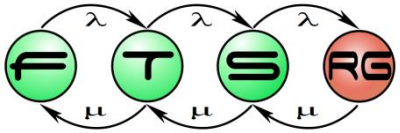
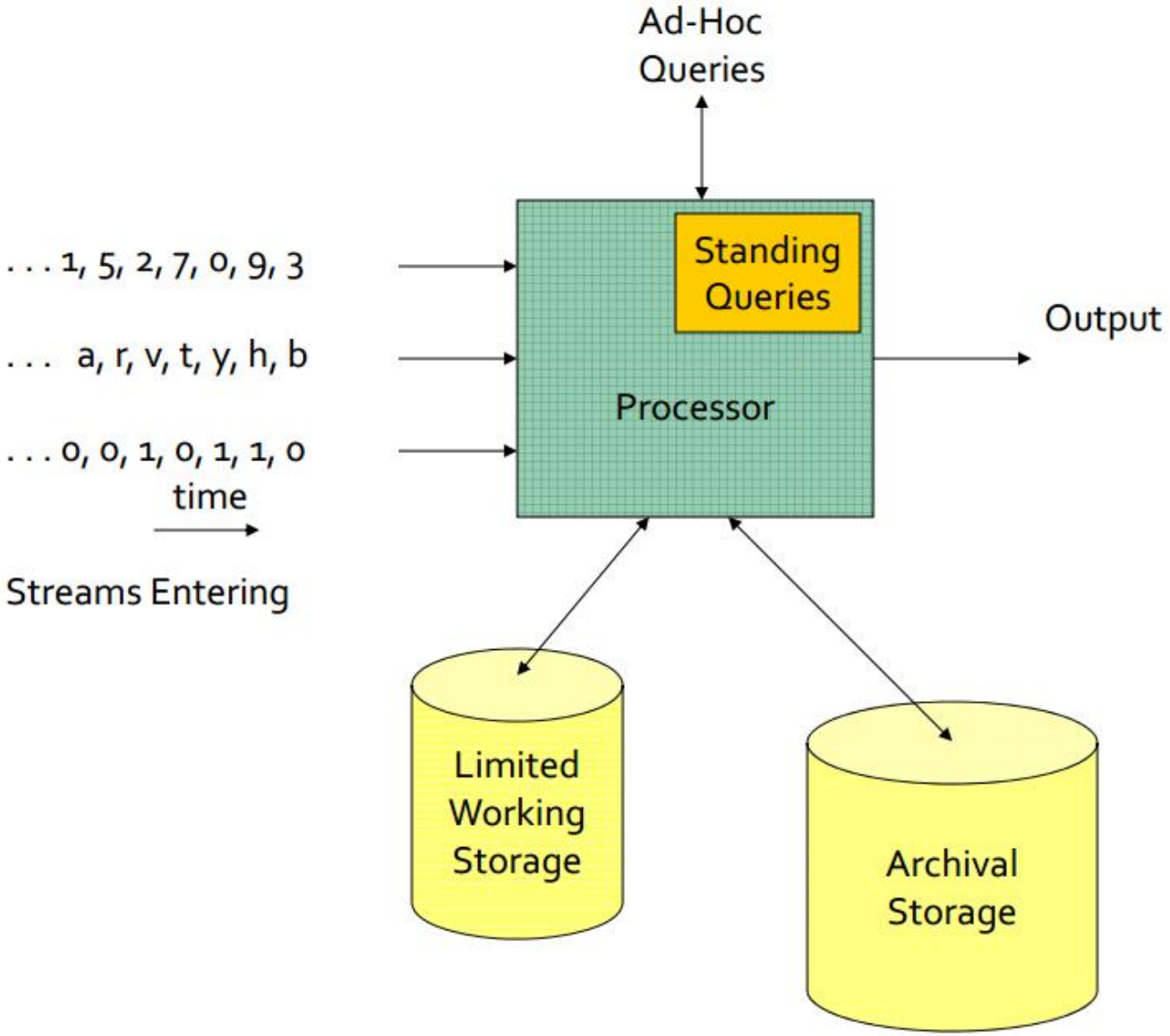
Stream processing

2017 ősz, 10. alkalom

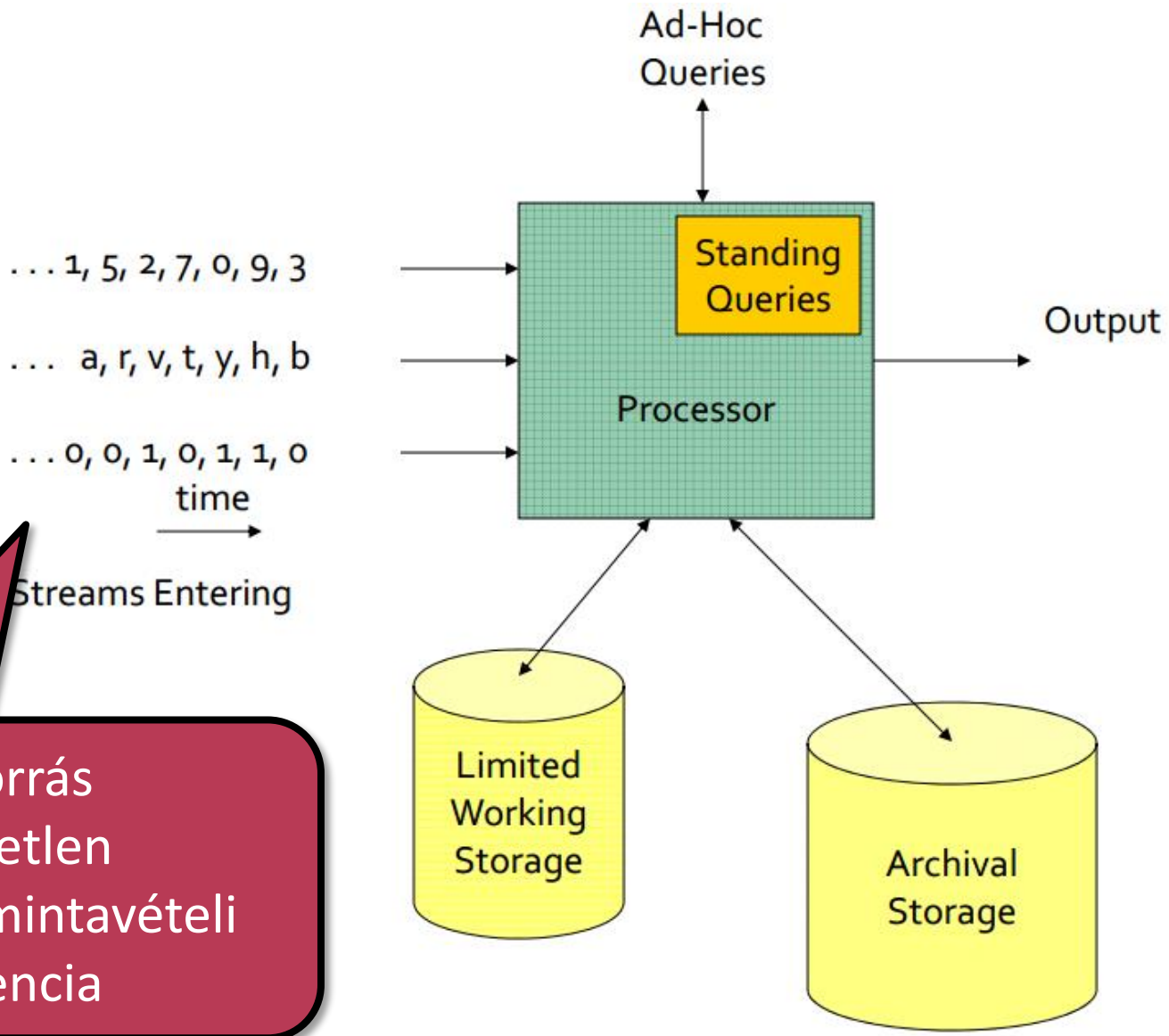
Kocsis Imre, ikocsis@mit.bme.hu



Az adatfolyam-feldolgozó elem: blokkséma

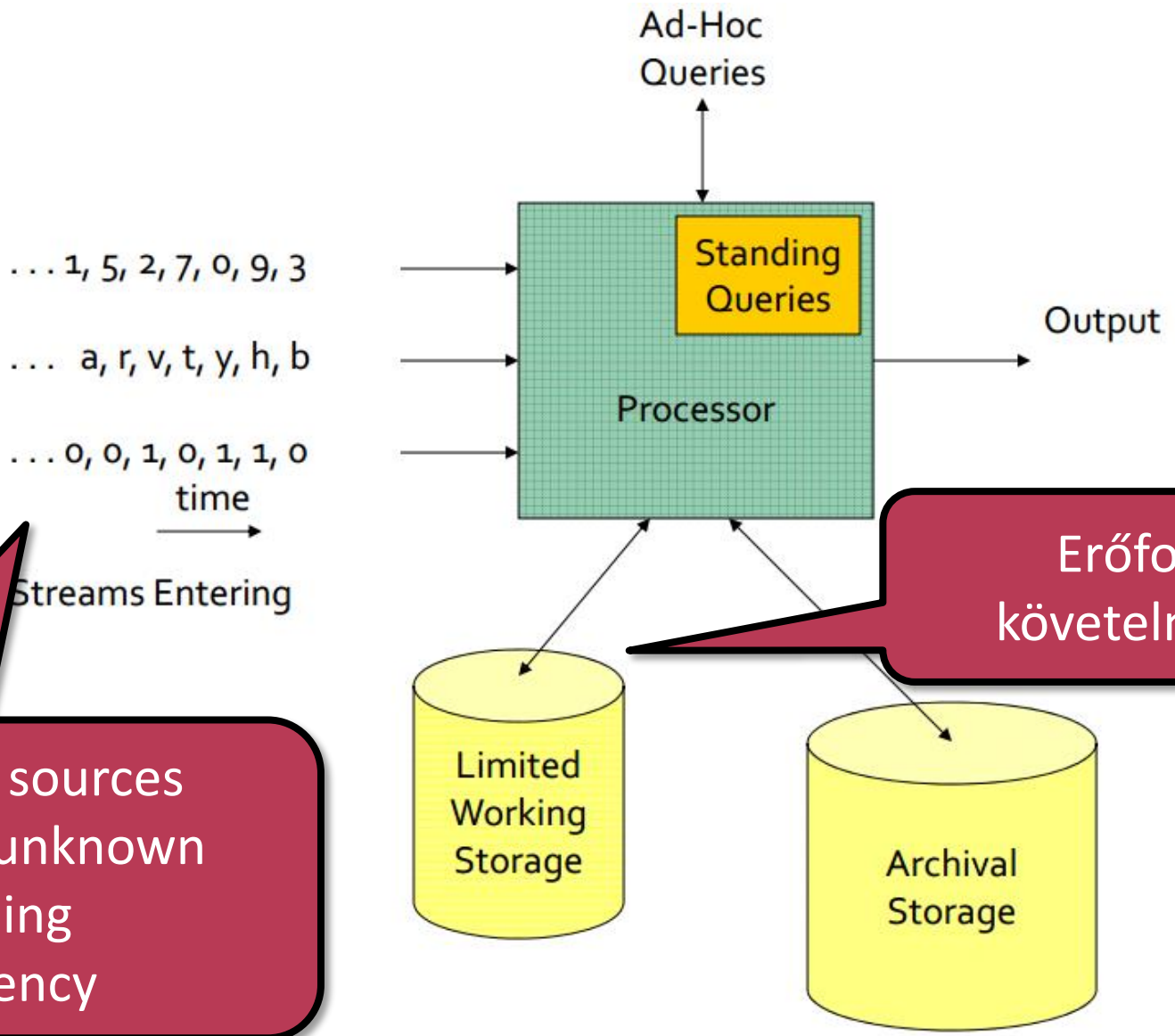


Az adatfolyam-feldolgozó elem



1. Sok forrás
2. Ismeretlen ráta/mintavételi frekvencia

Az adatfolyam-feldolgozó elem

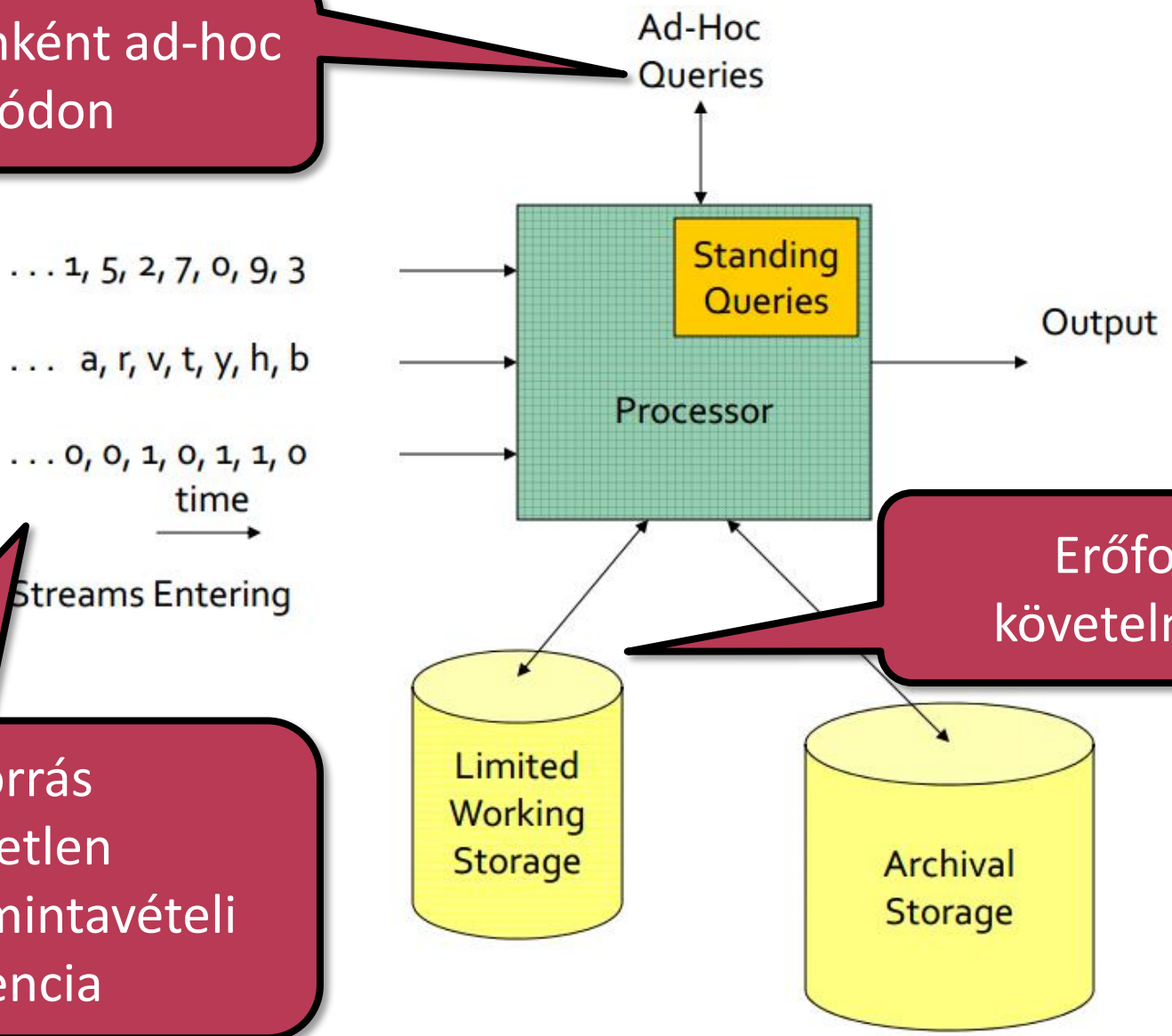


1. Many sources
2. With unknown sampling frequency

Erőforrás-követelmények

Az adatfolyam-feldolgozó elem

Folyamonként ad-hoc módon



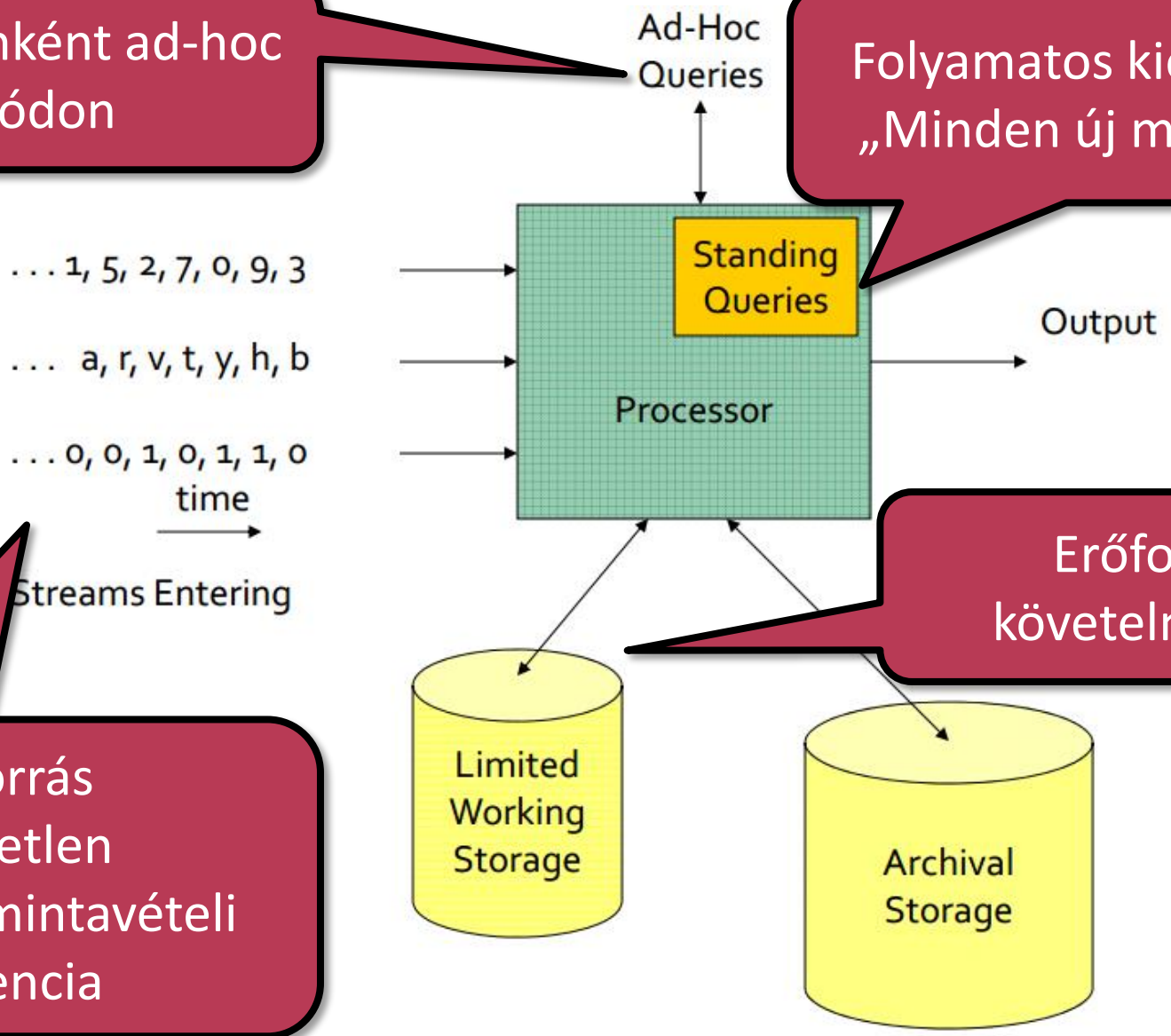
Erőforrás-követelmények

1. Sok forrás
2. Ismeretlen ráta/mintavételi frekvencia

Az adatfolyam-feldolgozó elem

Folyamonként ad-hoc módon

Folyamatos kiértékelés: „Minden új maximum”

