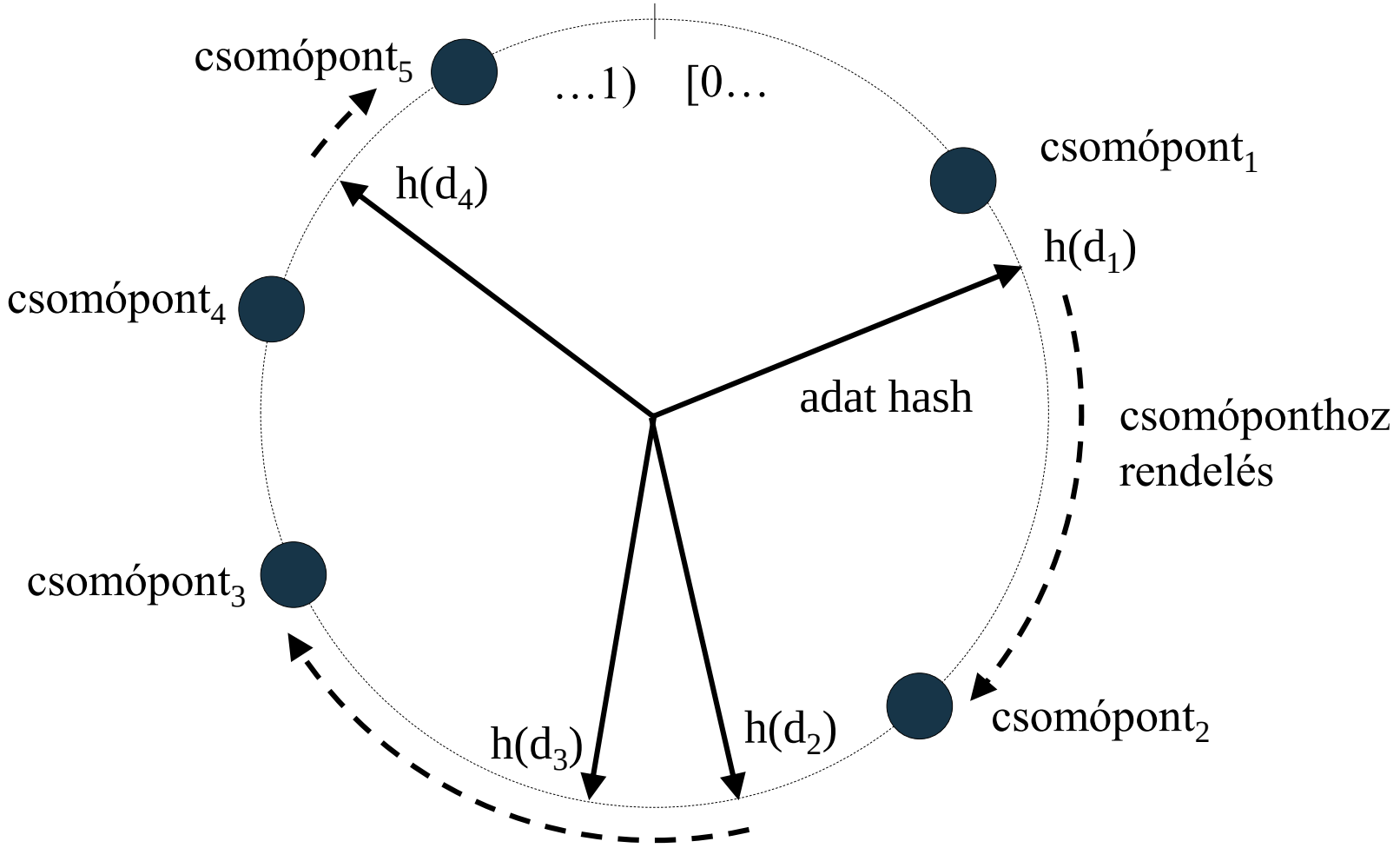
Konzisztens hash-elés

- Elosztott hash táblák esetén ügyelni kell arra, hogy új csomópont felvétele/törlése esetén az adatokat újra el kell osztani (hash-elni)
  - az alapvető hash-függvények (pl. osztómódszer) nem használhatóak, mert az adatok nagyrészét át kell csoportosítani

