Outlierdetektálás nagyméretű adathalmazokon

Salánki Ágnes
salanki@mit.bme.hu

Budapest University of Technology and Economics
Fault Tolerant Systems Research Group

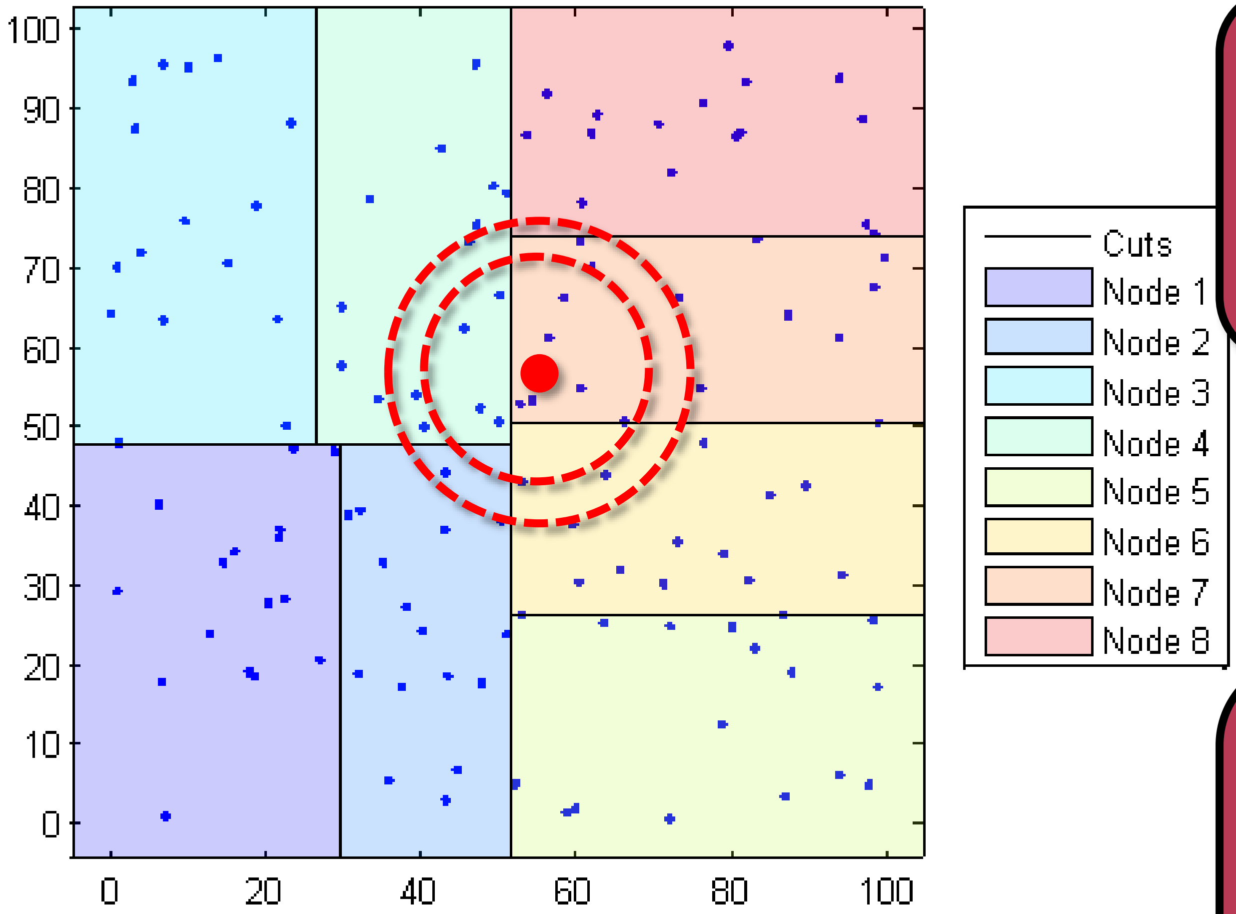
Hol tartunk?

- Eddig:
  - Outlier detektáló módszerek
    - DB, LOF, BACON stb.
- Most:
  - Hol segíthet a MapReduce az outlier detektálásban?
  - Adatfolyamokon

Implementációs kérdések

- Az alap mindig valamilyen távolságszámítás
- $kNN(x_i), NN(x_i, r')$ – milyen adatszerkezettel?
- Naiv
  - Távolságmátrixot tárolunk
  - $sort(x)[k], which(x \ \leq r')$
- Partíciós módszerek?
  - Pl. fák: k-d tree?

Implementációs kérdések



2. KERES: $k = 10$
2.1 $r' \leq 9$ (7. zóna)
2.2 r' = 7 (5-8. zóna)

1. ÉPÍT
Hierarchikus adatszerkezetben a közeli ponthalmazok

Implementációs kérdések

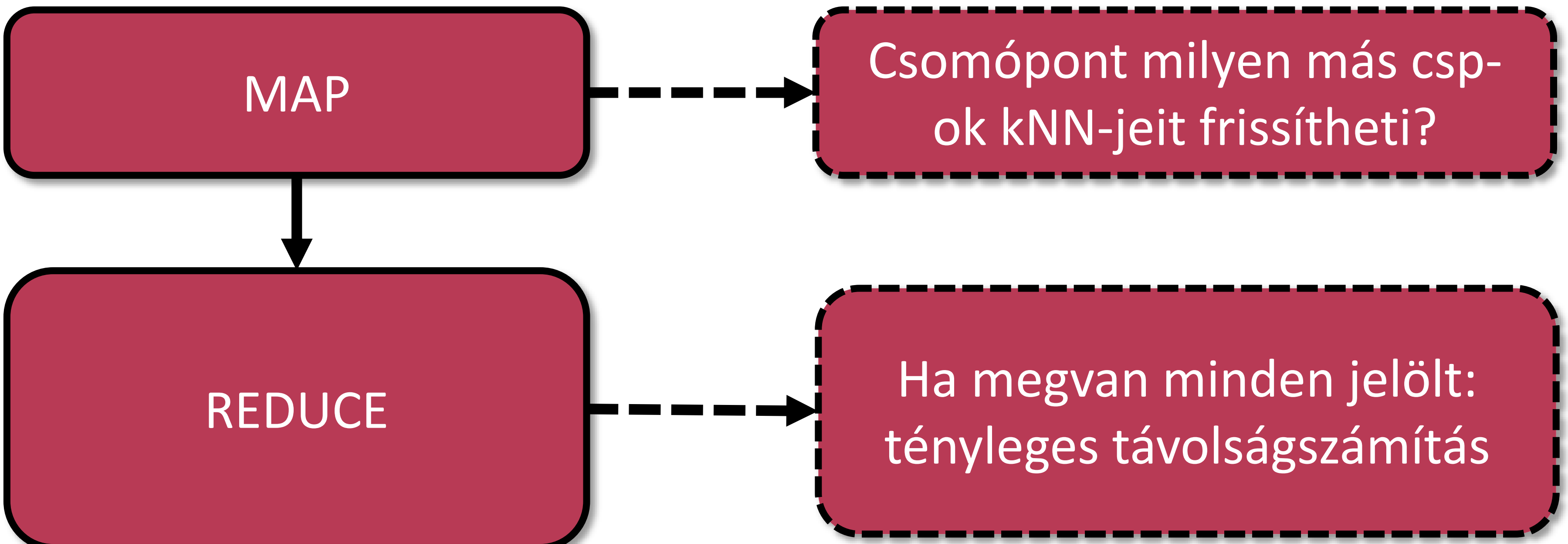
2. KERES: $k = 10$
2.1 $r' \leq 9$ (7. zóna)
2.2 r’ = 7 (5-8. zóna)

Nem kell mindent kiszámolni ☺
Többször kell kiszámolnunk ugyanazt ☹

adatszerkezetben a
közeli ponthalmazok

Map-Reduce?