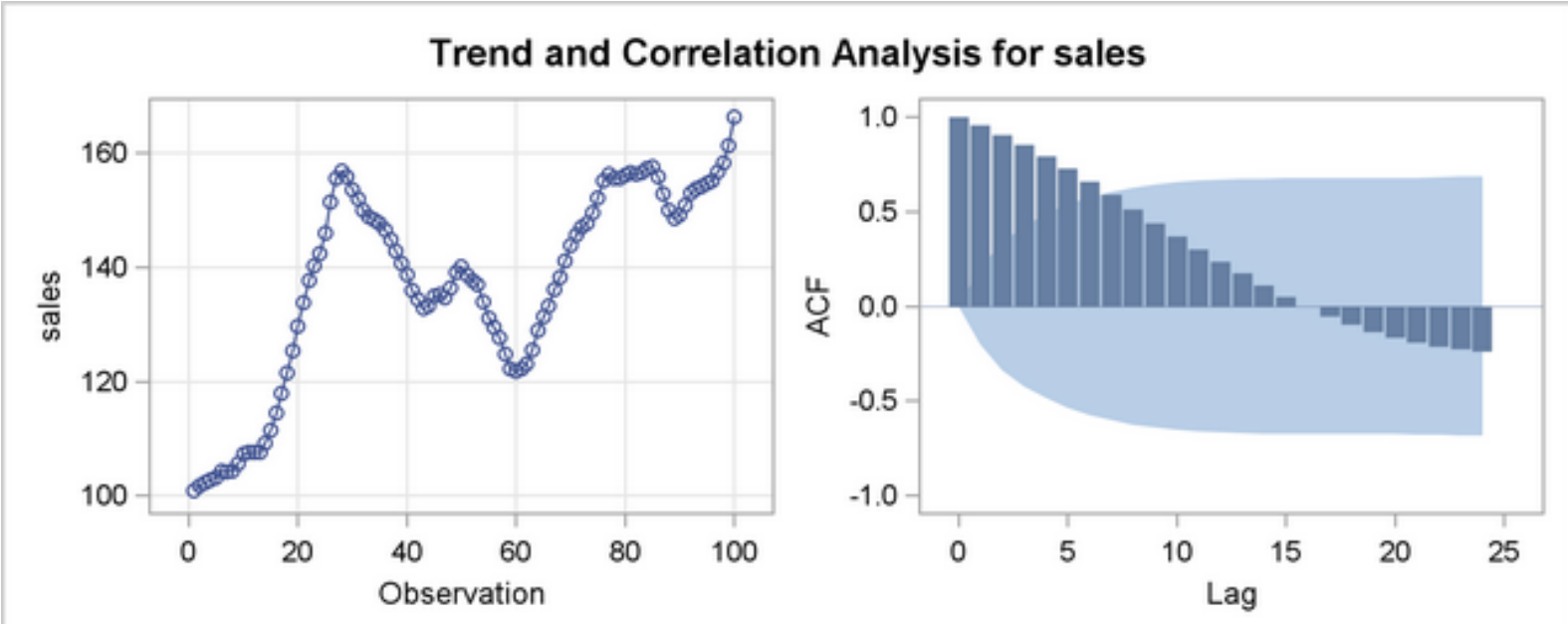


Erőforrás-követelmények

1. Sok forrás
2. Ismeretlen ráta/mintavételi frekvencia

Tipikus logika: mozgóablakos elemzések

- Pl. előzőleg felépített autoregresszív modellel előrejelzés
  - Hol térünk el a predikciótól?
  - Hol változik a modell?

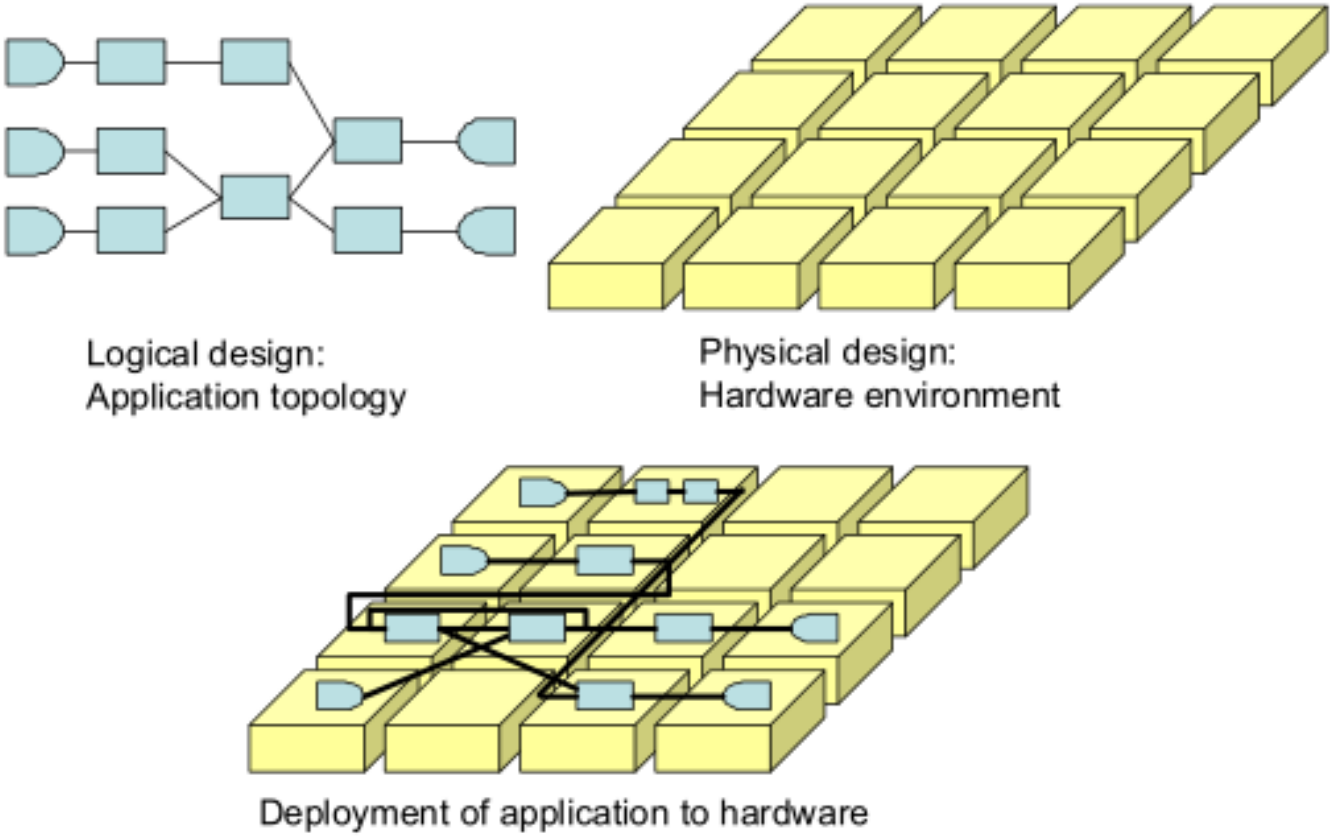


Feldolgozás: időkorlát!

- Diszk nem használható
- Megengedett memóriaigény: korlátos
- Elemenkénti számítási igény: korlátos

- Szokásos megoldások:
  - n-esenkénti (*tuple*) feldolgozási logika
  - Csúszóablakos tárolás és feldolgozás
  - Mintavételezés
  - Közelítő algoritmusok
  - WCET-menedzsment: skálázási logikán keresztül
    - Illetve lehet heurisztika/mintavétel-hangolás is, de az nehéz

Logikai topológia vs “deployment”

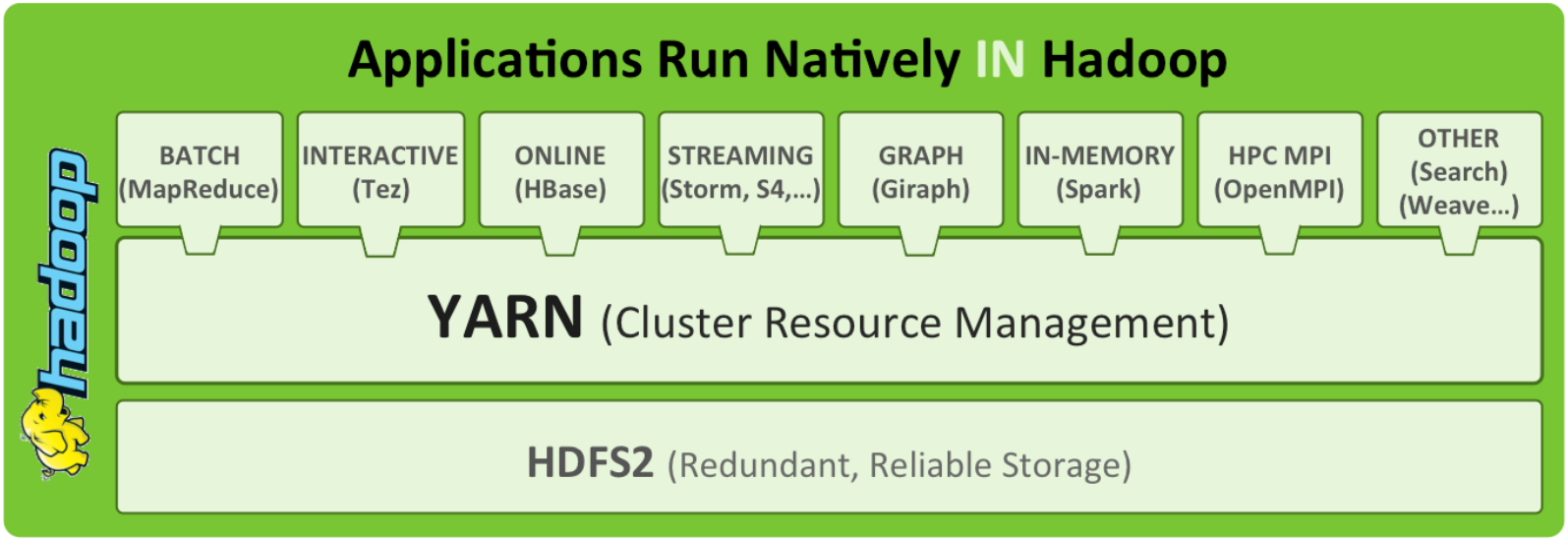


Forrás: [2], p 76

Eszközök

- Apache Spark Streaming
- Apache Storm

Ábra forrása: [3]



- IBM InfoSphere Streams
- Amazon Kinesis
- ~~LinkedIn~~ Apache Samza
- ...

+ kapcsolódó projektek

APACHE STORM

Demo: egy villanykari kollégium Netflow rekordjai

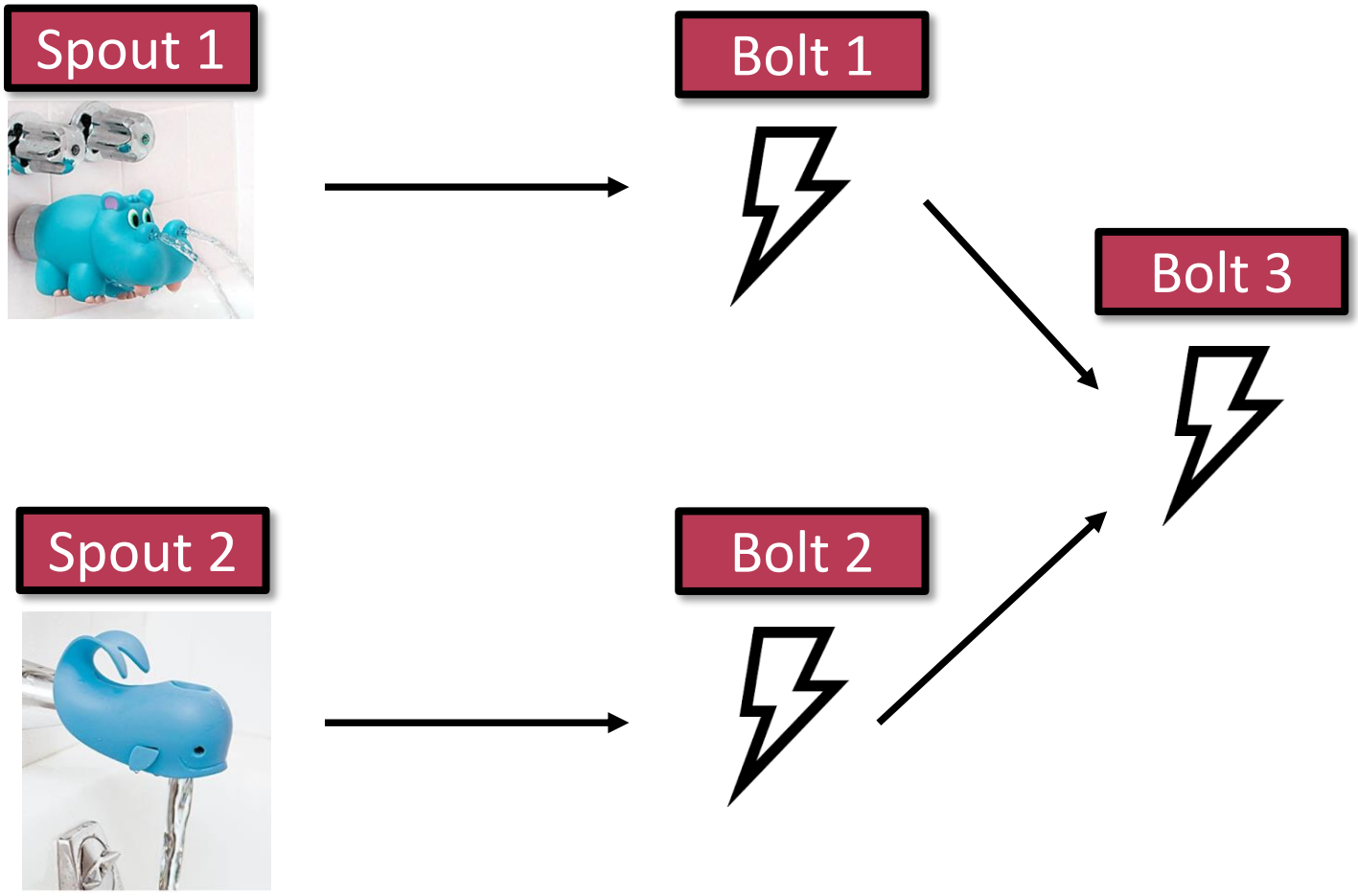
Apache Storm

- http://storm.apache.org
- Twitter, open source: 2011
- Belül: Clojure (“LISP JVM felett”)
  - Leginkább érdekesség

- Netflix, Twitter, Yahoo, Spotify, …

- “Igazi” stream processing
  - Nem mikrokötegelés (lehet azt is)
- Elosztott, “hibatűrő”
- Alacsony késleltetés
- Kiváló skálázódás

Topológia: “kifolyók” és “villámok”

