BIG DATA

Spracovanie „veľkých“ dát

Peter Bednár

Úvod

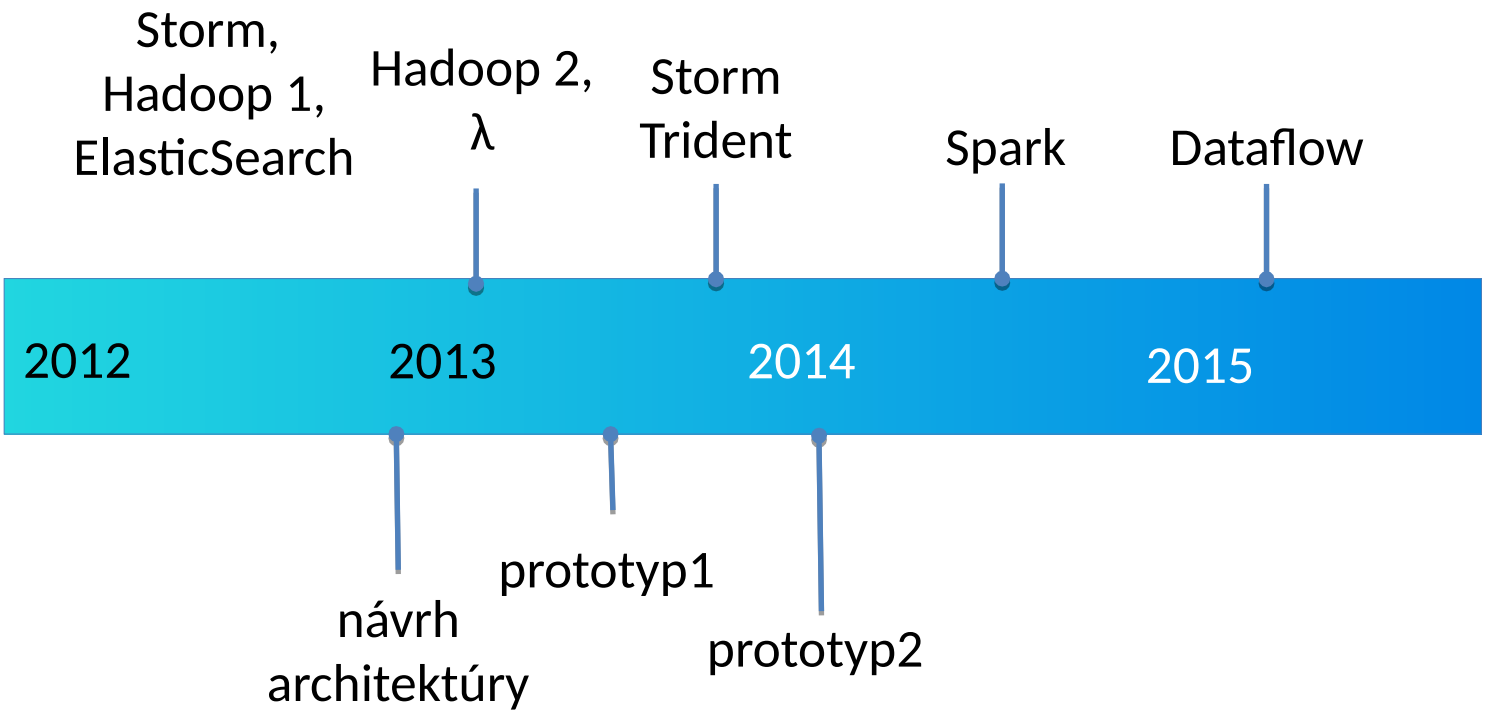
Prednáška sa bude zaoberať problematikou spracovania veľkých dát. Na začiatku uvedieme ~~charakteristiku veľkých dát~~ a niektoré prípadové štúdie. Ďalej sa budeme zaoberať horizontálnym škálovaním aplikácií a postupne uvedieme problematiku ~~paralelných~~ a distribuovaných výpočtov, spracovania prúdových dát a distribuovaných databáz. Prednáška je doplnená prehľadom technológií a podrobnejším popisom architektúry projektu *Urban Sensing pre spracovanie dát zo sociálnych sietí.