- n elég nagy → muszáj bontani



Map-Reduce?

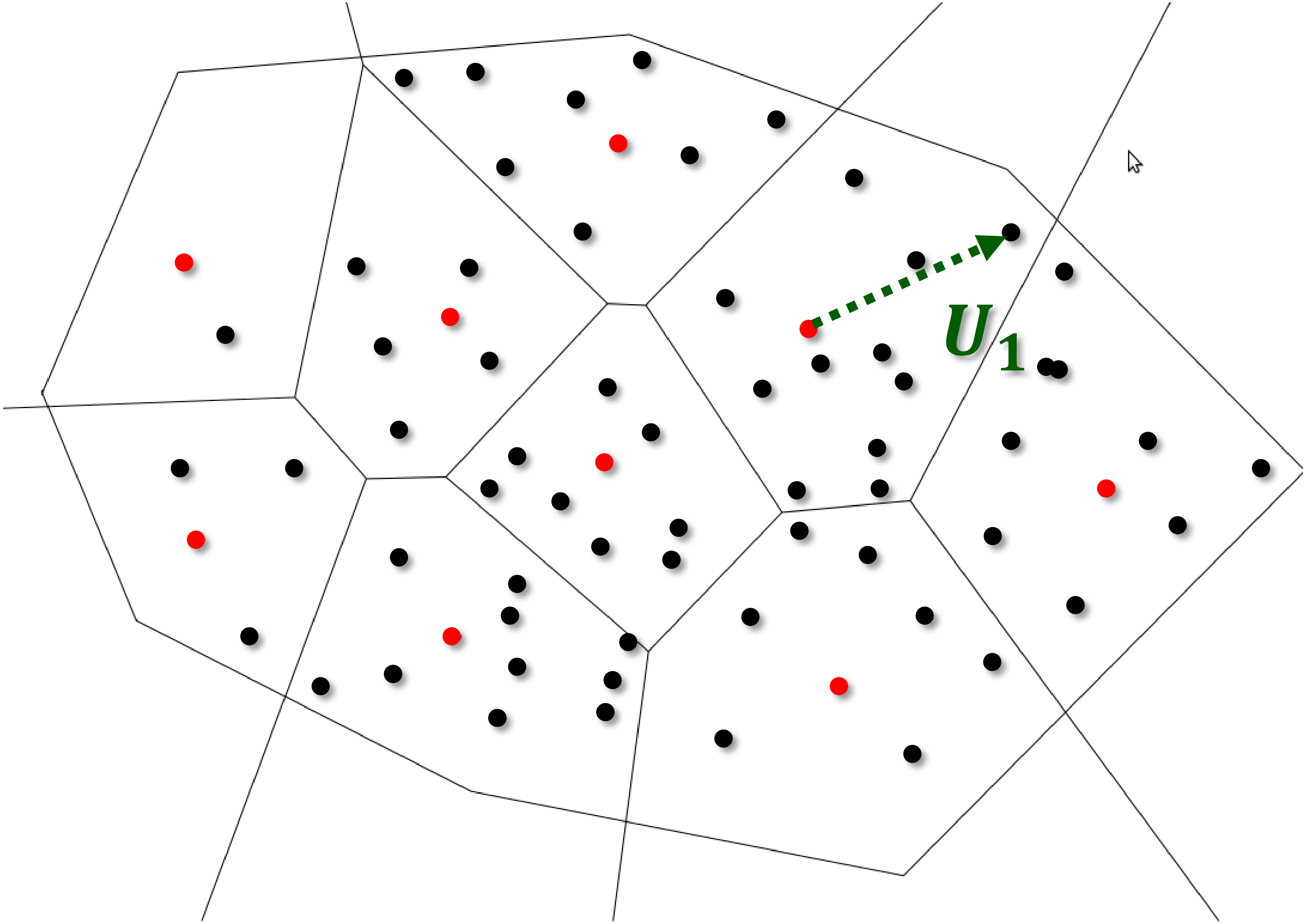
- n elég nagy → muszáj bontani

Mi van, ha már a felosztást is elosztottan akarom végezni?

REDUCE

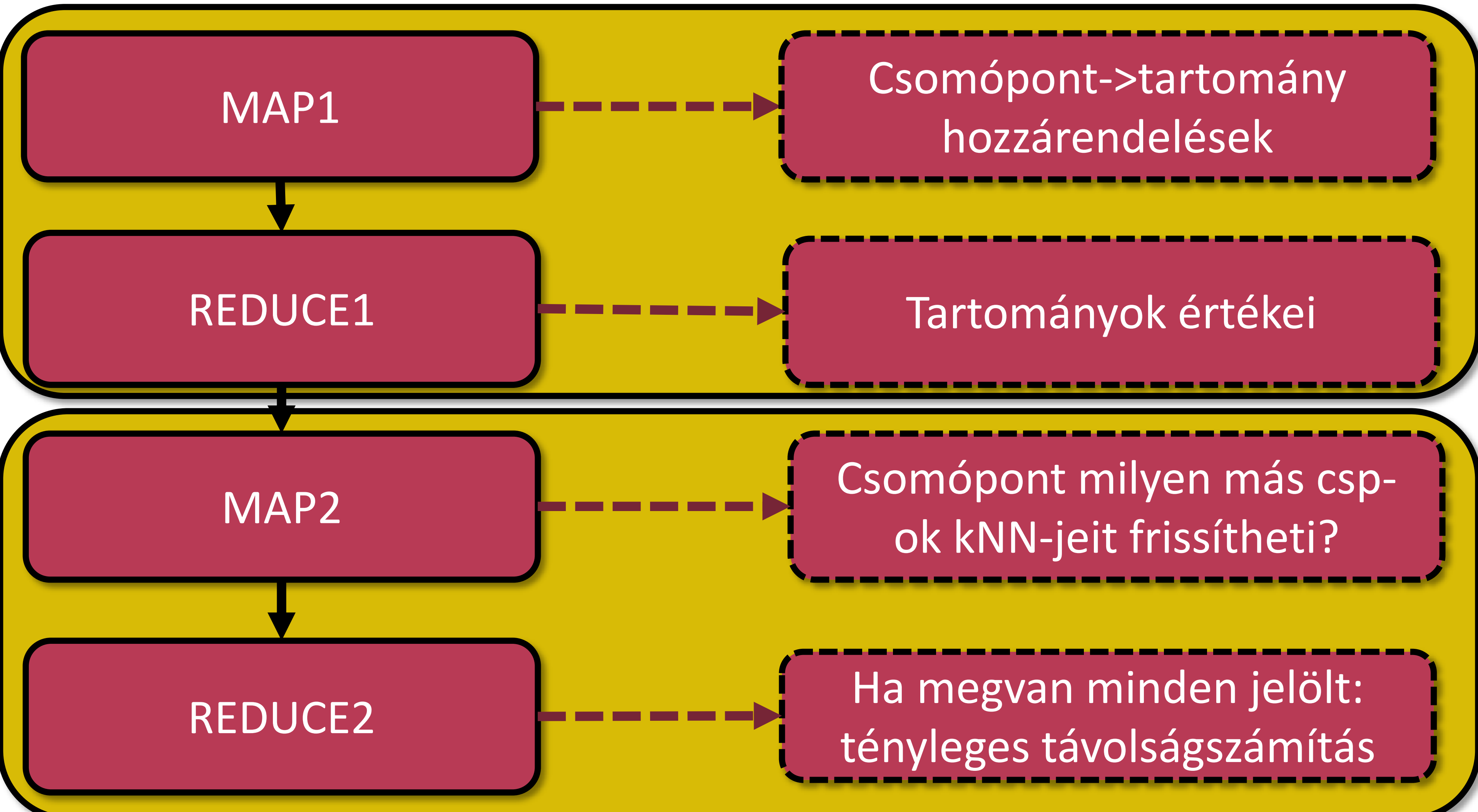
Ha megvan minden jelölt: tényleges távolságszámítás