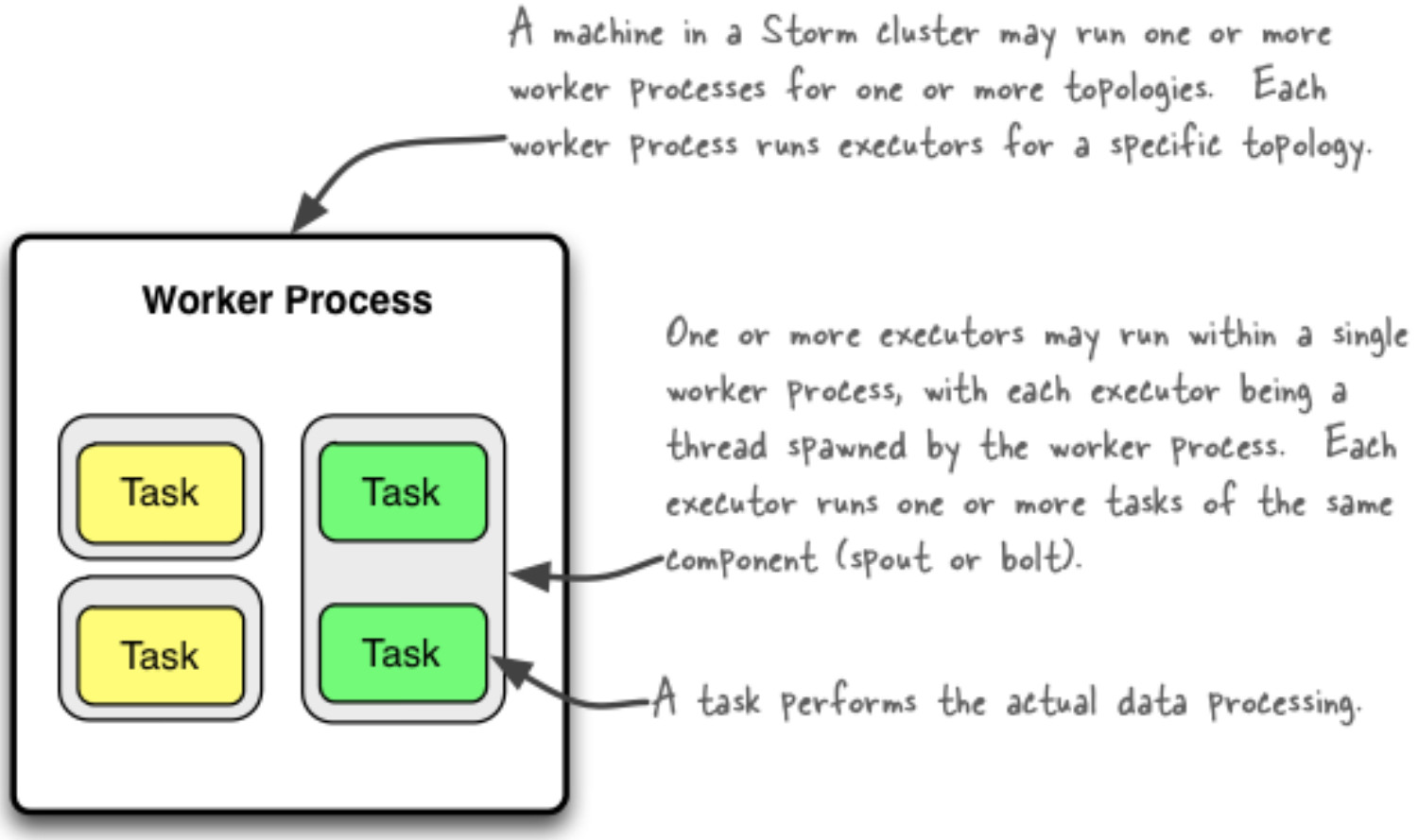


DAG topology: data + functions

Spouts and bolts

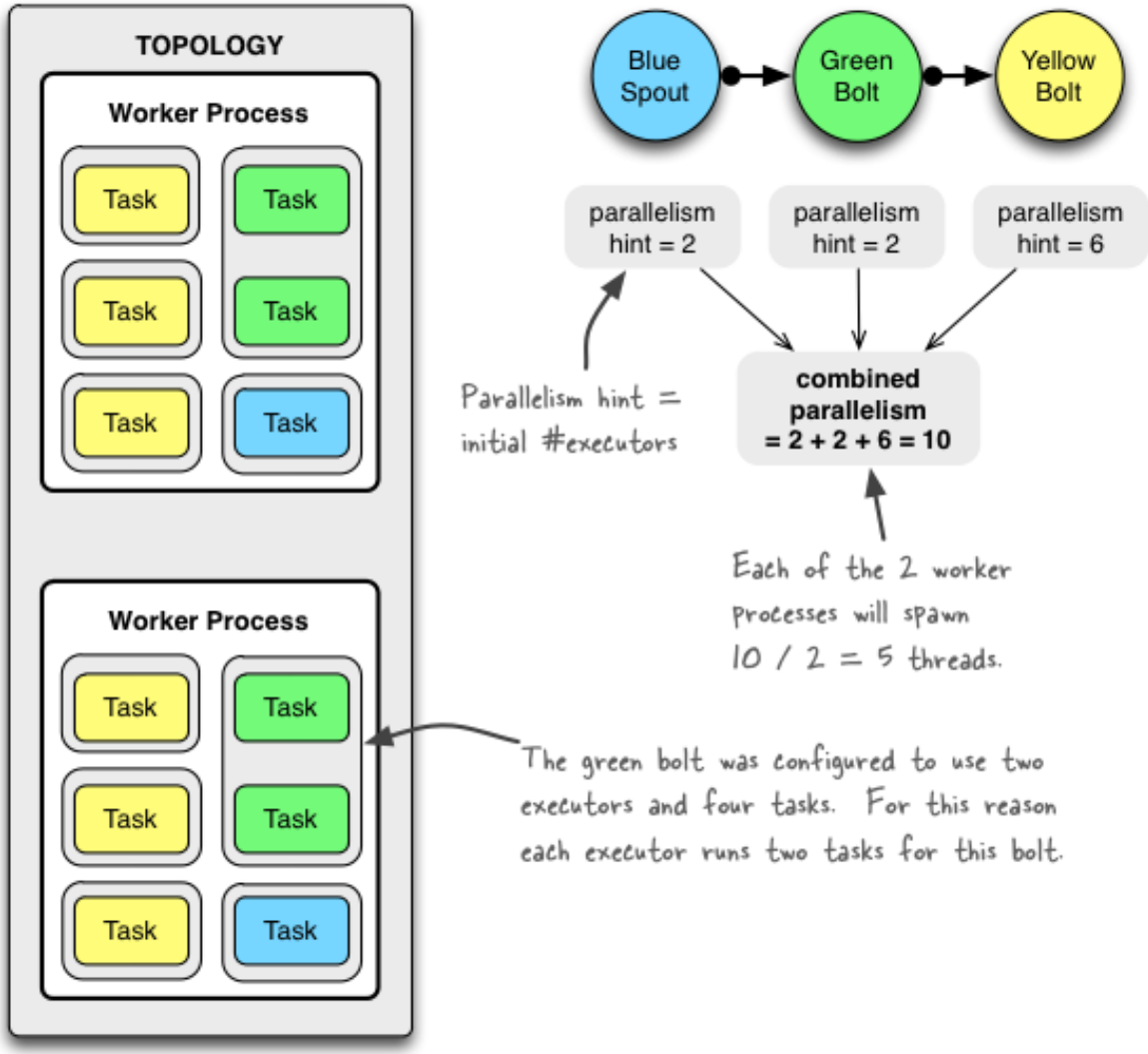
- Kifolyó: adatfolyam forrása
  - “Failed tuple” újrajátszása: fire & forget vagy reliable
  - Kafka, HDFS, MQTT, …
- Villám: 1 vagy több folyamot feldolgoz
  - És 0+ létrehoz
  - “Bármi” (időkereten belül) – filter, join, “kibeszélés”, …
  - Hbase, HDFS, JDBC, Mongo, Redis, Cassandra, JMS, ElasticSearch, …
- Nyelvi lehetőségek
  - JVM – Java, Scala, Jruby, Clojure, …
  - “Akármi” (JSON stdin/stdout) – Ruby, Python, Javascript, Perl, R, …

“Leterített” topológia - fogalmak



Ábra forrása: http://storm.apache.org/releases/2.0.0-SNAPSHOT/Understanding-the-parallelism-of-a-Storm-topology.html

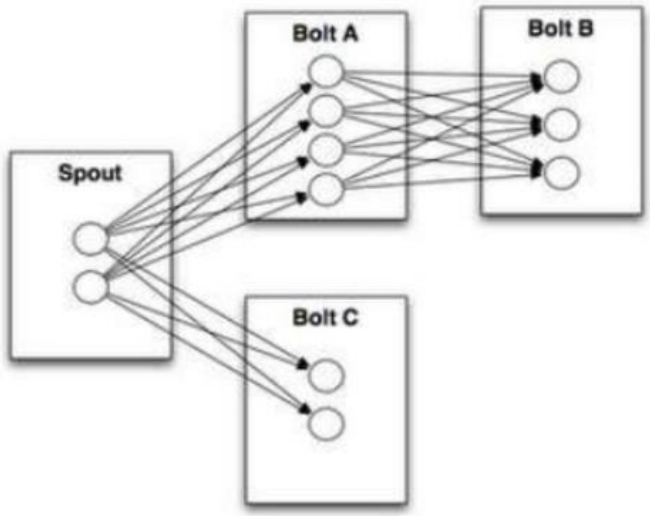
Storm - párhuzamosítás



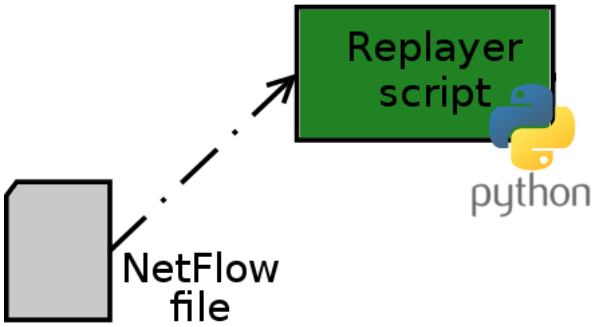
Ábra forrása: http://storm.apache.org/releases/2.0.0-SNAPSHOT/Understanding-the-parallelism-of-a-Storm-topology.html

Stream-csoportosítás – “routing”

- Shuffle: random
- Fields: GROUP BY k,l
- Partial Key: fields, de terheléselosztva két bolt-ra
- All: replikálás…
- Global: minden a legkisebb ID-jú bolthoz
- Direct: “a küldő oszt”
- …
- Custom

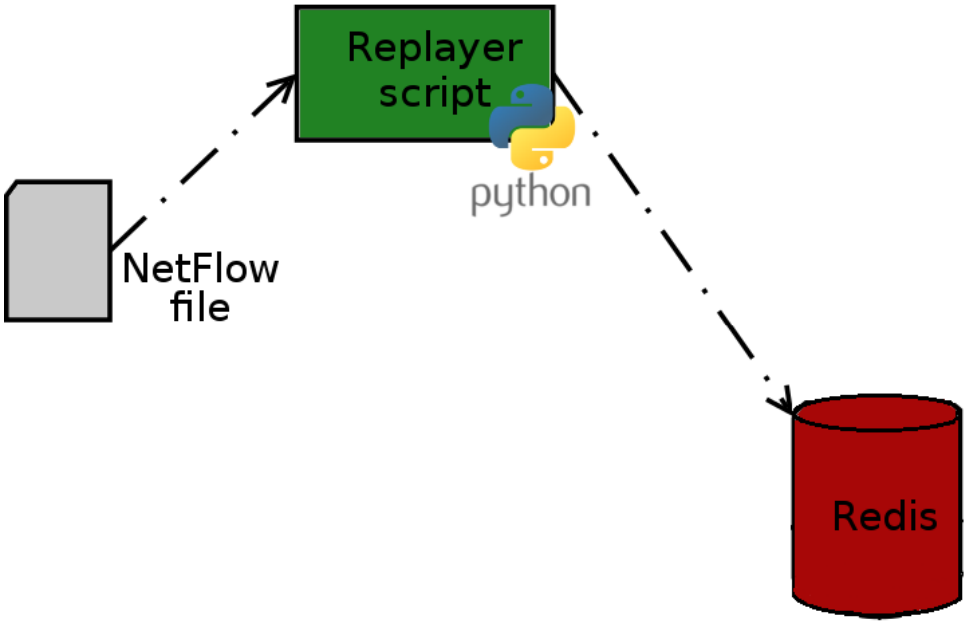


Alkalmazás adatfolyam



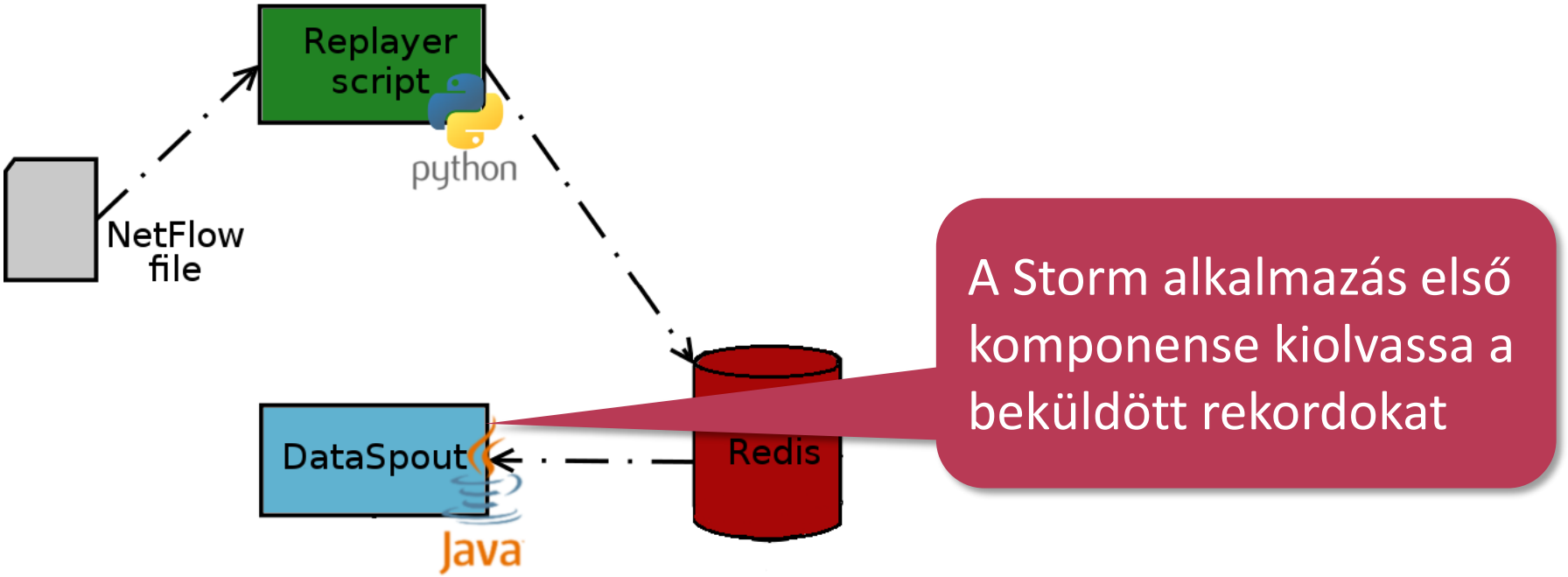
- A lementett hálózati adatokat tartalmazó rekordokat fájlból kiolvassuk
- Egy rekordban szerepel
  - a forrás és cél IP cím,
  - időpont
  - forgalmazott csomagszám
  - adatmennység