- A *konzisztens hash*-elés (*consistent hashing*) egy olyan technika, amely minimális adatmozgatást igényel a hálózat változása esetén
  - a kulcsok tere (*keyspace*) egy gyűrű (*ring*), általában a [0 ... 1) intervallumban
  - a csomópontokat egy véletlenszerű hash alapján helyezzük el
  - az adatokat a hash értékük a(z óramutató járása szerinti) rákövetkező csomópontban helyezzük el

Elosztott alkalmazások architektúrái

Konzisztens hash-elés



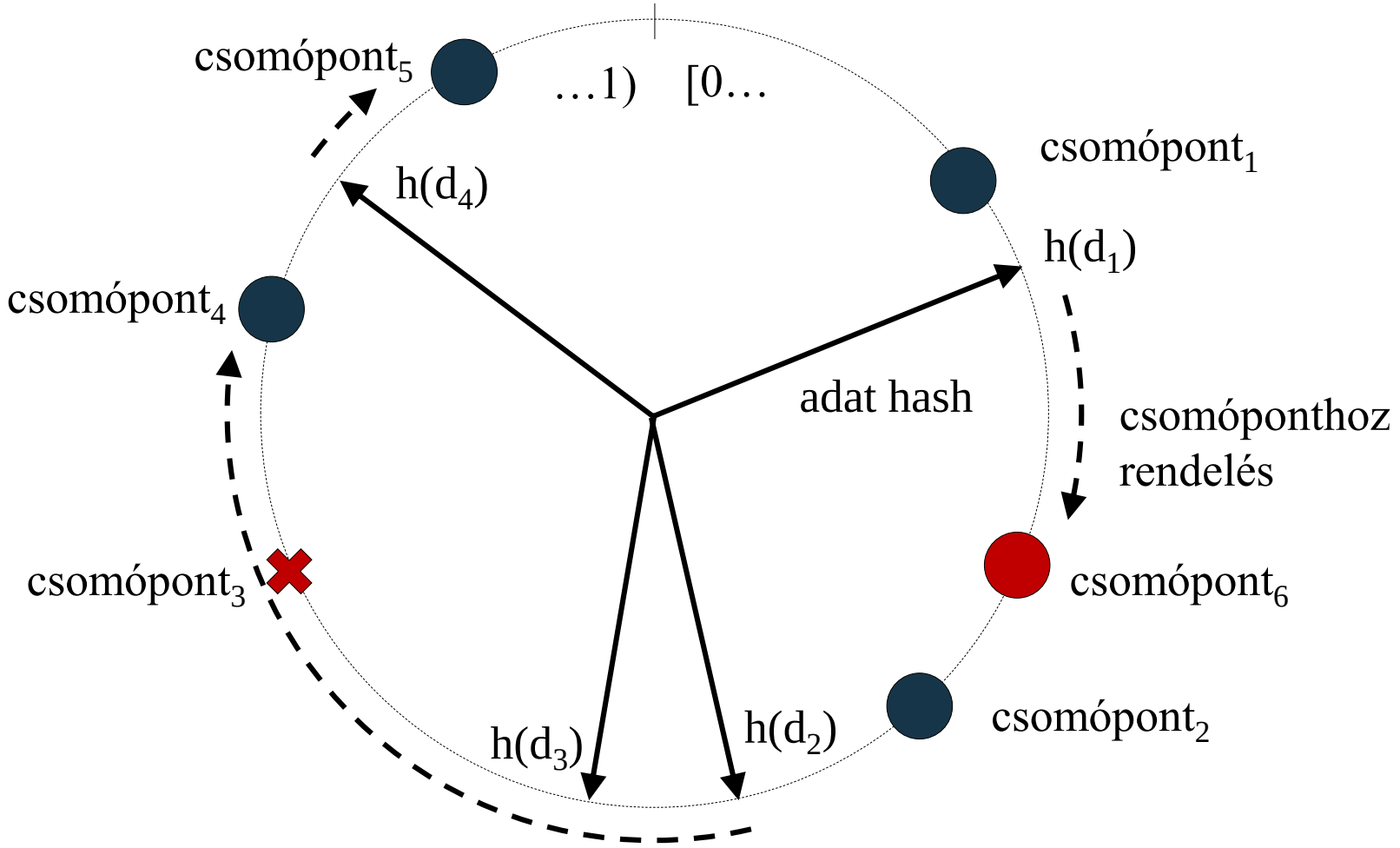
Elosztott alkalmazások architektúrái

Konzisztens hash-elés

- A konzisztens hash-elés során