Egy kis csalás.. Voronoi cellák



Amiért jó: MapReduce ☺

- n elég nagy → muszáj bontani

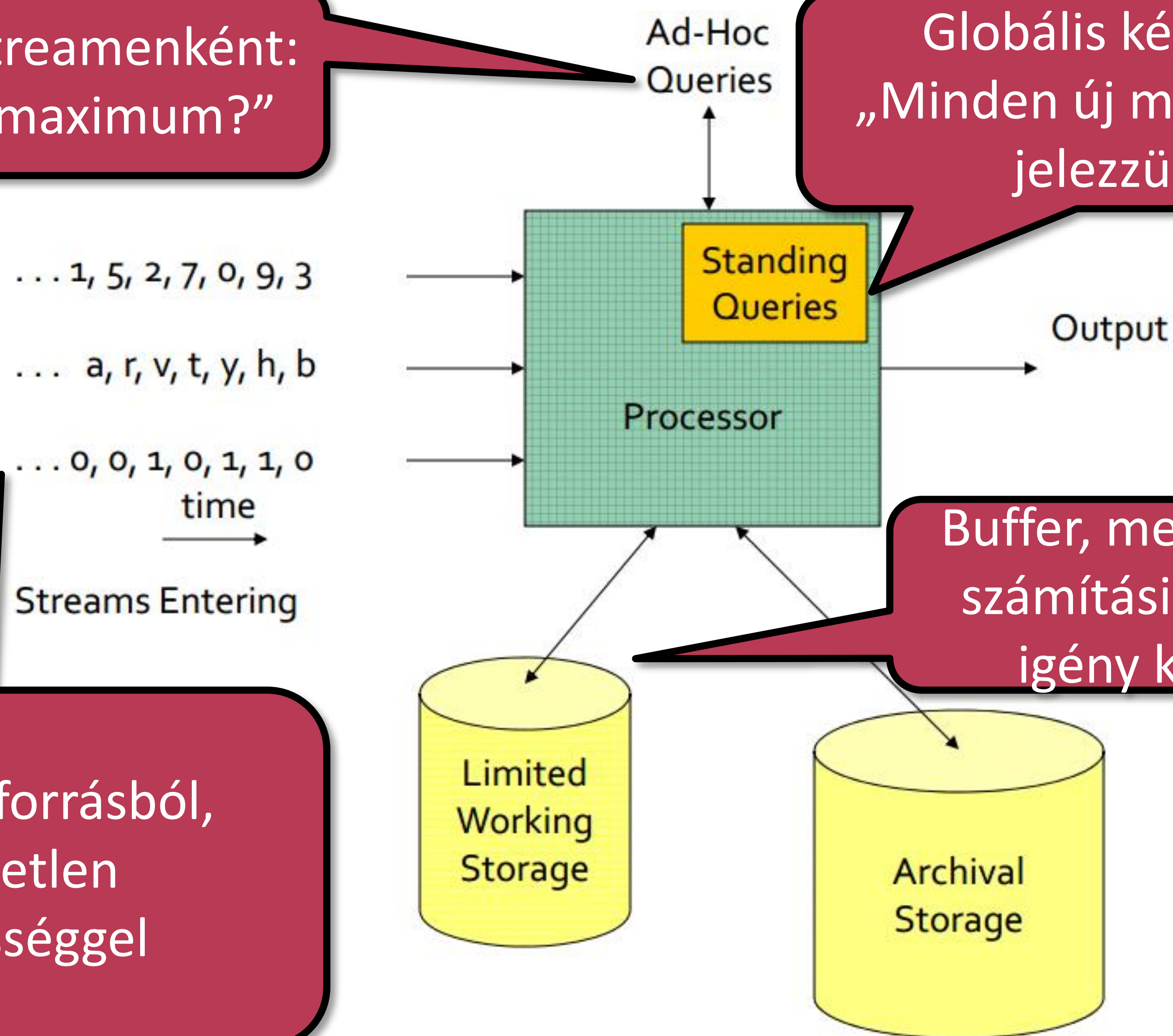


OUTLIEREK ADATFOLYAMOKBAN

Adatfolyamok

Egyszer streamenként:
„Lokális maximum?”

Globális kérdések:
„Minden új maximumot jelezzünk”



Buffer, megengedett számítási memória igény korlátos

1. több forrásból,
2. ismeretlen sebességgel

Ábra és a számértékes példák forrása: [1]

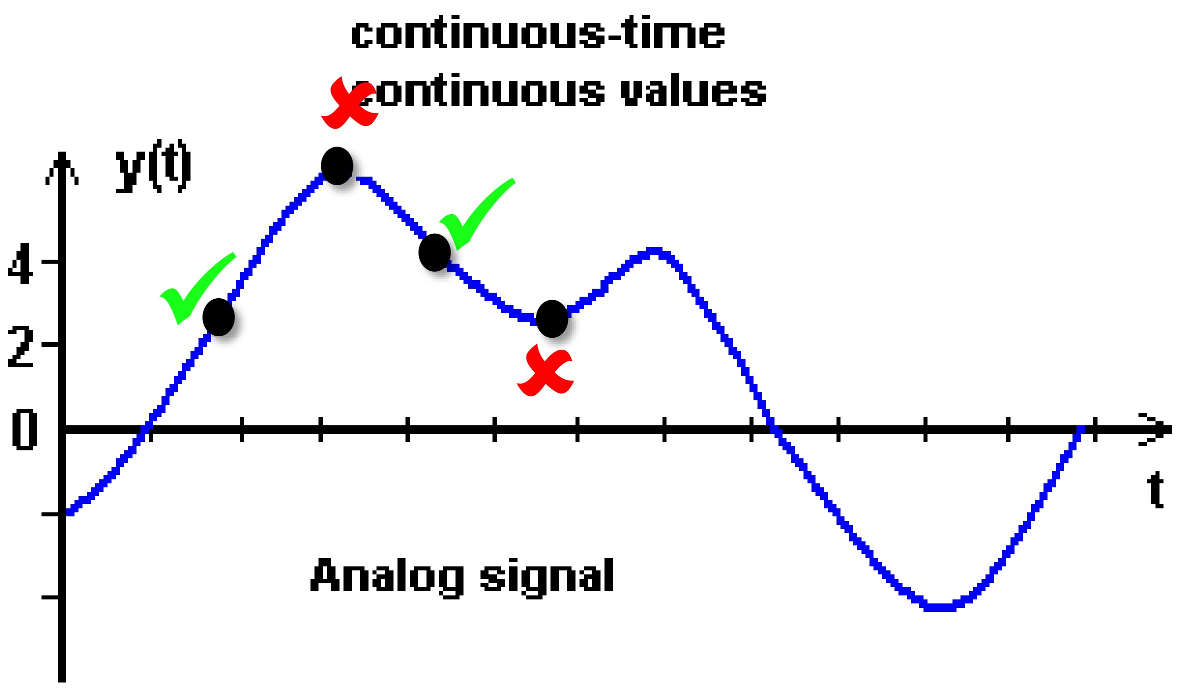
Kitérő: outlierek idősorokban

- IT Monitorozás
- Tőzsdei elemzések
- Banki csalásfelderítés

- Mindkét adattípus számít
  - Szenzorok: nagyrészt numerikus
    - CPU_nice: $0.12, 0.13, 0.12, 0.13, \ldots$
  - Naplózás: nagyrészt kategorikus
    - $VM\_operations$: Start, Stop, Start, Snapshot, Snapshot, …