Alkalmazás adatfolyam

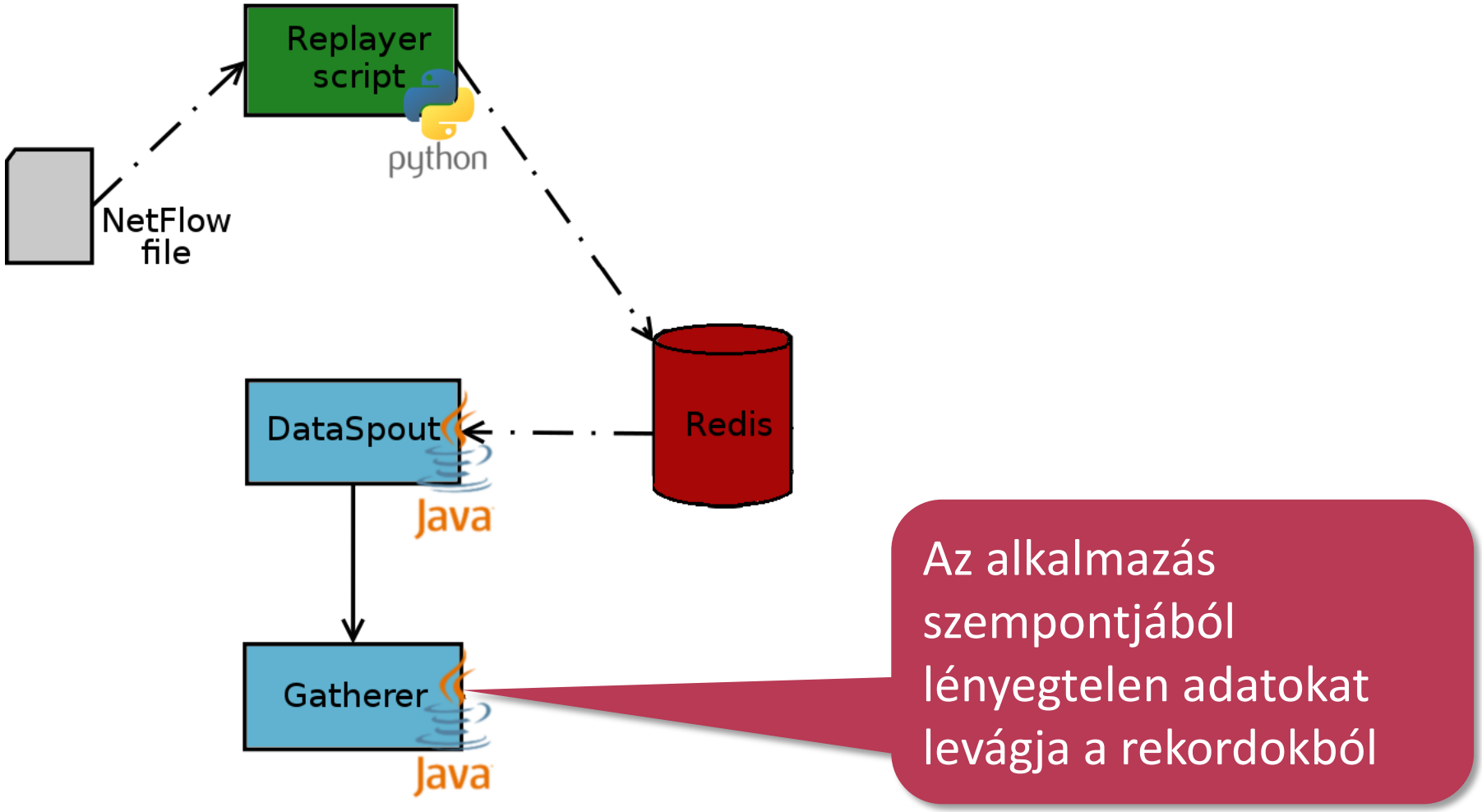


- A hálózati rekordokat egy adatbázisba küldjük

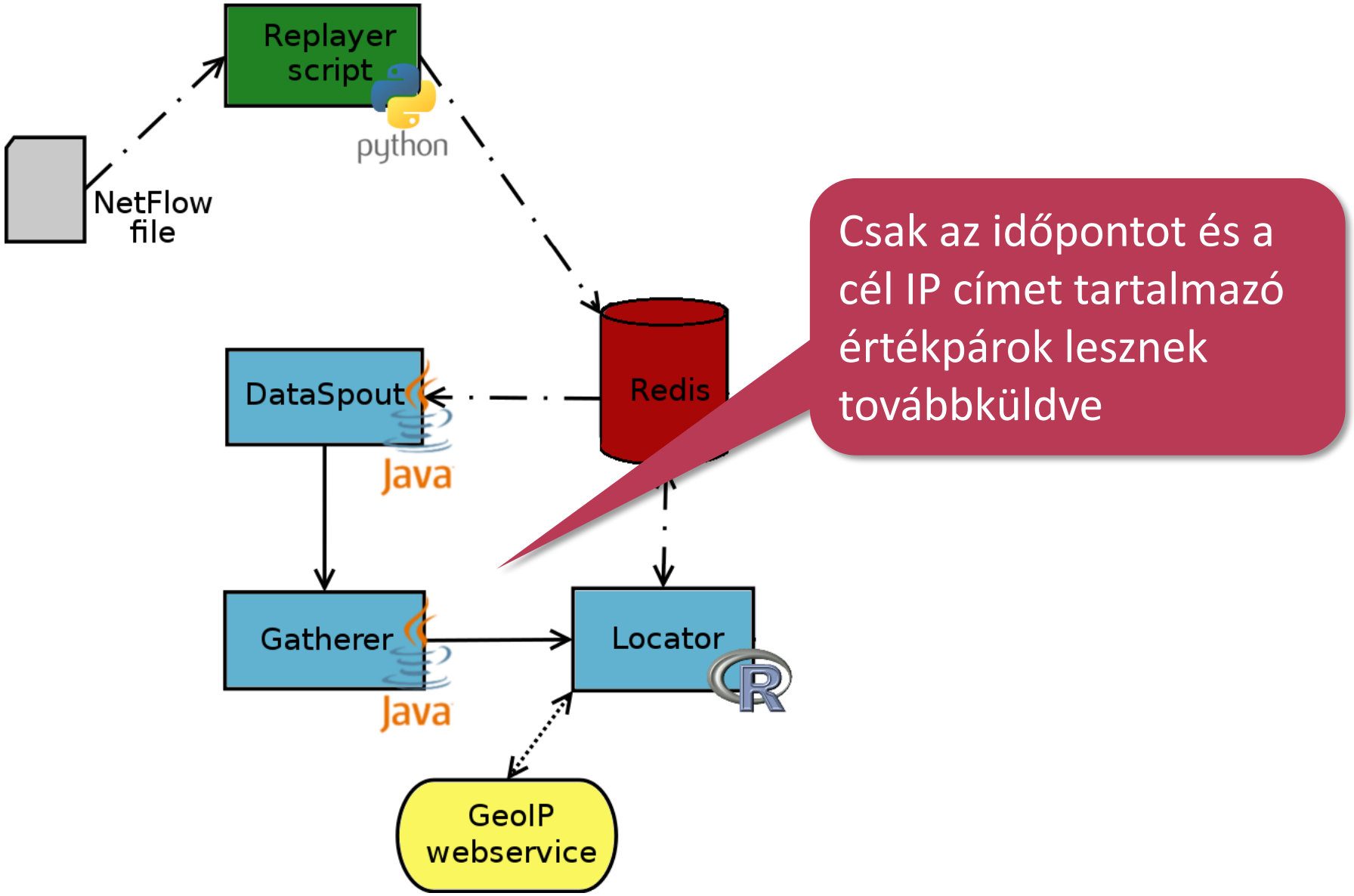
Alkalmazás adatfolyam



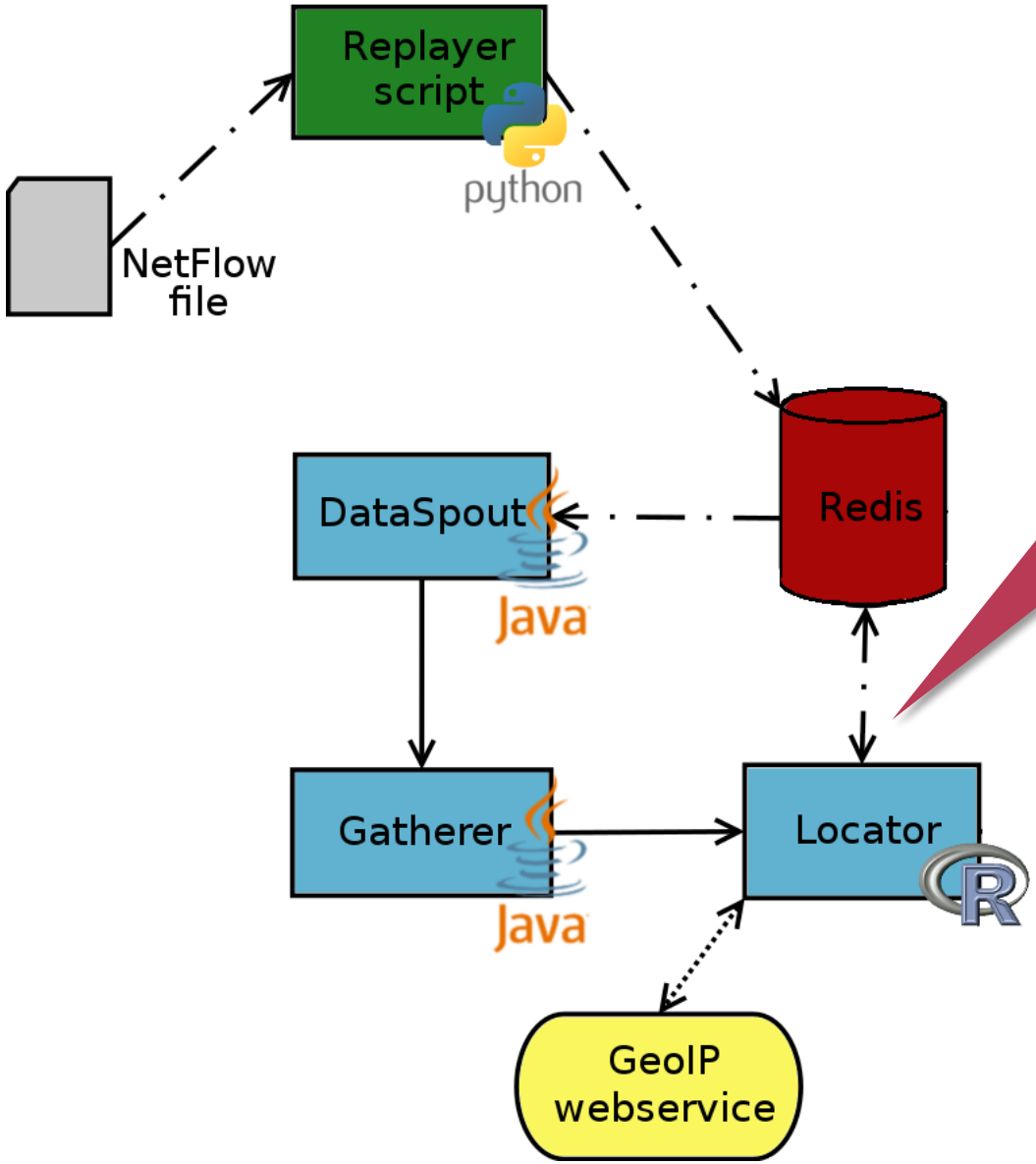
Alkalmazás adatfolyam



Alkalmazás adatfolyam

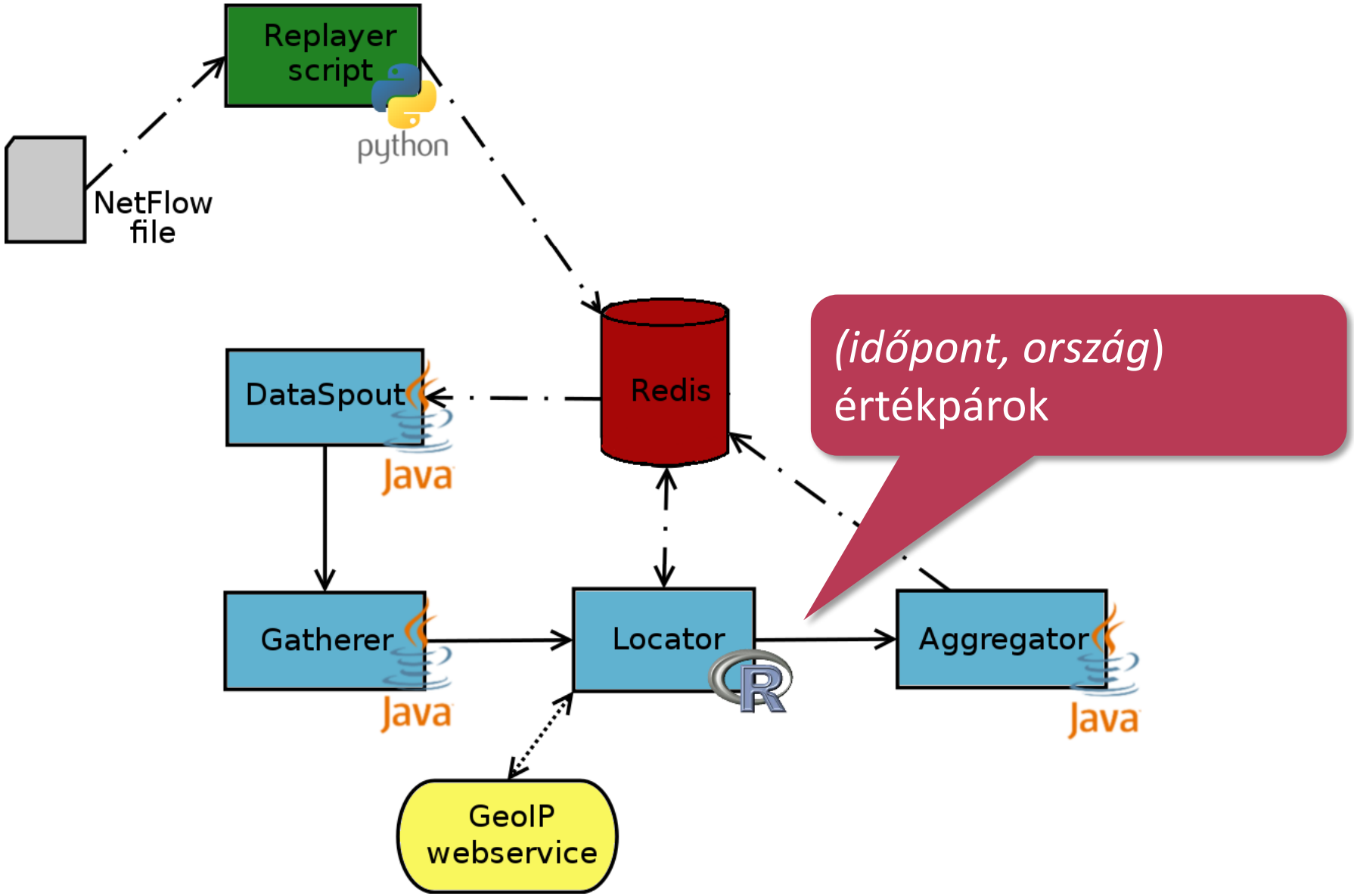


Alkalmazás adatfolyam

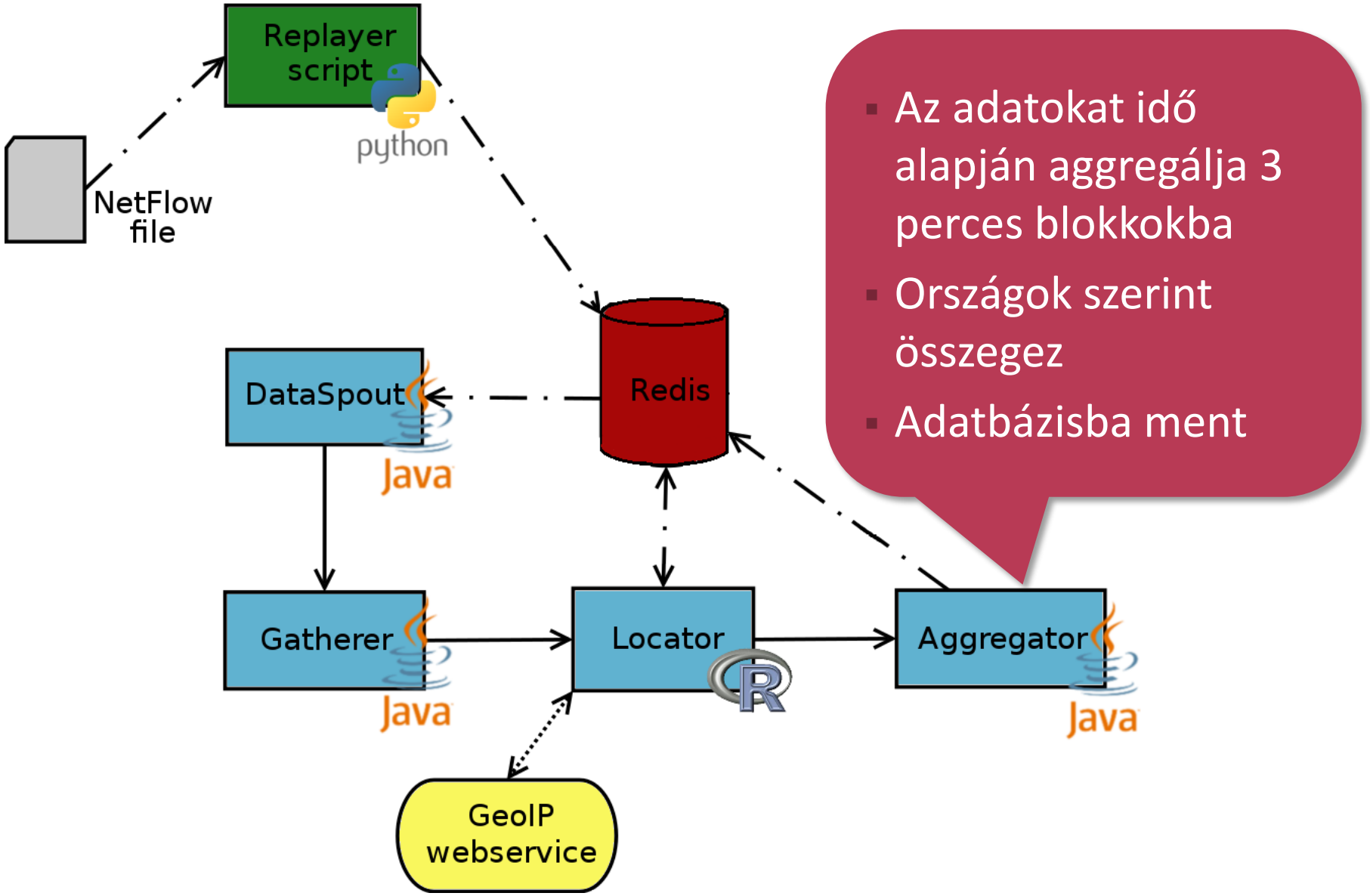


Egy külső web szolgáltatás segítségével az IP címekhez megkeresi a hozzátartozó országot