  - amennyiben új csomópontot adunk a hálózathoz, csak a közötte és az őt megelőző csomópont közötti hash értékű adatokat kell áthelyeznünk az új csomópontra, így minimális a mozgatás
  - amennyiben egy csomópont kiesik, az adatok a rákövetkező csomópontra kerülnek

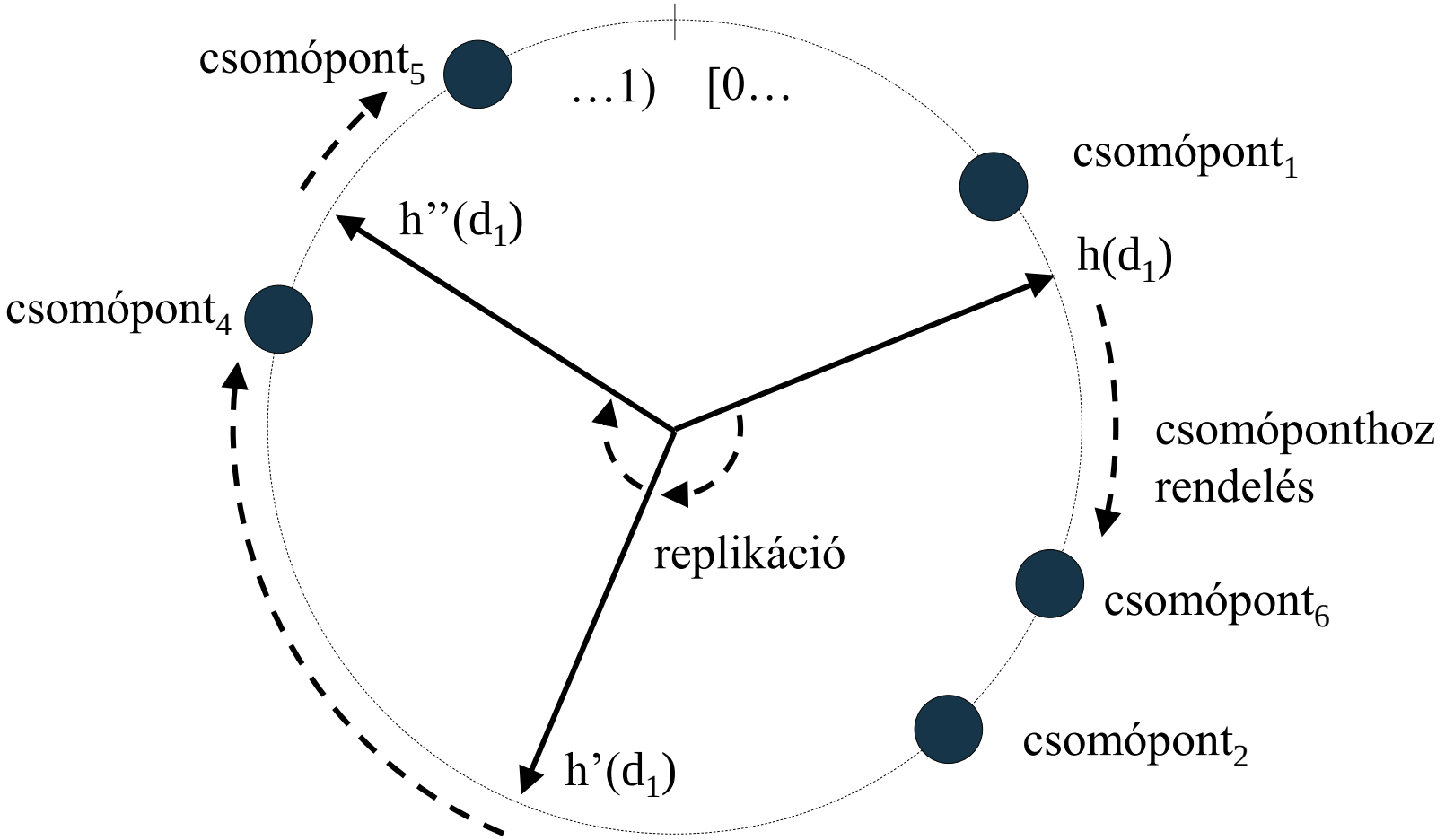
- Amennyiben kellő számú csomópont van a rendszerben, a véletlenszerű kiosztás miatt az adatok egyenletes eloszlásúak lesznek
  - az adatok replikációjával növelhető a hibatűrés, és javítható az adatok eloszlása
  - pl. a hash-érték eltolása rögzített szöggel (amely nagyobb, mint a csomópontok által bezárt szövegek minimuma)