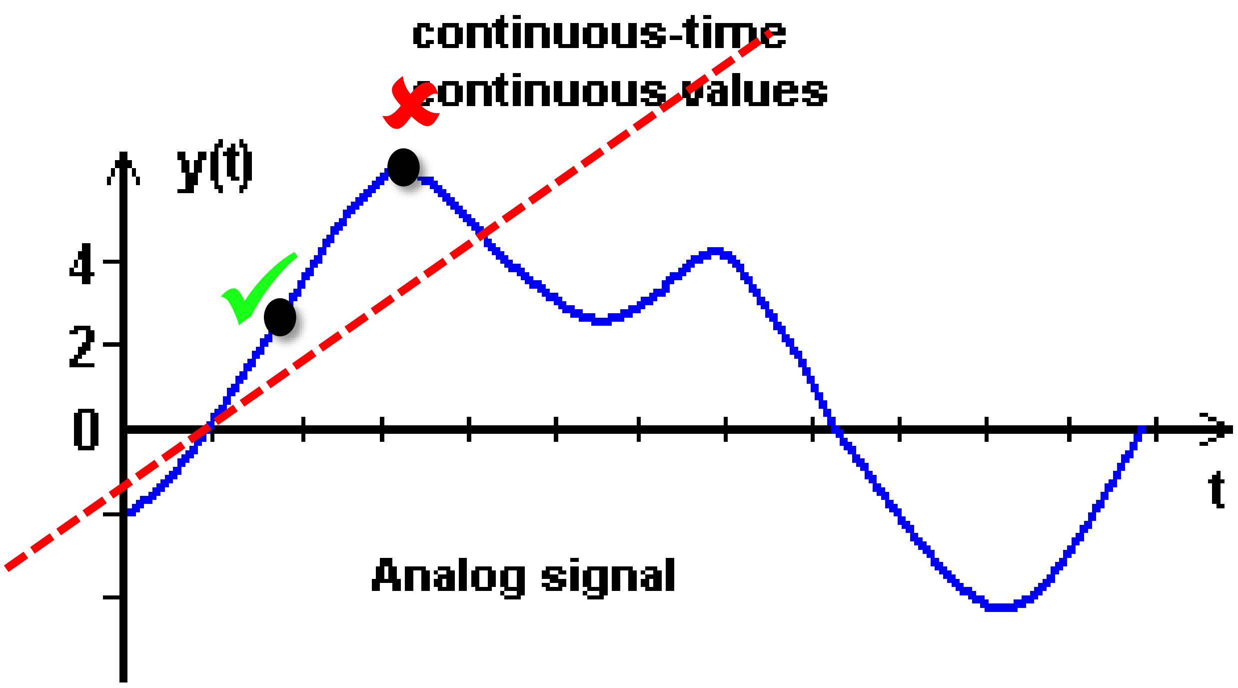
Outlierek *szekvenciákban*

- A legkiugróbb pont megtalálása
  - $abs(T[k] - mean(T[k-l], \ldots, T[k+l]))$: max



Outlierek *szekvenciákban*

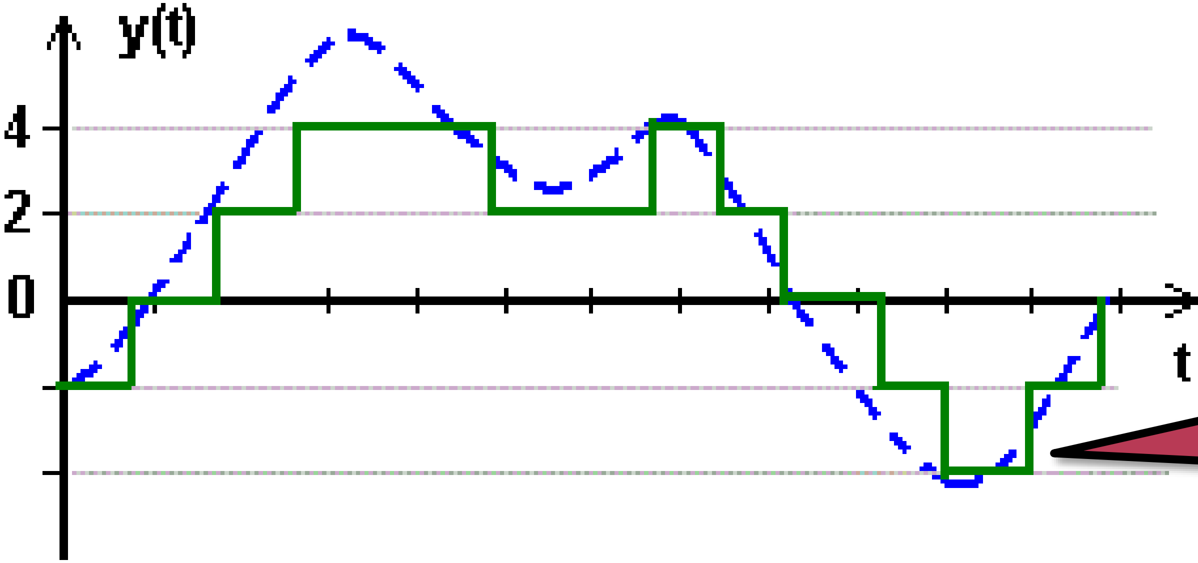
- A legkiugróbb pont megtalálása
  - $abs(T[k] - mean(T[k - l], \ldots, T[k + l]))$: max
  - Square Error regresszióból: min



Outlierek *szekvenciákban*

- A legkiugróbb pont megtalálása
  - $abs(T[k] - mean(T[k-l], \ldots, T[k+l]))$: max
  - Square Error regresszióból: min
  - A pont törlésével a „minimum description length” a lehető legjobban lecsökken.

Eredeti: 5 különböző érték



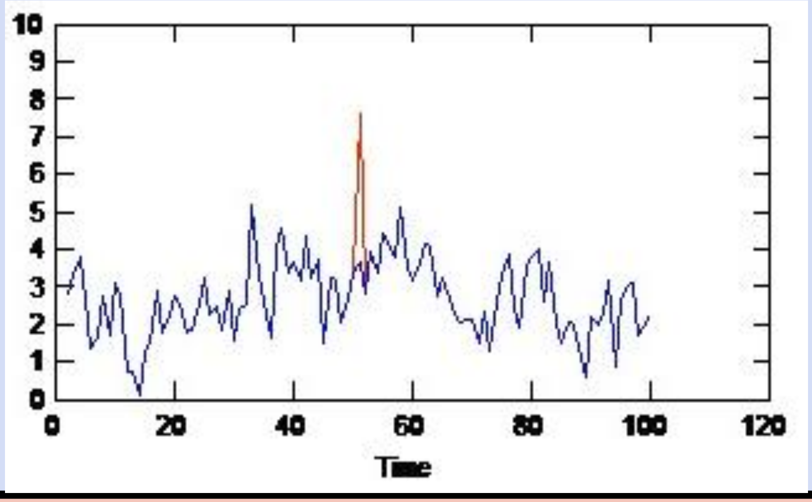
-2 törlése után: 4 különböző érték is elég