Alkalmazás adatfolyam

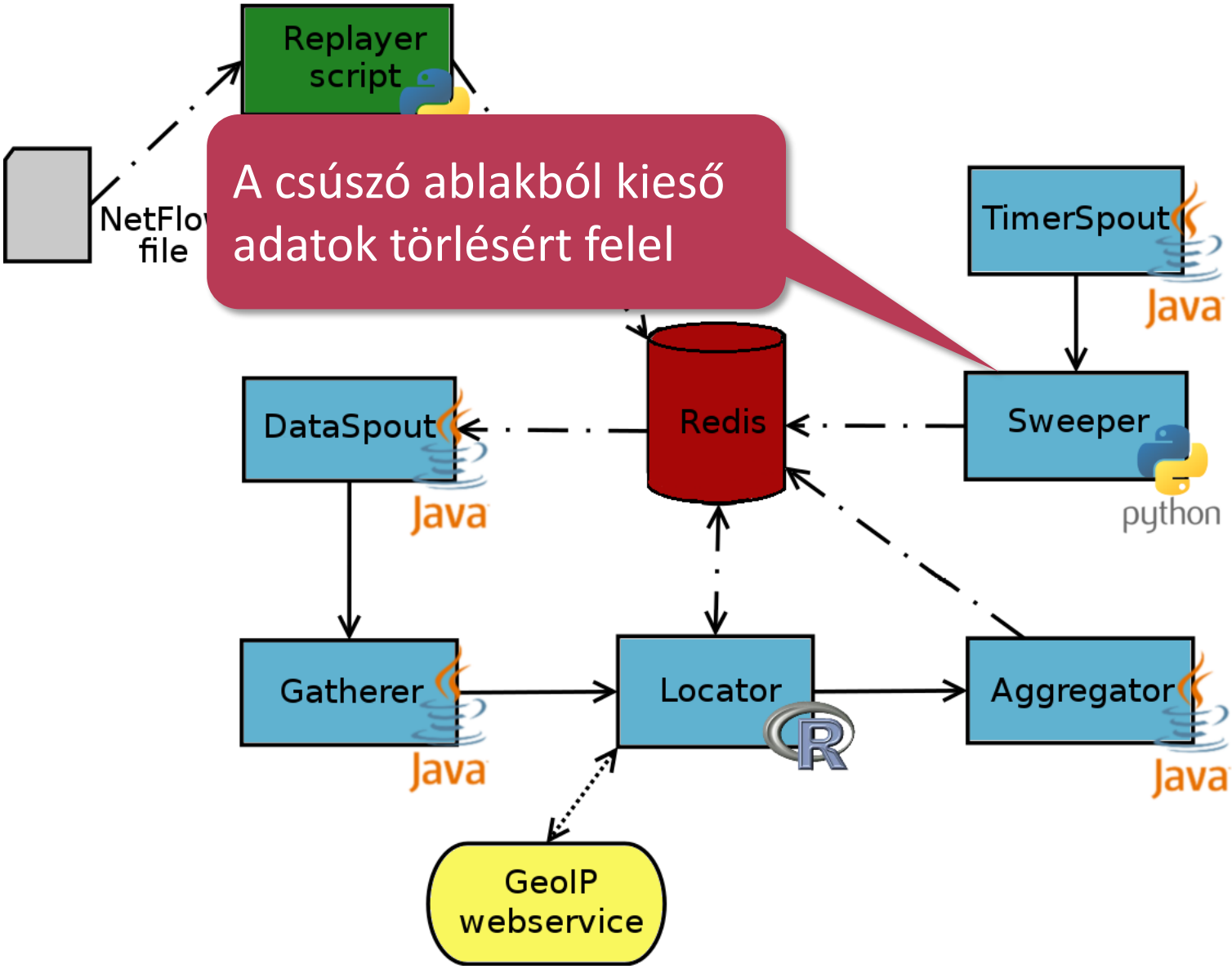


Alkalmazás adatfolyam

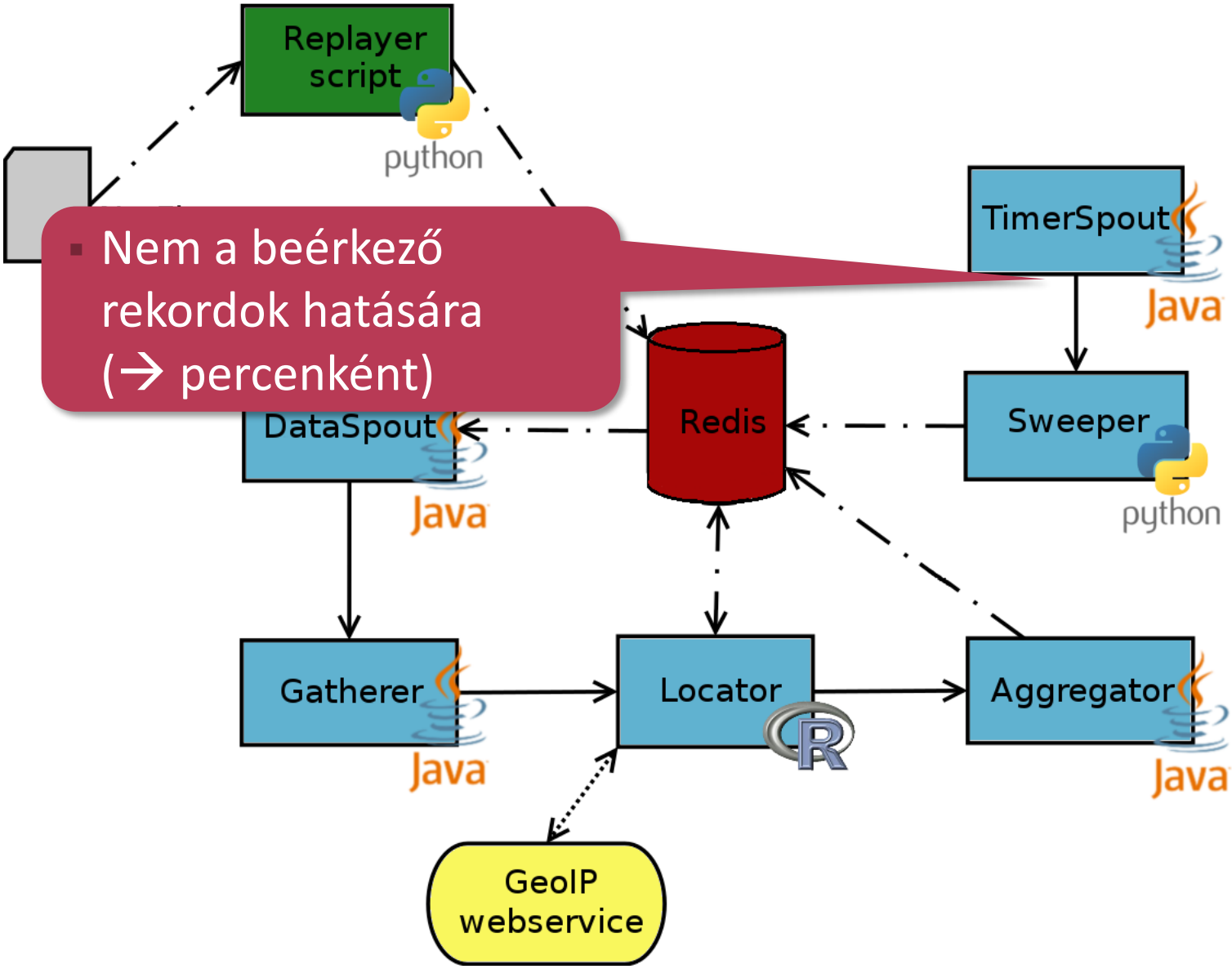


- Az adatokat idő alapján aggregálja 3 perces blokkokba
- Országok szerint összegez
- Adatbázisba ment

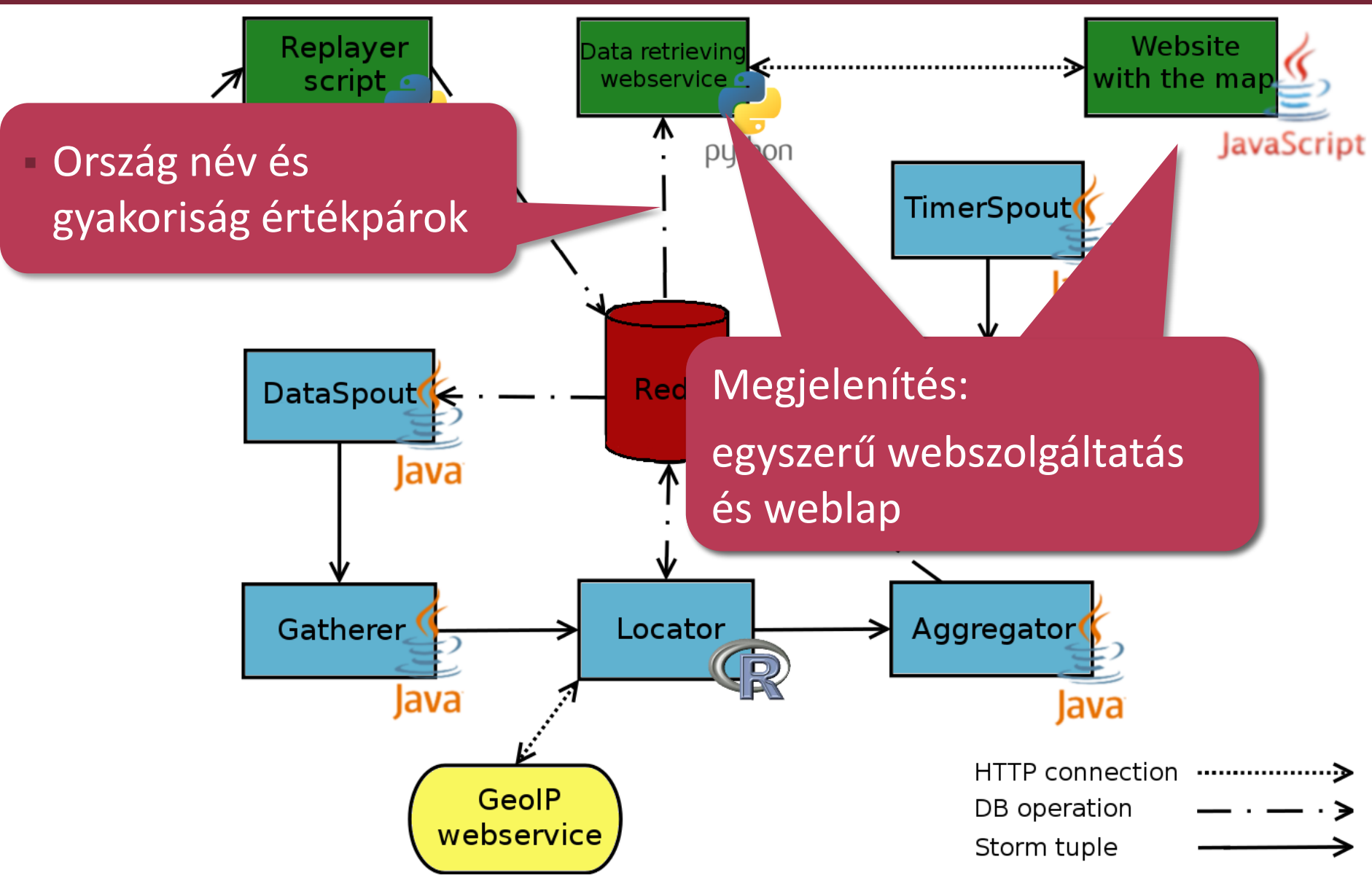
Alkalmazás adatfolyam



Alkalmazás adatfolyam



Alkalmazás adatfolyam

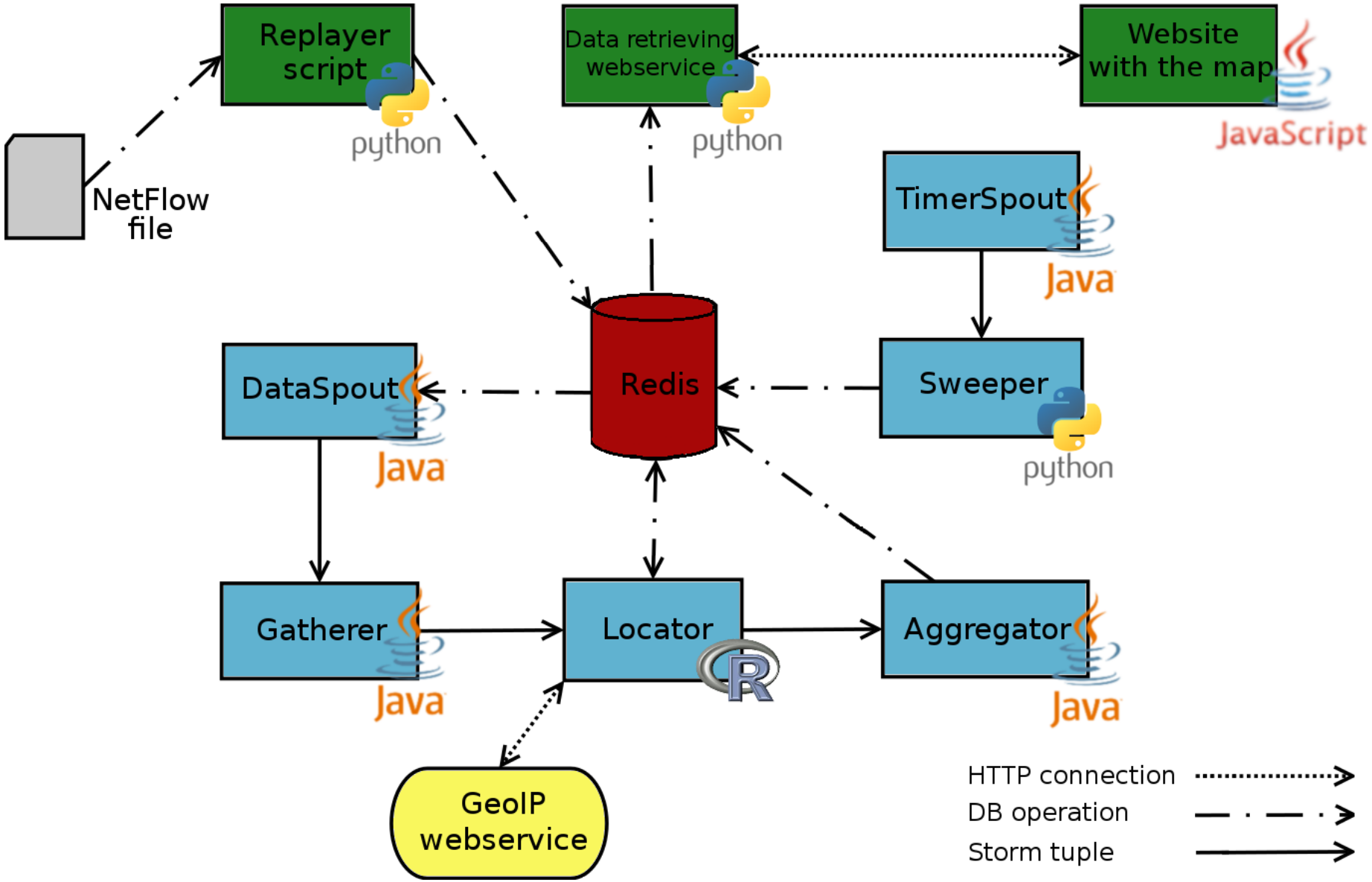


- Ország név és gyakoriság értékpárok

Megjelenítés:

egyszerű webszolgáltatás és weblap

Alkalmazás adatfolyam



Szöveges “folyamat” (topológia)

```
TopologyBuilder builder = new TopologyBuilder();

builder.setSpout("redis_spout", new RedisSpout(), 1);
builder.setBolt("gatherer", new Gatherer(), 5)
                                .shuffleGrouping("redis_spout");
builder.setBolt("locator", new GeoTagger(), 10)

        .shuffleGrouping("gatherer");
builder.setBolt("aggregator", new Aggregator(), 10)
                                        .fieldsGrouping("locator",
new Fields("date"));
builder.setSpout("timer_spout", new TimerSpout(), 1);
builder.setBolt("sweeper", new Sweeper(), 5)

        .shuffleGrouping("timer_spout");
```

Kód példa: szószámlálás

```
public static class WordCount extends BaseBasicBolt {
  Map<String, Integer> counts = new HashMap<String, Integer>();

  @Override
  public void execute(Tuple tuple, BasicOutputCollector collector) {
    String word = tuple.getString(0);
    Integer count = counts.get(word);
    if (count == null)
      count = 0;
    count++;
    counts.put(word, count);
    collector.emit(new Values(word, count));
  }

  @Override
  public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declare(new Fields("word", "count"));
  }
}
```

https://github.com/nathanmarz/storm-starter/blob/master/src/jvm/storm/starter/WordCountTopology.java

Szószámlálás topológiája

```
public static void main(String[] args) throws Exception {

  TopologyBuilder builder = new TopologyBuilder();

  builder.setSpout("spout", new RandomSentenceSpout(), 5);

  builder.setBolt("split", new SplitSentence(), 8).shuffleGrouping("spout");
  builder.setBolt("count", new WordCount(), 12).fieldsGrouping("split", new Fields("word"));
```