Konzisztens hash-elés



Elosztott alkalmazások architektúrái

Konzisztens hash-elés



Elosztott alkalmazások architektúrái

Dynamo

- Az *Apache Dynamo* egy elosztott hash-táblára épülő *kulcs/érték tároló* (*key-value store*), amely a következetességet áldozza fel a nagyobb rendelkezésre állás és hibatűrés érdekében
  - a rendszer felépítése és az adatok elosztása konzisztens hash-elés segítségével történik, replikáció igénybevételével
    - N fokú replikáció esetén minden csomópont az őt a gyűrűben követő $N - 1$ elérhető csomópontra replikál
  - a csomópontok virtuálisak, és a gyűrűbeli sorrendtől függetlenül vannak fizikai csomópontokon elhelyezve, így biztosítva a hibatűrést
  - a csomópontok változását több lépésben közli a rendszer, véletlenszerű csomópontoknak továbbadva az információt (*gossip protocol*)

Elosztott alkalmazások architektúrái

Dynamo

- az adott csomópontra a műveletek végrehajtási sorrendje változó lehet, így nem garantált a konzisztencia, egyes adatok más állapotban lehetnek a csomópontokon, amely *vektor órák* (*vector clock*) segítségével van nyilvántartva
  - minden adat rendelkezik egy órával (verziószámmal), amely állapotváltás során frissül, és ezzel együtt továbbítja az adatot, így megállapítható a legfrissebb változat
  - az adatok korábbi verziói is a rendszerben maradnak
  - a független változtatások miatti verzióütközés esetén külön kell gondoskodni a feloldásról (pl. legfrissebb módosítás érvényesítése)
- a műveletek (írás és olvasás) akkor tekinthetőek sikeresnek, ha bizonyos számú csomópont teljesíti a kérést (*sloppy quorum*)

Elosztott alkalmazások architektúrái

Aktor-alapú architektúra

- Az *aktor modell* (*actor model*) egy olyan megosztásmentes architektúra, amelyben egymással kommunikáció aktorok végzik a tevékenységeket, amelyek
  - könnyűsúlyú folyamatok, kevés tevékenységgel
    - gyorsan példányosíthatóak, szüneteltethetőek
    - erőforrásaikat a rendszer központilag kezeli
  - direkt módon, aszinkron üzenet alapon kommunikálnak
    - az üzeneteket üzenetsorokon keresztül fogadják
    - egymás fizikai elhelyezkedését nem ismerik
  - elsősorban számításigényes, és kevéssé adatcentrikus feladatok elvégzését tudják biztosítani

Elosztott alkalmazások architektúrái

Orleans

- A *Microsoft Orleans* egy aktor modellt megvalósító rendszer .NET környezetben
  - alapegysége a *Grain*, amelyek aszinkron metódusokkal ellátott objektumok, egyedi kulccsal
    - élettartamát és futtatási helyét a rendszer felülegyeli (*virtuális aktor*)
    - a kliensek a kulcs alapján hivatkoznak az aktorra, és figyelők (*observer*) segítségével kapják meg a műveletek eredményét
  - végrehajtási környezete a *Silo*, amely tárolja az aktorokat
    - általában gépenként egy Silo fut, együttesen egy klasztert alkotnak

Elosztott alkalmazások architektúrái

Orleans