Hatások szerinti osztályozás

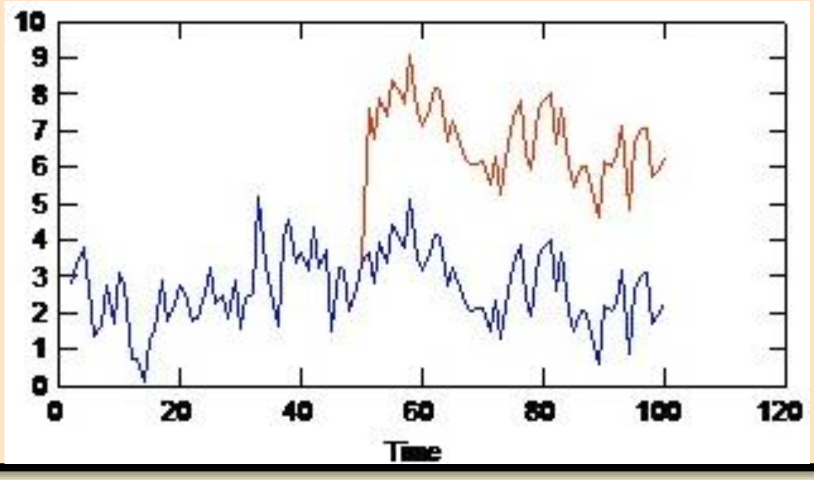
- Additive outlier
  - A rákövetkező elemekre teljesen hatástalan
- Level Shift Outlier
  - Permanens hatás
- Innovational Outlier
  - Kezdeti hatás + lecsengés, az ismétlések számával ez erősödhet
- Transient Change Outlier
  - ~Innovational outlier, de exponenciálisan lecseng a hatás, később visszatér normálra