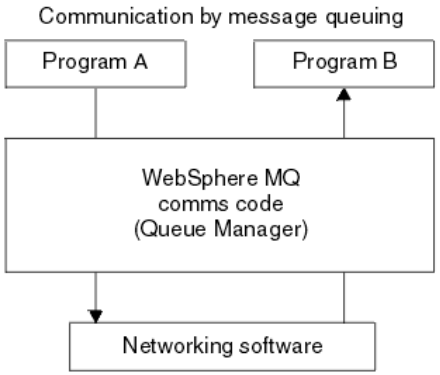
https://github.com/nathanmarz/storm-starter/blob/master/src/jvm/storm/starter/WordCountTopology.java

Amiről nem beszélünk

- Méretezés
  - Áteresztőképesség
  - Késleltetés
  - Rate limiting mechanizmsok
- Architektúra
  - Terítés, erőforráskezelés (semmi, YARN, Mesos, …)
  - Topológia-koordináció: ZooKeeper
  - *nem* kötelezően kötött a Hadoop stackhez
  - HDFS és YARN kellhet; Kafka integráció valószínűleg fontosabb
- További használati módok
  - DRPC (Distributed RPC), folyamatosan futó számítások

Kafka + Storm

- Valamikor réges-régen
  - MQ, pub/sub
  - WebSphere MQ, JMS, …

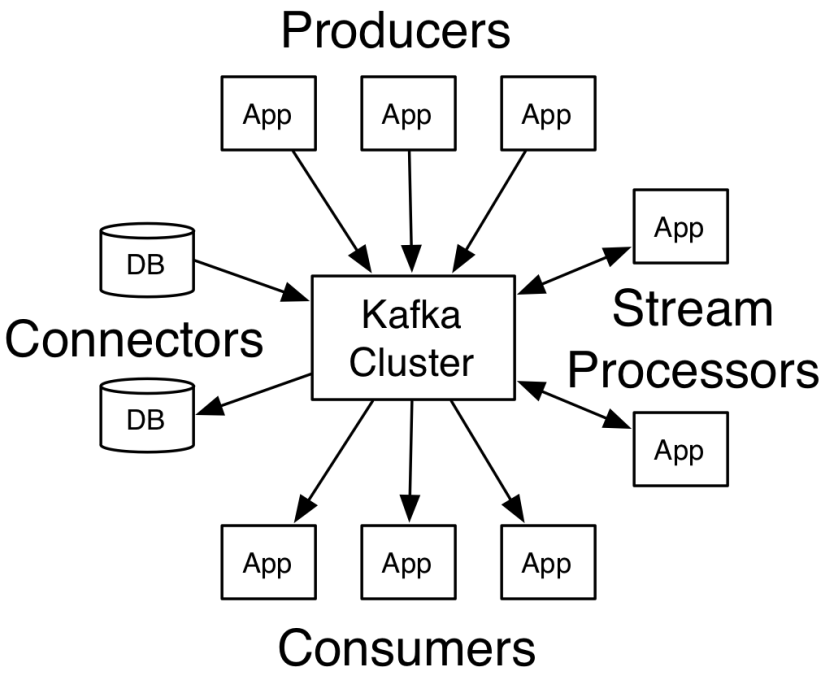


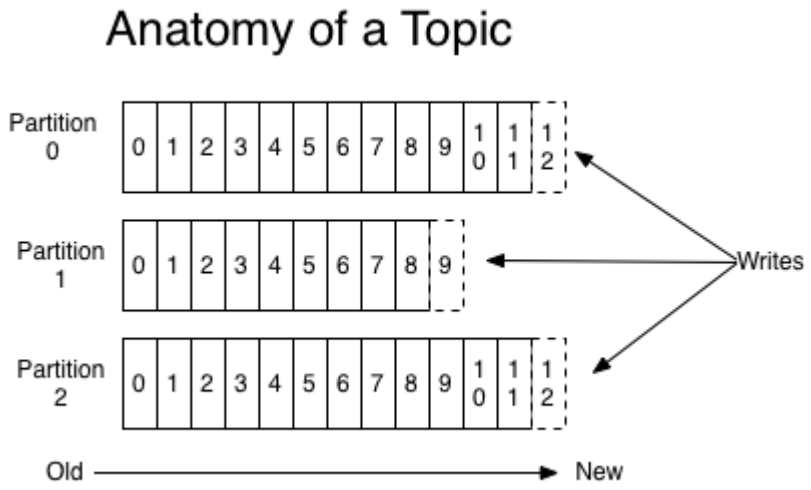
- Innováció
  - **F/OSS**
  - **DC**
    - elosztott, real time, nagy áteresztőképességű, hibatűrő, perzisztens pub/sub stream-ek
    - pl. Kafka
  - **Embedded/IoT/CPS**
    - alacsony overhead, nyílt, esetleg “brokerless”
    - pl. OMG DDS

kafka

DDS The Proven Data Connectivity Standard for the IoT

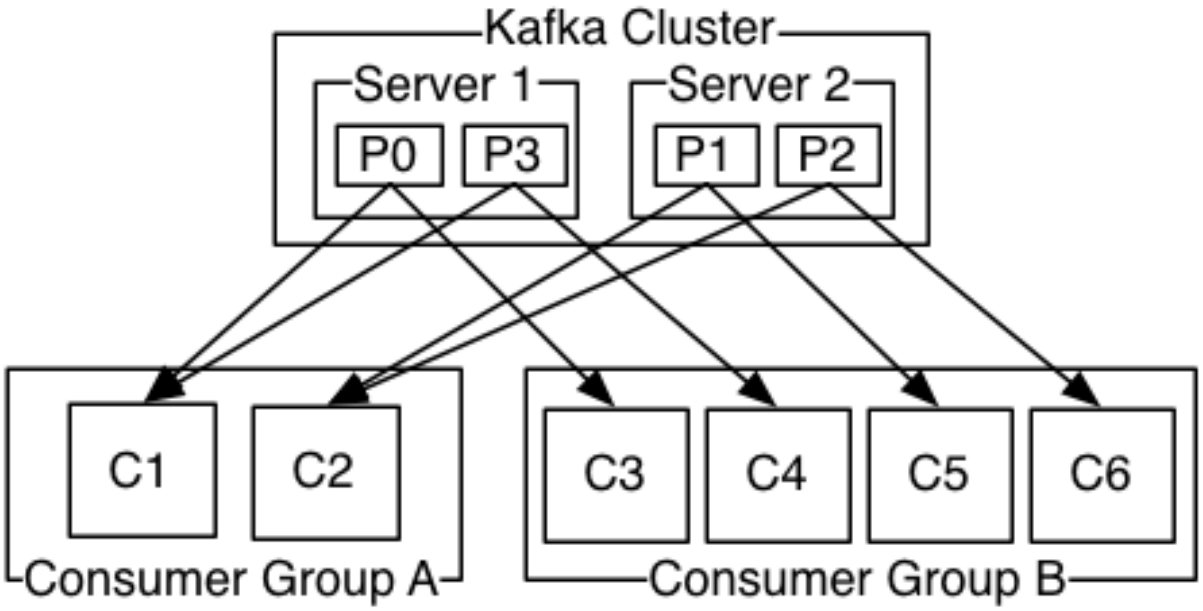
Apache Kafka





Ábrák forrása: https://kafka.apache.org/intro

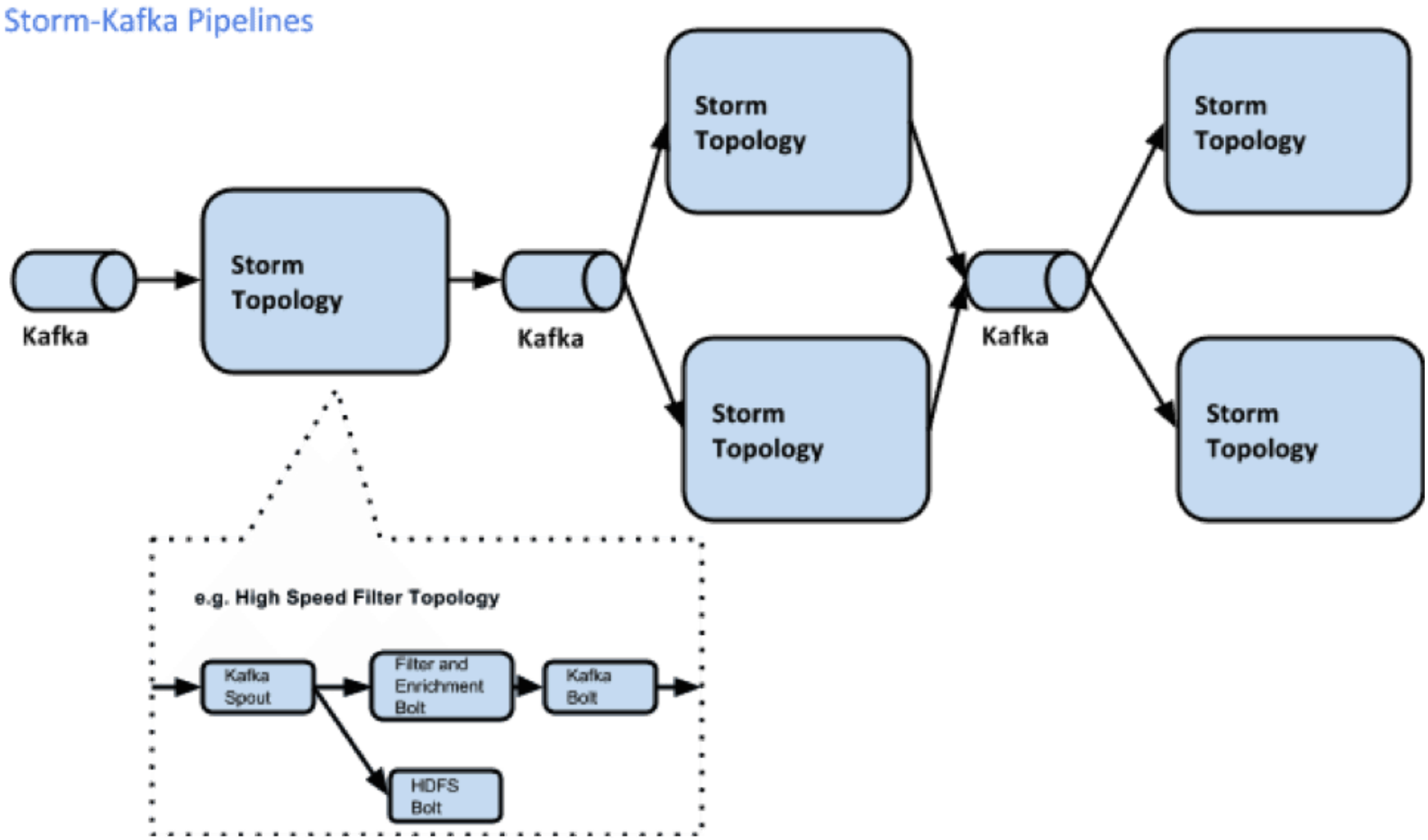
Apache Kafka



- Producer dönti el a partícióba rendelést
- Egy P-t CG-onként egy C olvashat, de több CG (sub)
- Írás és olvasás szigorúan rendezett partíciónként
- N- szeres partíció-replikáció: N-1 hiba tűrése

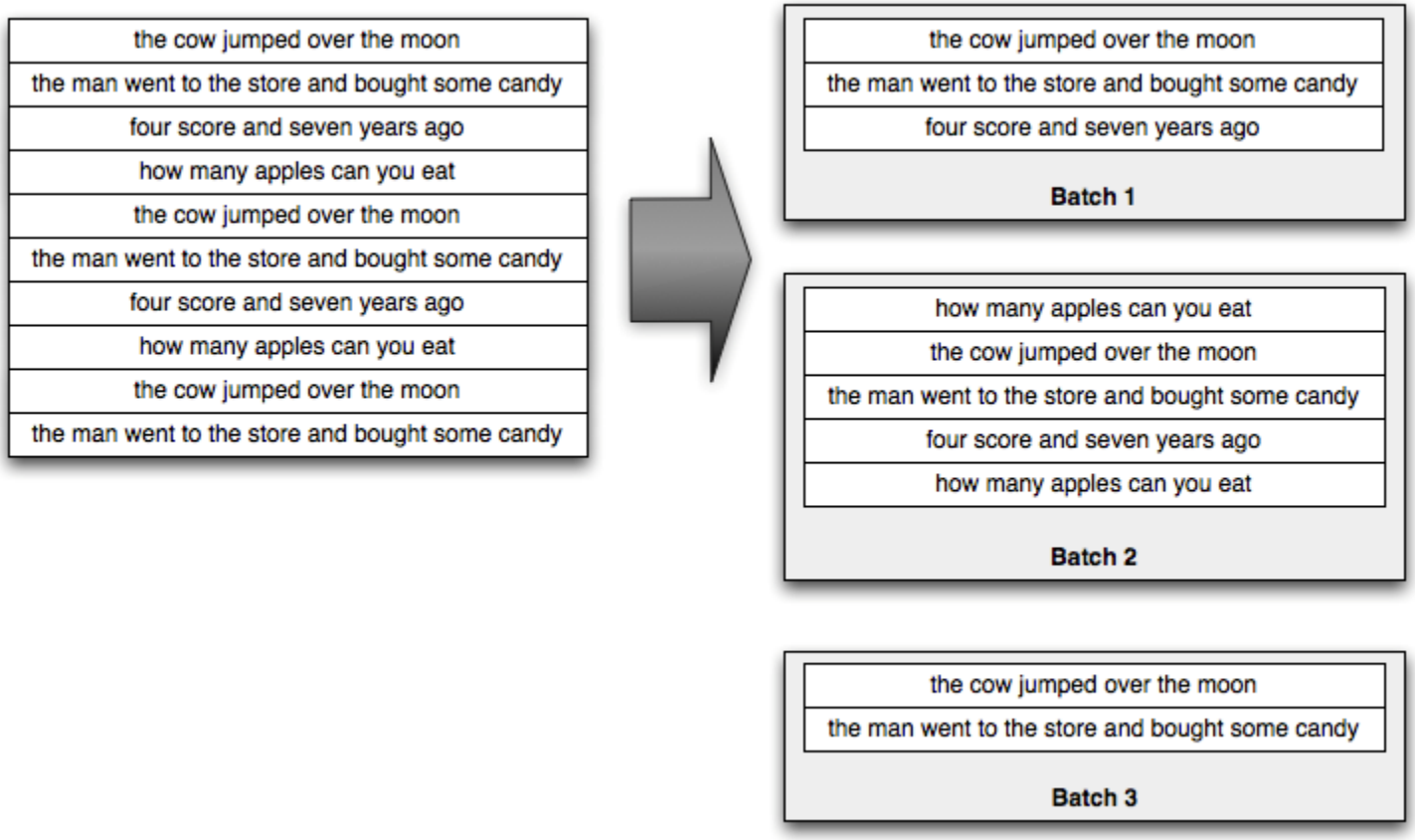
Ábrák forrása: https://kafka.apache.org/intro

Kafka + Storm



Ábra forrása: https://hortonworks.com/blog/storm-kafka-together-real-time-data-refinery/

Mikrokötegelés (microbatching)



A “mikro” azért relatív

http://storm.apache.org/releases/2.0.0-SNAPSHOT/Trident-tutorial.html

Magas szintű microbatching: **Trident**

Állapot-definíció

```
TridentTopology topology = new TridentTopology();
TridentState wordCounts =
     topology.newStream("spout1", spout)
       .each(new Fields("sentence"), new Split(), new Fields("word"))
       .groupBy(new Fields("word"))
       .persistentAggregate(new MemoryMapState.Factory(), new Count(), new Fields("count"))
       .parallelismHint(6);
```

Query-definíció

```
topology.newDRPCStream("words")
       .each(new Fields("args"), new Split(), new Fields("word"))
       .groupBy(new Fields("word"))
       .stateQuery(wordCounts, new Fields("word"), new MapGet(), new Fields("count"))
       .each(new Fields("count"), new FilterNull())
       .aggregate(new Fields("count"), new Sum(), new Fields("sum"));
```

Lekérdezés

```
DRPCClient client = new DRPCClient("drpc.server.location", 3772);
System.out.println(client.execute("words", "cat dog the man");
// prints the JSON-encoded result, e.g.: "[[5078]]"
```

http://storm.apache.org/releases/2.0.0-SNAPSHOT/Trident-tutorial.html

Magas szintű microbatching: **Trident**

Állapot-definíció

```
TridentTopology topology = new TridentTopology();
TridentState wordCounts =
     topology.newStream("spout1", spout)
       .each(new Fields("sen                                    ))
       .groupBy(new Fields("wor                                 ))
       .persistentAggregate(new MemoryMapState.Factory(), new Count(), new Fields("count"))
       .parallelismHint(6);
```

Karbantartott állapot

Query-definíció

```
topology.newDRPCStream("words")
       .each(new Fields("a                                      "))
       .groupBy(new Fields("                                    "))
       .stateQuery(wordCounts, new Fields("word"), new MapGet(), new Fields("count"))
       .each(new Fields("count"), new FilterNull())
       .aggregate(new Fields("count"), new Sum(), new Fields("sum"));
```