Hatások szerinti osztályozás

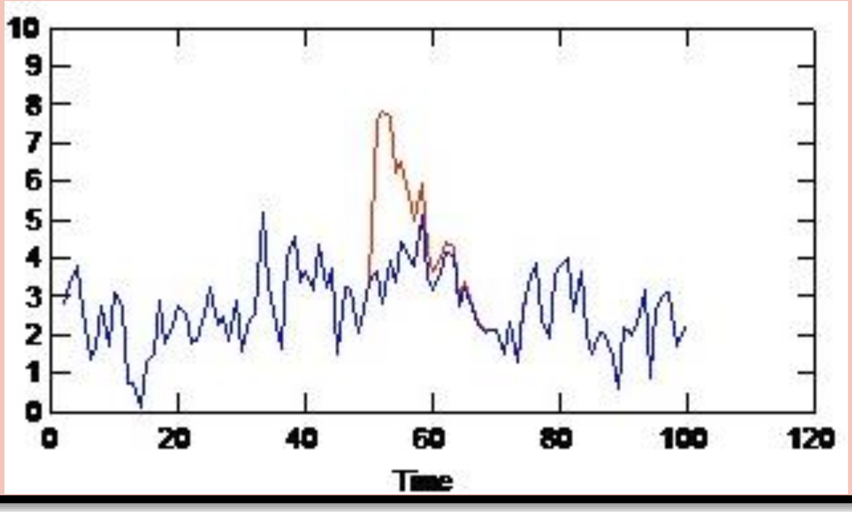
Additive



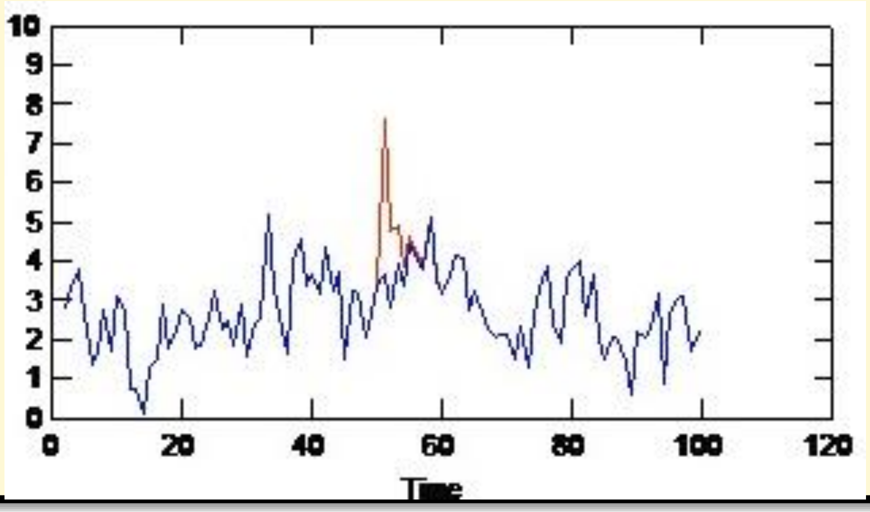
Level Shift



Innovational

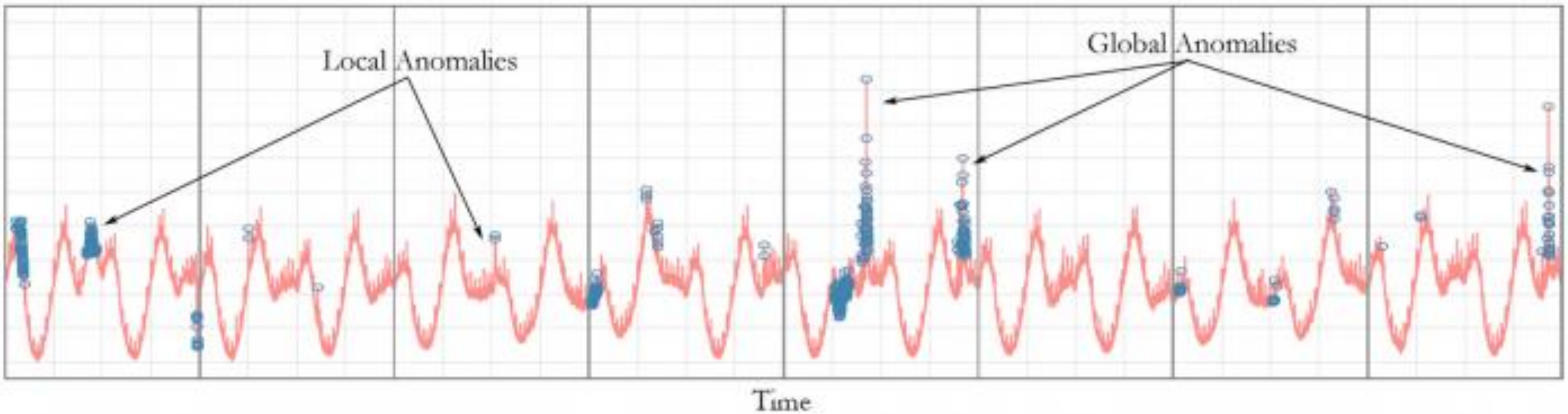


Transient change



Additive és level-shift outlierek a Twitternél

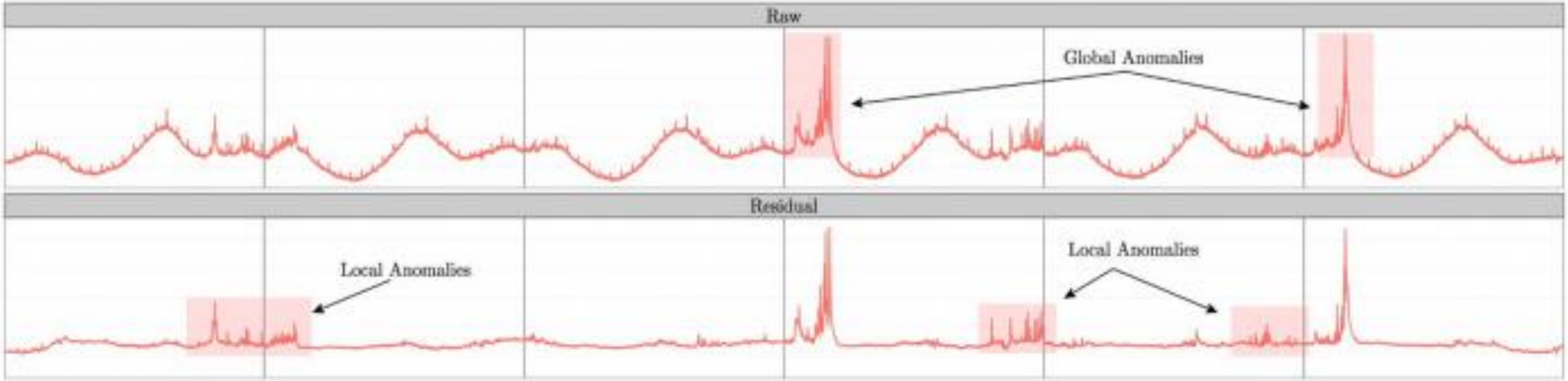
- Globális és lokális megkülönböztetése



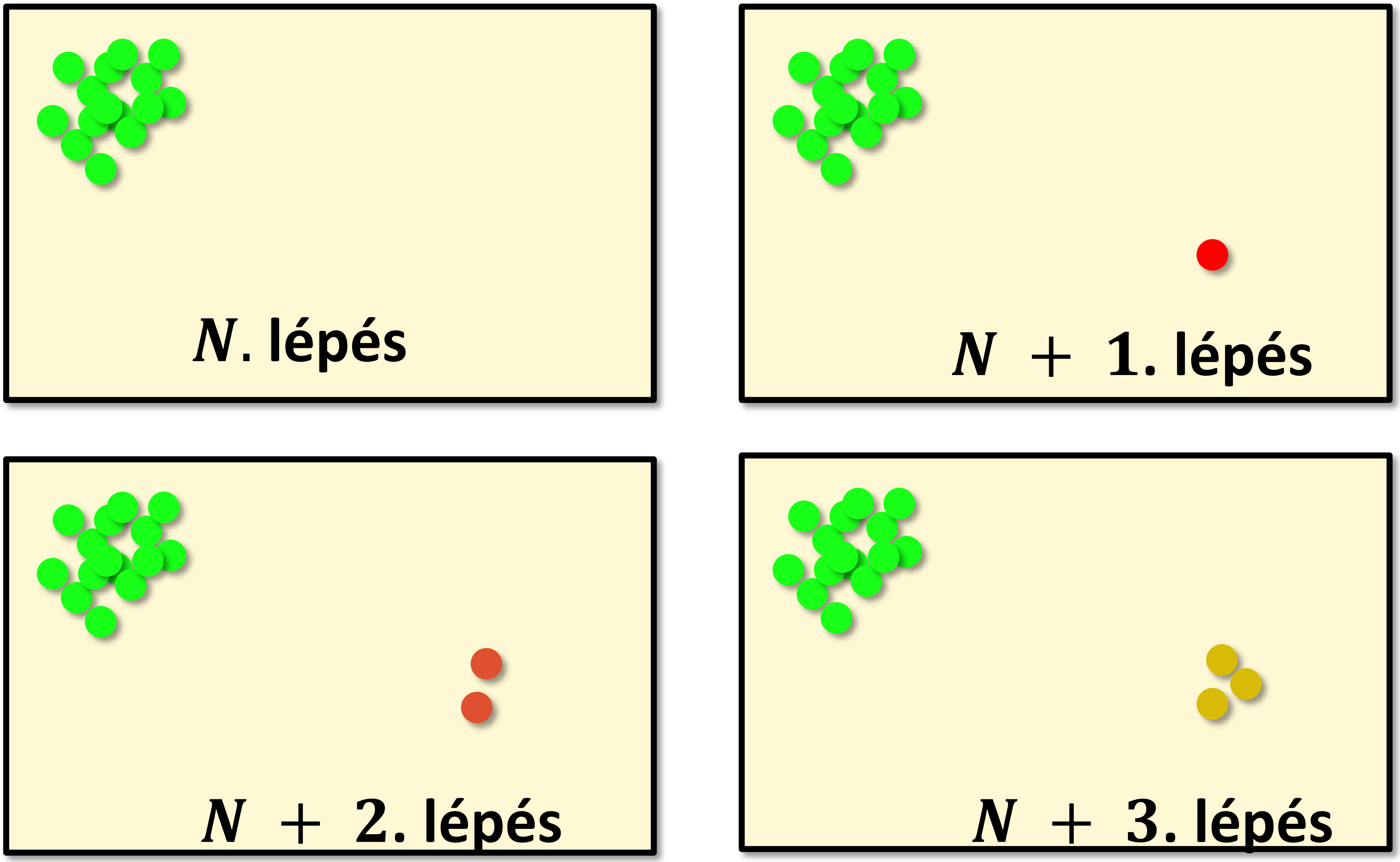
- Alapötlet:
  - A globálisak látszanak a robusztus statisztikák kiszámítása után
  - A lokálisak látszanak a “maradékból” (idősor – trend – szezonalitás stb.)

Additive és level-shift outlierek a Twitternél

- Pozitív outlierek: kapacitástervezéshez
- Negatív outlierek: HW vagy adatgyűjtési hibák felderítéséhez



“Elvárt viselkedés” streameken



Offline algoritmusok közvetlen adaptálása

- Periodikus
  - Minden n. adatpont után futtassuk le az X algoritmust
  - Probléma: $x_n - t$ nem tudjuk jelezni
- Iterált
  - Minden lépésben újrafuttatjuk az X algoritmust
  - Probléma: lassú ☹
- “Felügyelt”
  - Az elején kiszámítjuk a “normál” működést, aztán mindent ahhoz viszonyítunk
  - Probléma: az x_{n+3} is outlier lesz, hiszen a normál működést nem frissítjük

Storm

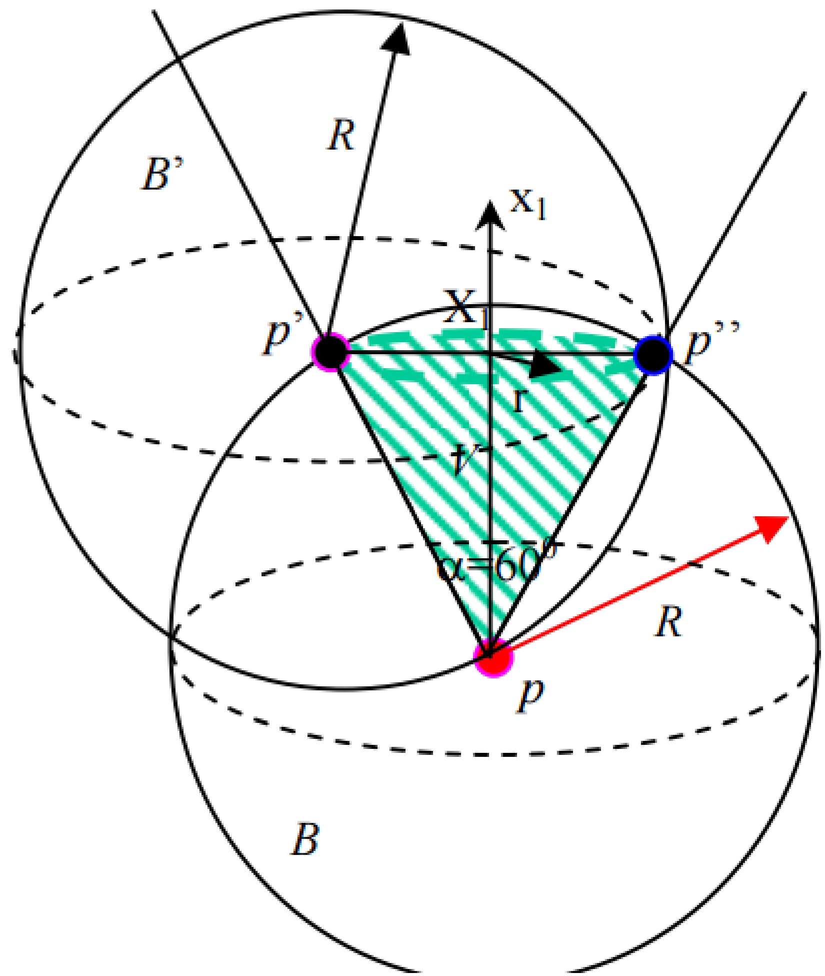
- STream OutlieR Miner: DB egyfajta streamesítése
- Lekérdezés: “Kérem az adott ablakban talált outliereket”
- Alapötlet
  - Minden pontot kategorizáljunk a beérkezése pillanatában, később esetleg tartsuk karban az értékeit
  - Háromféle csomópont típus
    - “safe inlier”: már a bekerülése pillanatában elég szomszédja van
    - “inlier”: a bekerülése után még jöttek hasonló pontok
    - “outlier”: a “lejárati idejéig” sem jött elég szomszédja

Storm

- Exact-Storm
  - Minden iterációban
    - Kiszámoljuk az új elem szomszédságát
    - Ezek alapján frissítjük a régiek értékeit
- Approximate-Storm
  - Nem tároljuk el az összes safe inliert
  - Nem tároljuk el az összes szomszédot
  - Még így is határon belül tudunk becsülni..

Inkrementális LOF

- Közelítjük a kNN listát
- Alapötlet:
  sokdimenziós geometria



Outlierek szekvenciák *között*

- „Az aggregált adatokon látjuk, hogy baj van. Pontosan a rendszer melyik komponense hibás?”

- Feltételezések
  - Az idősorok hossza azonos
  - Keressük a legkiugróbbat

Outlierek szekvenciák *között*

- Ötletek
  - Képezzük le egy értékre az idősort/idősor párokat
  - Elemek egy hasonlósági mátrixba
- Innentől már akármelyik klasszikus klaszterezési módszer működik

- Távolságfüggvény a szomszédossághoz?

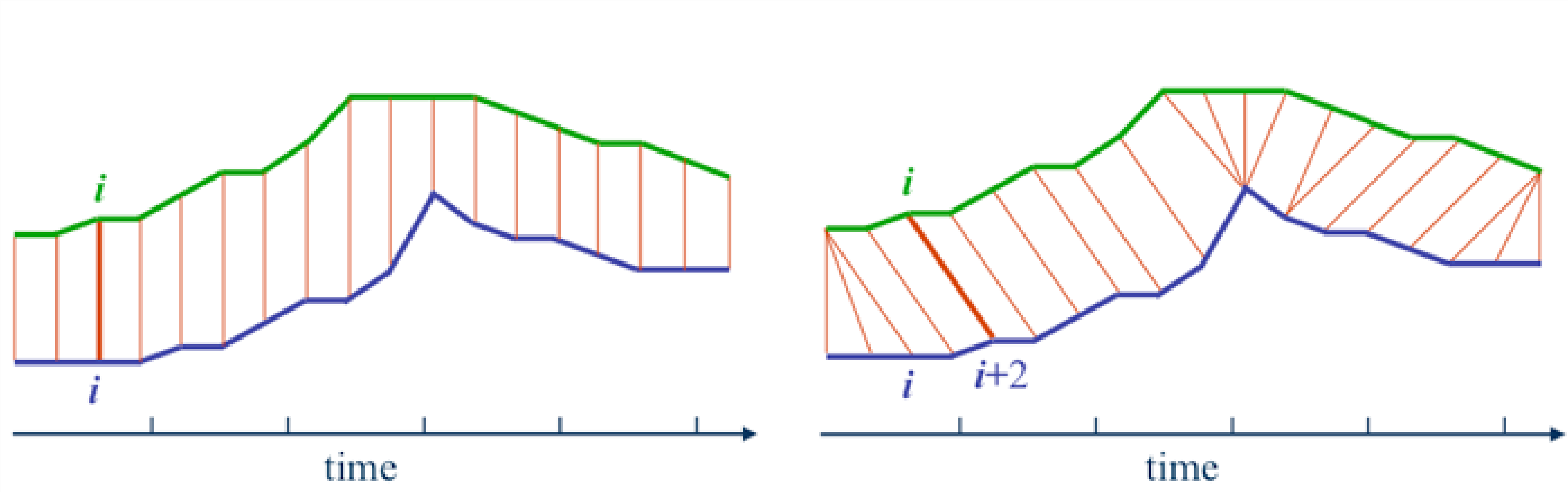
Idősorok távolságfüggvényei

- Euklideszi távolság
  - X tengely menti eltolás (offset)?

- Lehetséges megoldások:
  - Dynamic time warping
    - eleve kiugró értékek alapján hasonlítunk
  - Length of common subsequence

Dinamikus idővetemítés

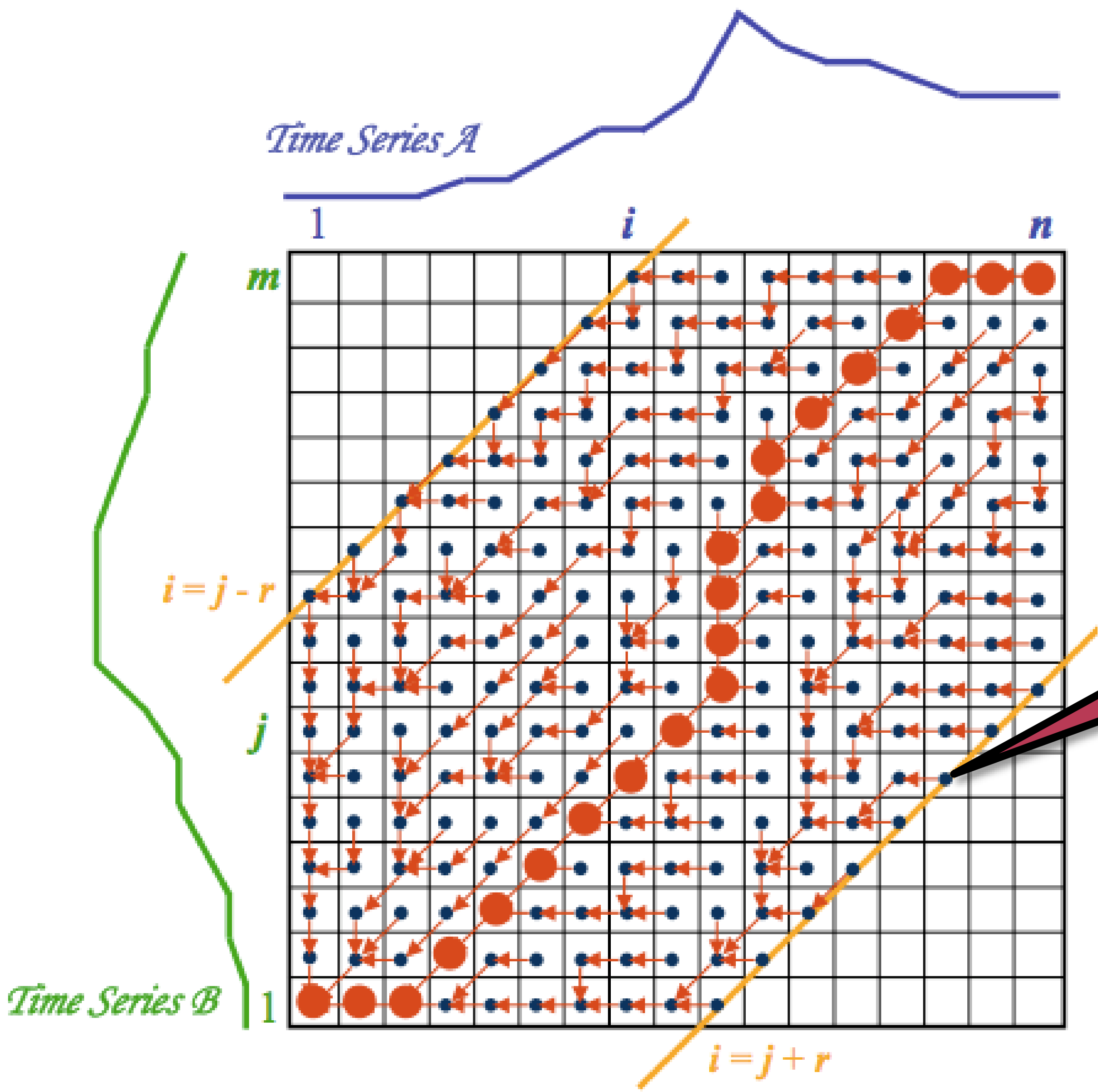
- Az idősorok pontjait nem indexenként hasonlítjuk össze
  - Motiváció pl. hangfelismerésnél