Karbantartott állapot

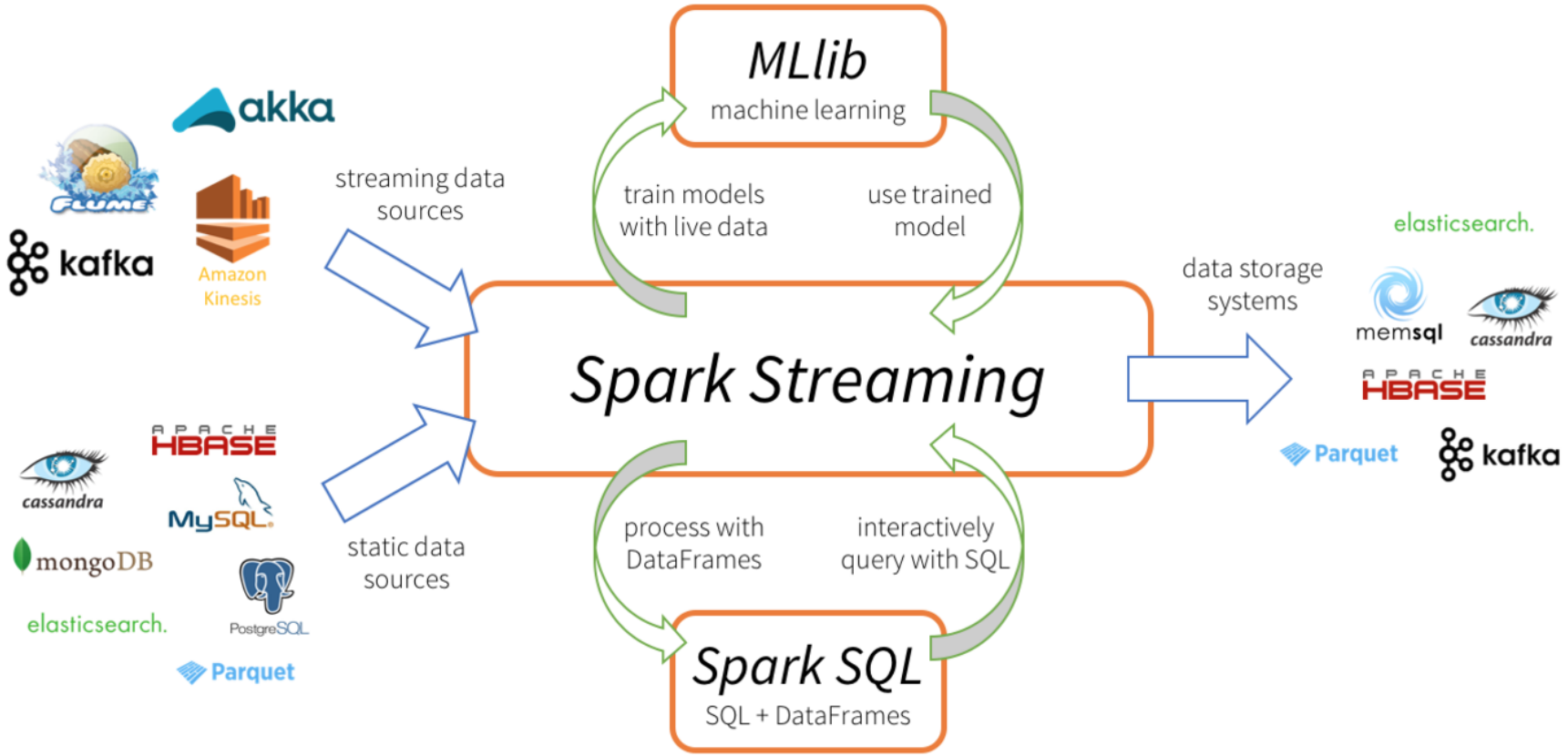
Lekérdezés

```
DRPCClient client = new DRPCClient("drpc.server.location", 3772);
System.out.println(client.execute("words", "cat dog the man");
// prints the JSON-encoded result, e.g.: "[[5078]]"
```

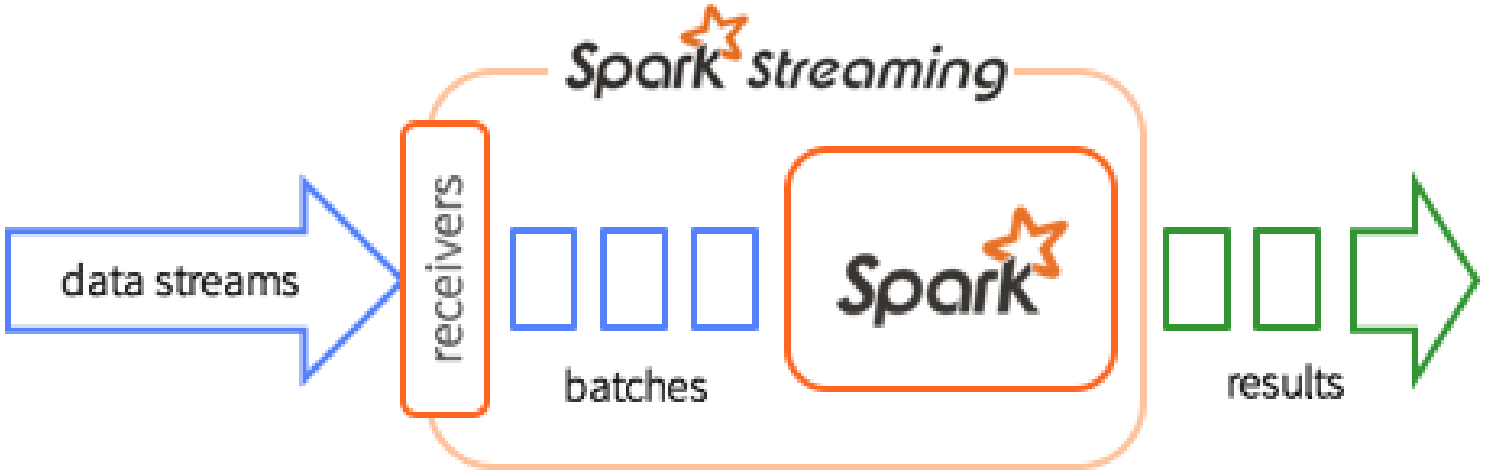
http://storm.apache.org/releases/2.0.0-SNAPSHOT/Trident-tutorial.html

SPARK STREAMING

Sparks streaming



“Fogadók” és mikrokötegek



Példa

- https://docs.cloud.databricks.com/docs/latest/databricks_guide/07%20Spark%20Streaming/01%20Streaming%20Word%20Count%20-%20Scala.html

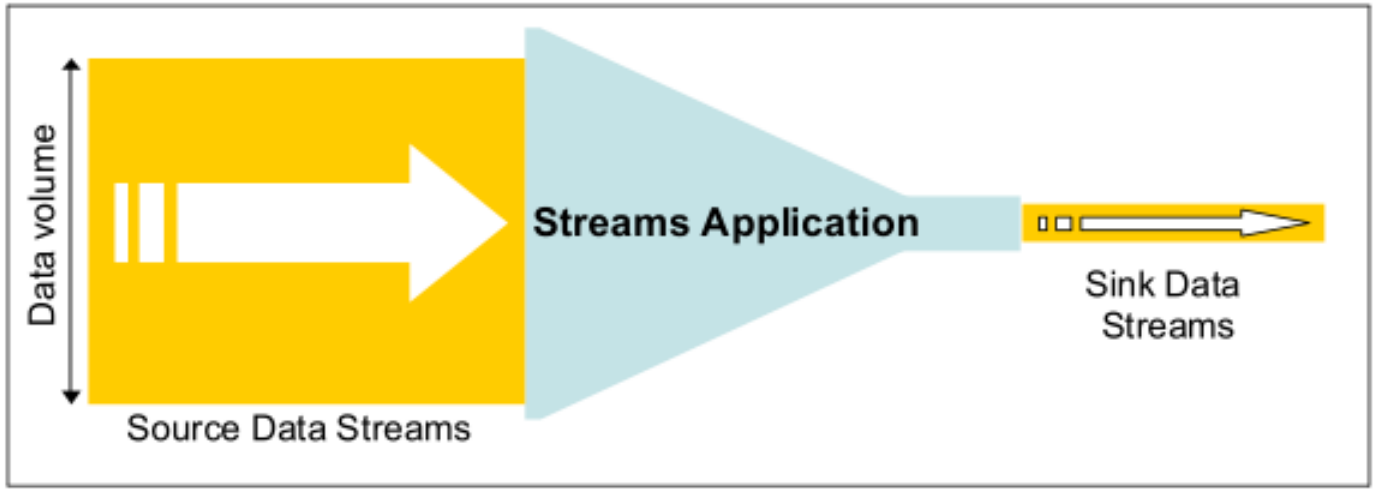
ALKALMAZÁSI MINTÁK

Alkalmazás-osztályok

Class	Sources	Output	Dependency	Performance
Throughput	Known, Fixed	Analysis		Throughput
Latency	Known, Fixed	Opportunities		Latency
Discovery	Partly known, Dynamic	Novel observation	Hypotheses, Stored data	
Prediction	Known, Multiple	Warnings	Human input	Accuracy
Control	Known, Controllable	Controls, Feedback		Latency

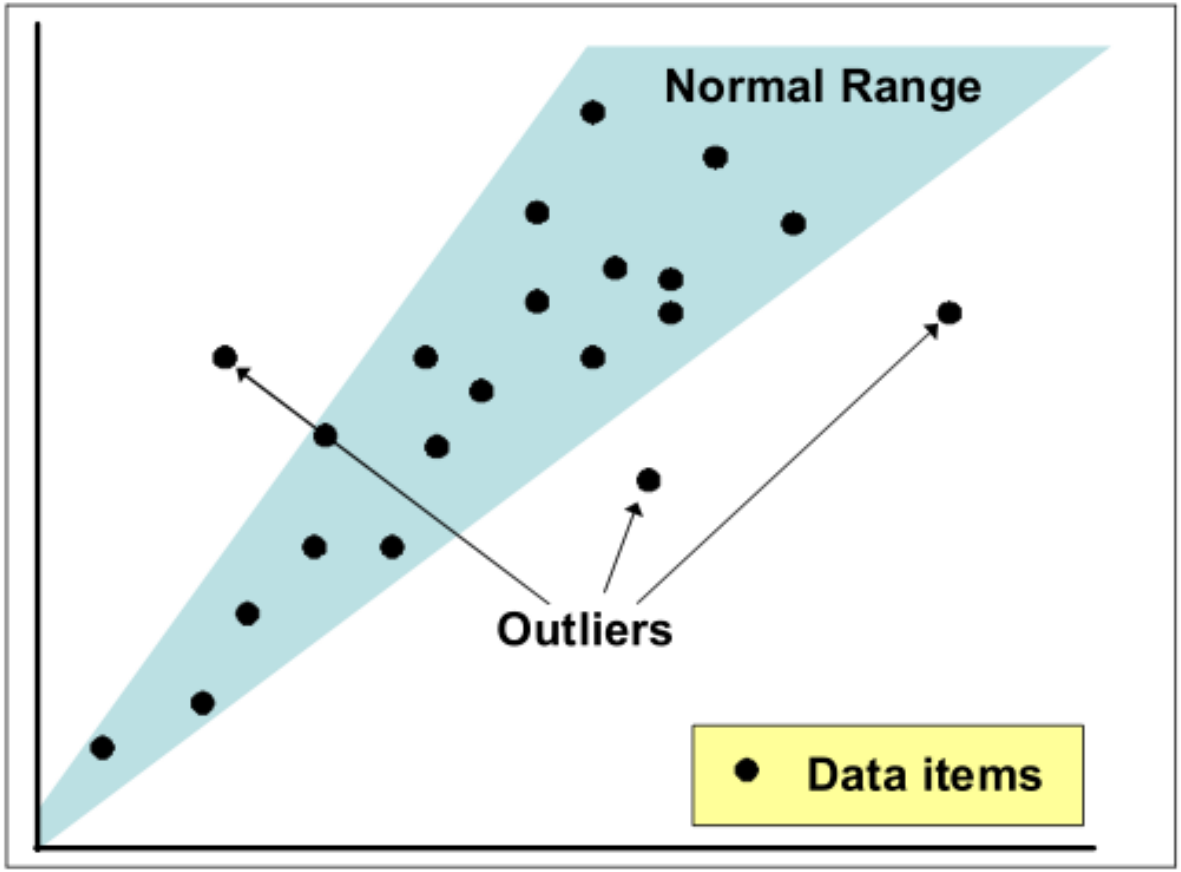
Forrás: [2], p 80

Tervezési minták: filter

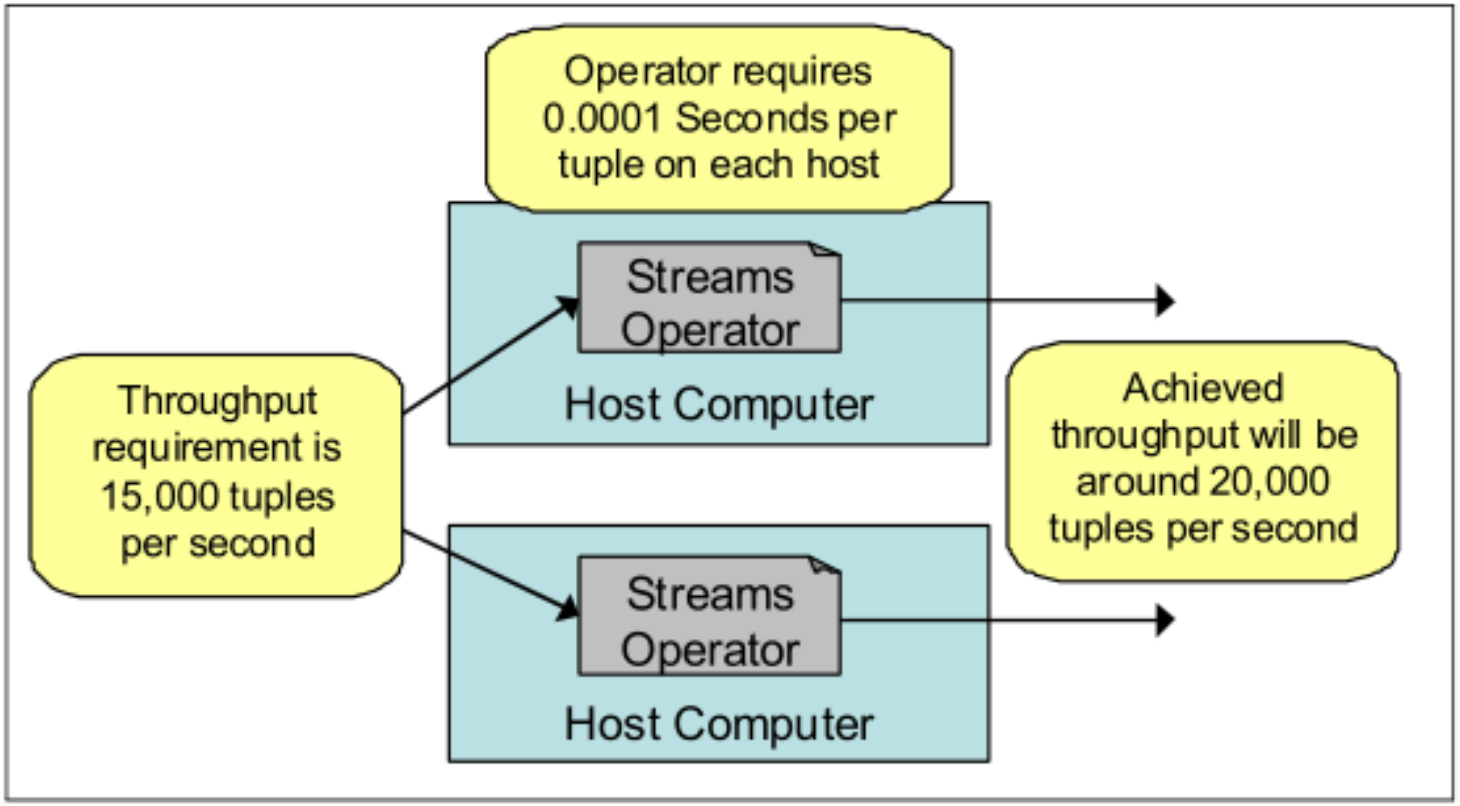


Forrás: [2], 3.2 alfejezet

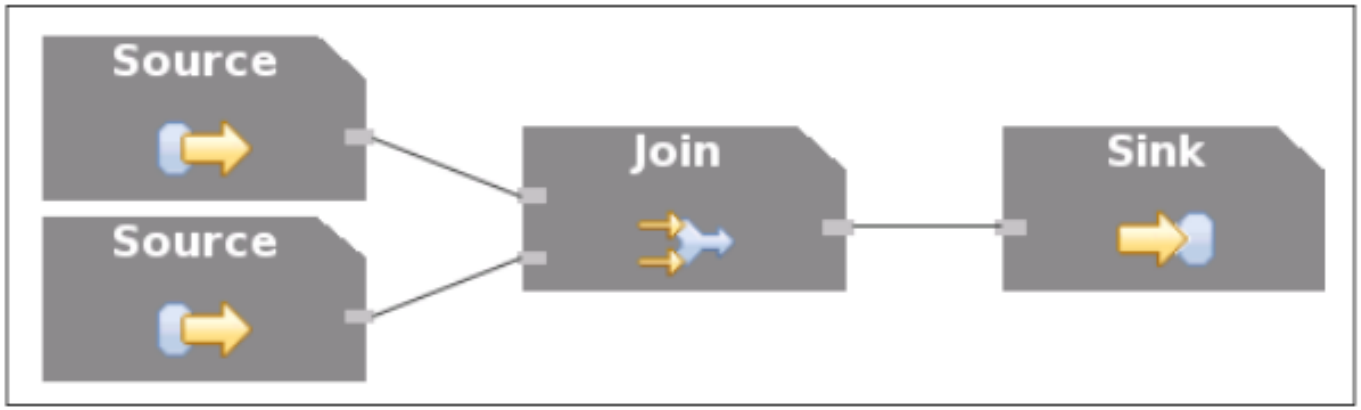
Tervezési minták: outliers



Tervezési minták: parallel

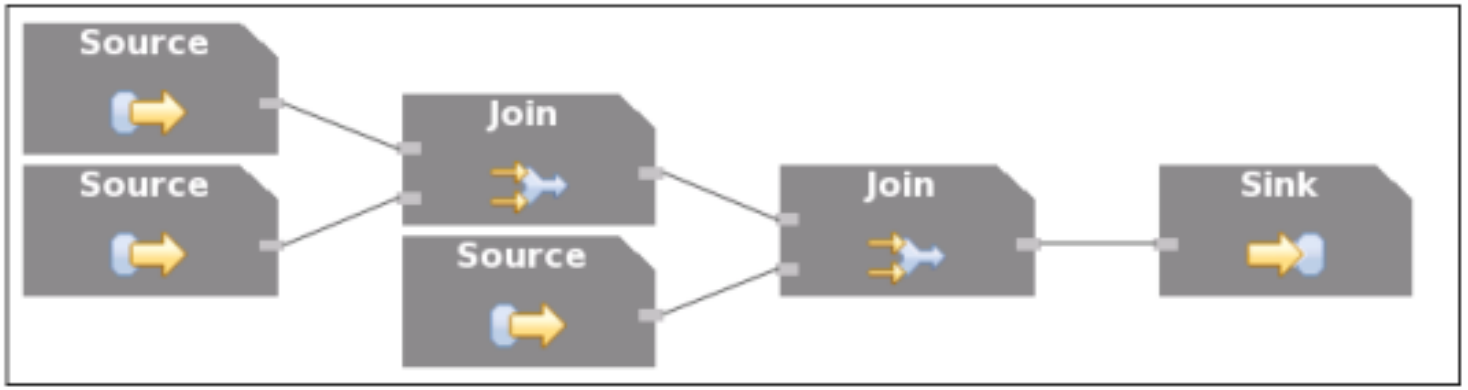


Tervezési minták: supplemental data



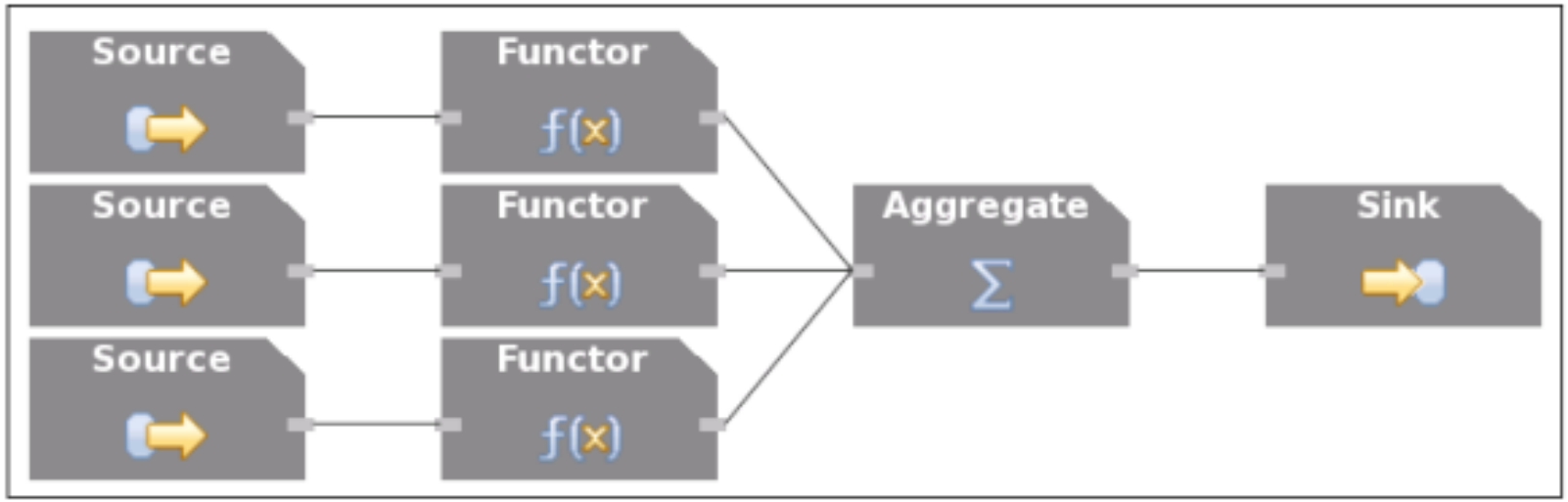
Lassan/nem változó referenciaadatokkal “dúsítás”

Tervezési minták: consolidation



Pl. “időablakos join”

Tervezési minták: merge



Azonos információ, (formailag) különböző folyamok

ALGORITMIKAI SZEMELVÉNYEK

Folyam-algoritmikai szemelvények

- A számítási modellt láttuk
- Fő korlát: adott tár + WCET, „be nem látott” adat

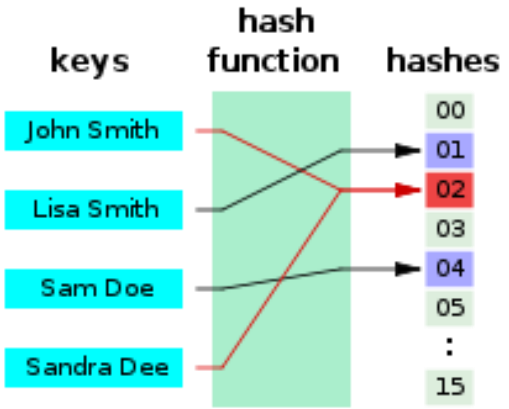
- Néhány tipikus probléma
  - Mintavételezett kulcstér, kulcsok minden értéke
  - „Elég jó” halmazba tartozás-szűrés kicsi leíróval
  - „Count distinct” *korlátos tárral*
  - Momentumok

- Részletes tárgyalás: [1] 4. fejezete

Kitérő: hash-függvények

- Cél: *U* nem rendezett univerzum elemein (átlagosan) gyors keresés, beszúrás, törlés, módosítás

- Eszköz: *h* hash függvény, ami rekordhoz logikai címet rendel
  - A címtartomány jellemzően sokkal kisebb, mint $|U|$
  - Ütközések: $K \neq K' \nRightarrow h(K) \neq h(K')$
  - Vödrös hash-elés, …



http://en.wikipedia.org/wiki/Hash_function

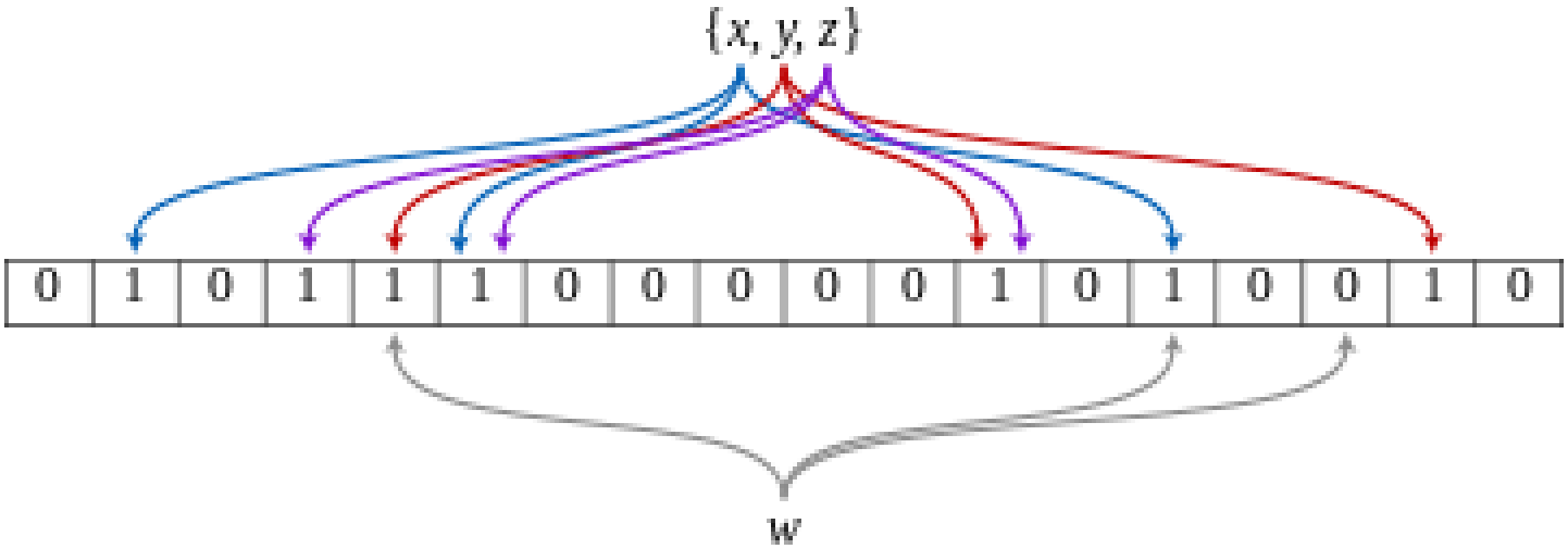
Hash-függvények: jellemző követelmények

- Alkalmazási területenként eltérőek!
  - Kriptográfia ←→ indexelés adattároláshoz
- Néhány tipikus követelmény
  - Determinizmus
  - Uniformitás
  - Meghatározott értékkészlet
  - Folytonosság
  - Irreverzibilitás („egyirányú” függvény)
- http://en.wikipedia.org/wiki/List_of_hash_functions

Mintavételezés

- Modell:
  - *n* komponensű elemek
  - ezek egy része *key* (pl. **user**,query,time)
  - a kulcsok felett mintavételezünk
- Probléma
  - Egy kulcsnak vagy minden értéke megjelenjen, vagy egy sem
- Megoldás
  - *a/b* méretű mintához *a* (kulcstér)méretű folyamon a kulcsot *b* vödörbe hasheljük
  - A hash-függvény valójában „konzisztens random-generátor”: a < b esetén “tárolunk” (továbbengedünk)
  - Nem véges minta – kisebb módosítás
- Példa: „a felhasználók mekkora része ismétel meg lekérdezéseket” a felhasználók 1/10 mintáján

Bloom filterek



http://en.wikipedia.org/wiki/Bloom_filter

Szűrés: Bloom filterek

- Bloom filter:
  - *n* bites vektor, kezdetben azonosan 0
  - Hash függvények kollekciója: $\mathrm{h}_1, h_2, \ldots h_k$. Mindegyik kulcsokat rendel *n* vödörhöz (a vektor elemeinek felelnek meg).
  - S: kulcshalmaz ($|S| = m$)

- Cél: minden $K \in S$ átengedése, a ***legtöbb*** $K \notin S$ kiszűrése – tárhely-hatékonyan

- Példa: spam email-cím alapján

Szűrés: Bloom filterek

- Indulás: minden j bit-et 1-re állítunk, amire van h_i és $K \in S,$ hogy $h_i(K) = j$

- Kulcs tesztelése: minden függvény eredménye 1 értékű bitbe visz-e
  - Igen: továbbengedés (lehet hogy S-ben)
  - Nem: dobás (nem lehet S-ben)

- Kaszkádolható!

- False positive valószínűség: lásd könyv (darts-modell)

Bloom filterek: néhány tétel

- Hibás pozitív valószínűség (uniform hashekkel):
  - $\approx (1 - e^{-km/n})^k$

- Optimális hashfüggvény-szám
  - $k = \frac{n}{m} ln2$

„Count-Distinct”: a Flajolet-Martin algoritmus

- N.B. nem-algoritmikai megoldások is működhetnek
- Legyen egy „bit-sztring” hash-függvénynek több kimenete, mint az univerzum elemei
  - Pl. 64 bit elég az URL-ekhez
- Ezekből “sokat” alkalmazunk
- $h(a)$ a folyam-elemre r 0-ban végződik (*tail length*). Legyen ezek maximuma R.
- Count-Distinct közelítés: 2^R
  - Ha $m \gg 2^r$, akkor szinte biztos van legalább r hosszú farok
  - Ha $m \ll 2^r$, akkor szinte biztos nincs legalább r hosszú farok
- Sok hash függvény, kis csoportok (legalább $c \log_2 m$) átlaga, ezek mediánja

„Count-Distinct”: a Flajolet-Martin algoritmus

- Intuitívan:
  - Egy a –ra $h(a)$ legalább r 0-ban végződik: 2^{-r} val. (Uniform hash azért nem árt.)
  - m különböző elem, egyikükre sem legalább r hosszú a tail:
    $$(1-2^{-r})^m$$
  - Átírható: $\left((1-2^{-r})^{2^r}\right)^{m2^{-r}}$
  - Elég nagy r-re a belső tag $\approx \frac{1}{e}$
  - Nincs elem legalább r hosszú tail-el: $e^{-m2^{-r}}$ val.
  - m jóval nagyobb, mint 2^r: szinte biztosan látunk legalább r hosszút
  - m jóval kisebb, mint 2^r: szinte biztosan nem látunk r hosszút
  - 2^R nem valószínű, hogy túl nagy vagy túl kicsi lenne
  - “Kiátlagolás” önmagában nem jó: túl nagy a felfelé “nagyot tévedő” 2^R-ek hatása
  - Medián sem jó: mindig 2 hatvány, $2^{x+1}-2^x$ egyre nagyobb, szoros becslést nem tud adni

Momentumok

- Rendezett univerzum
- m_i: i-ik elem előfordulási száma
- Stream k-adrendű momentuma (k-ik momentum): $\sum_i (m_i)^k$
- Néhány momentum
  - 0: „count distinct”
  - 1: stream hossza
  - 2: előfordulások négyzetösszege – *surprise number*: eloszlás egyenetlensége
    - V.ö. $I(\omega_n) = -\log(P(\omega_n))$

Az Alon-Matias-Szegedy algoritmus

- Legyen a stream n hosszú,
- Nem tudunk minden m_i-t tárolni,
- Második momentum közelítése,
- Korlátos tárhellyel (több → jobb közelítés)
  - Elemei "változók"

- Minden X változónkhoz tárolni fogjuk:
  - Az univerzum egy elemét: $X.element$
  - Egy $X.value$ egészet.

- Inicializálás: uniform, véletlenszerű választással 1 és n között pozíciót rendelünk minden változóhoz.
- $X.element$: amit a pozíciókban találunk
- $X.value$: előfordulások száma a sorsolt kezdőpozíciótól

Az Alon-Matias-Szegedy algoritmus

- Minden X-ből lehet becsülni:

$$\mathrm{n} \times (2 \times X.value - 1)$$

- Legyen $e(i)$ a stream i-ik eleme; legyen $c(i)$ ezen elem előfordulási száma az i-ik pozíciótól. Első lépés: a kifejezés várhatóértéke az uniform módon választott pozíciókon egy átlag.

$$E(n(2 \times X.value - 1)) =$$

$$\frac{1}{n}\sum_{i=1}^{n} n \times (2 \times c(i) - 1) =$$

$$\sum_{i=1}^{n} (2c(i) - 1)$$

Az Alon-Matias-Szegedy algoritmus

- Tekintsünk egy *a* elemet, ami m_a-szor fordul elő.
- $\sum_{i=1}^{n}(2c(i)-1)$-ben a kontribúciói “hátulról”:
- Utolsó előfordulása: 2x1 – 1 = 1
- Azelőtti: 2x2 – 1 = 3
- Összes: $\sum 1+3+5+\cdots+2(m_a-1)$

- $\sum_{i=1}^{n}(2c(i)-1)$ szétbontható ilyen csoportokra!
- Akkor

$$E(n(2\times X.value-1))=\sum_{a}1+3+5+\cdots+2(m_a-1)$$

Az Alon-Matias-Szegedy algoritmus

- Indukcióval: $1 + 3 + 5 + \cdots + (2m_a - 1) = (m_a)^2$
- Vagy egyszerűen $\sum_{x=1}^{m_a}(2x - 1)$ átrendezés

- Így:

$$E(2 \times X.value + 1) = \sum_a (m_a)^2$$

Az Alon-Matias-Szegedy algoritmus

- k-ik momentumra:

$$v = X.value \rightarrow n \times (v^k - (v-1)^k)$$

- Kiterjesztés nem véges stream-ekre:
  - Nem a tárolás, hanem a korai elemekkel elfogultság a probléma
  - Mindig s változót tárolunk, inicializáció
  - Minden új elemet $\frac{s}{n+1}$ valószínűséggel választunk változónak
  - Ha választjuk, egy régit eldobunk
  - Megmutatható: így a pozíció-kiválasztási valószínűség egyenletes eloszlású marad

Hivatkozások

- [1] Rajaraman, A., & Ullman, J. D. (2011). Mining of Massive Datasets. Cambridge: Cambridge University Press. doi:10.1017/CBO9781139058452
- [2] International Technical Support Organization. IBM InfoSphere Streams: Harnessing Data in Motion. September 2010. http://www.redbooks.ibm.com/abstracts/sg247865.html
- [3] http://hortonworks.com/blog/hdp-2-0-community-preview-and-launch-of-hortonworks-certification-program-for-apache-hadoop-yarn/
- [4] http://www.ibm.com/developerworks/library/bd-streamsrtoolkit/
- [5] http://storm.apache.org/talksAndVideos.html