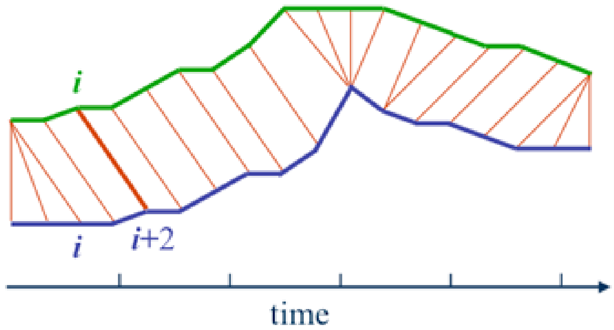


Dinamikus idővetemítés számítása

1. $n \times m$-es D mátrixban rögzítjük a sorok egymástól való távolságát
2. Kell: $p = [p_1, p_2, \dots p_k]$ útvonal a $D[1,1]$ és $D[n,m]$ között
3. Cél: minimális költség
4. Szabályok:
    1. Minden lépésben előre haladunk (nem távolodhatunk, tehát $[i,j] \rightarrow [\tilde{\imath},\tilde{\jmath}]$ esetén $\tilde{\imath} \geq i, \tilde{\jmath} \geq j$)
    2. Az út folytonos, mindig csak szomszédos cellákra léphetünk

Dinamikus idővetemítés





Sakoe-Chiba sáv

Longest common subsequence

- Nem a pontos időpont számít
- Csak a sorrend

$x_1: abcdefg$

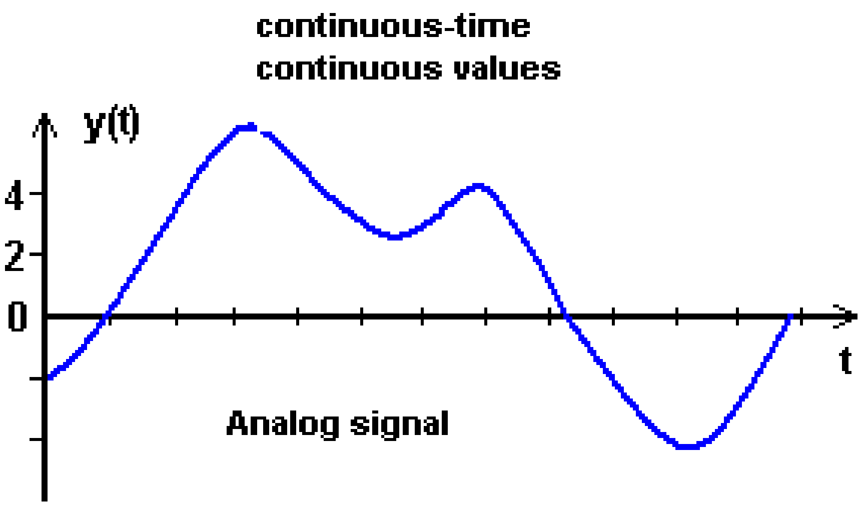
$x_2: fabdceg$

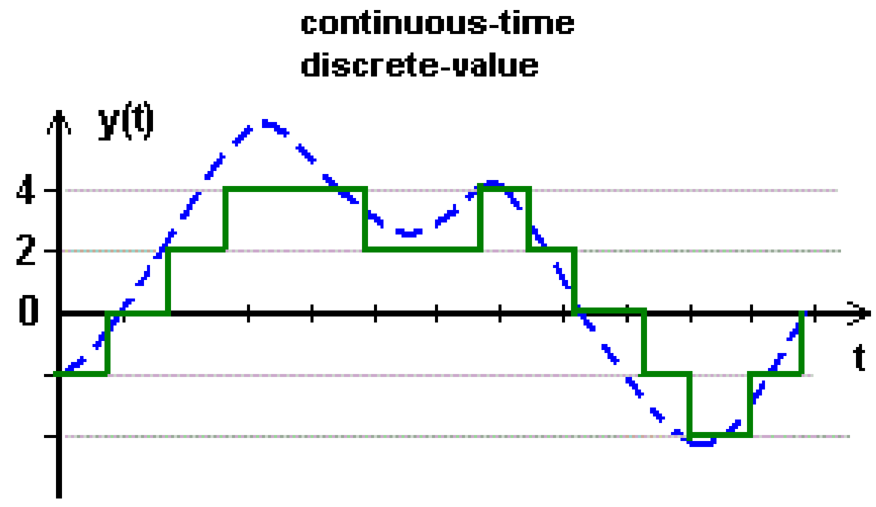
$lcs(x_1, x_2): abceg$

$nlcs(x_1, x_2) = 5$

Longest common subsequence

- Nem a pontos időpont számít
- Csak a sorrend
- Általánosítás folytonos értékekre





Hivatkozásjegyzék

- [1] Inkrementális LOF
  - Pokrajac, Dragoljub, Aleksandar Lazarevic, and Longin Jan Latecki. "Incremental local outlier detection for data streams." *Computational Intelligence and Data Mining, 2007. CIDM 2007. IEEE Symposium on*. IEEE, 2007.
- [2] Hatás szerinti outlier detektálás idősorokban
  - http://www-01.ibm.com/support/knowledgecenter/SS3RA7_15.0.0/com.ibm.spss.modeler.help/ts_outliers_overview.htm

Hivatkozásjegyzék

- Exact-Storm
  - Fabrizio Angiulli and Fabio Fassetti. Detecting distance-based outliers in streams of data. CIKM '07
- Twitter BreakoutDetection package: https://blog.twitter.com/2015/introducing-practical-and-robust-anomaly-detection-in-a-time-series