*Urban Sensing

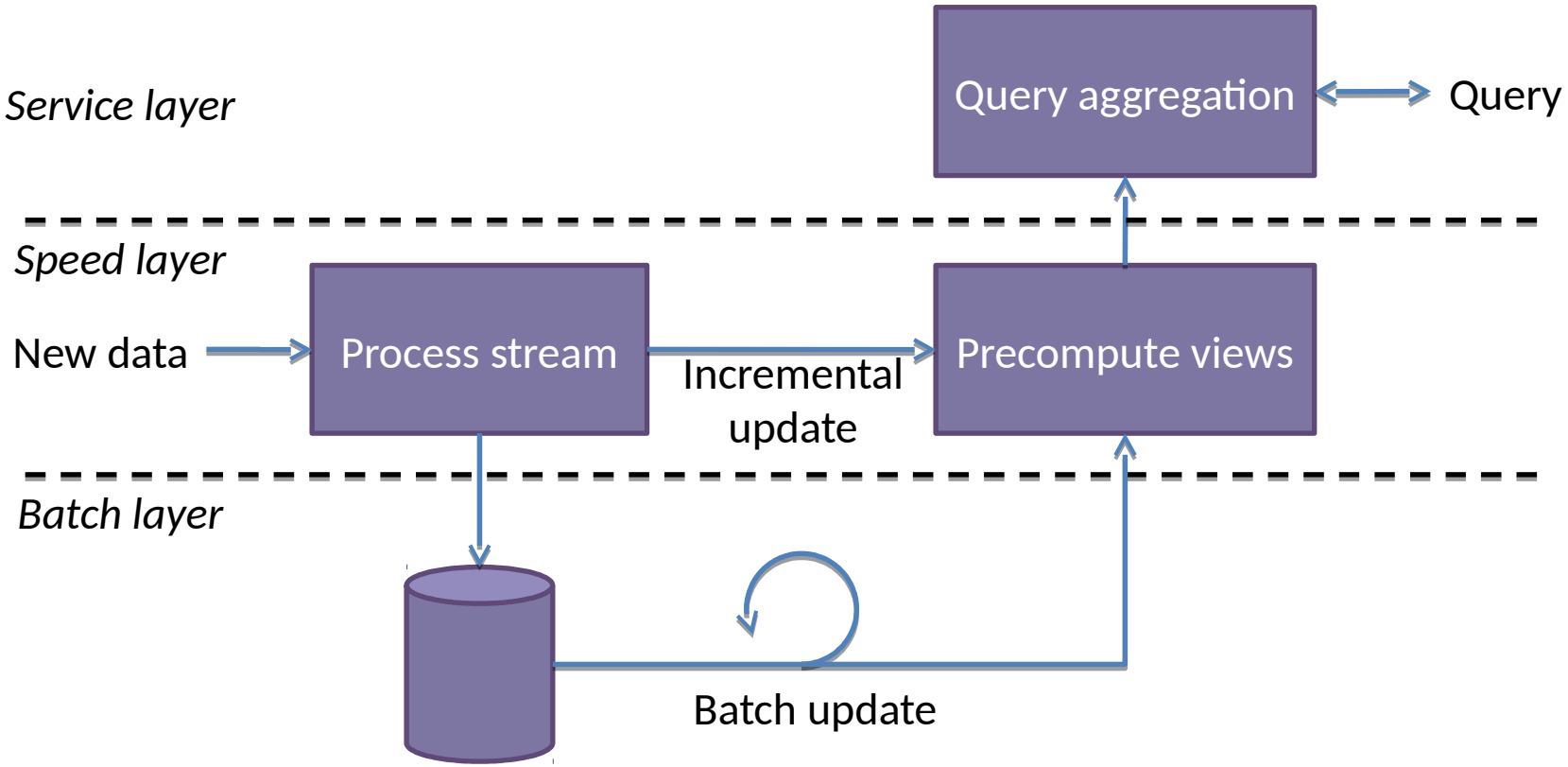
- integrovanie (geo-lokalizovaných) dát zo sociálnych sietí
- extrahovanie entít
- analýza sentimentu
- interaktívna vizualizácia agregovaných dát – geografické mapy

- objem 1TB rok/oblasť záujmu
- do 10 000 aktualizácií

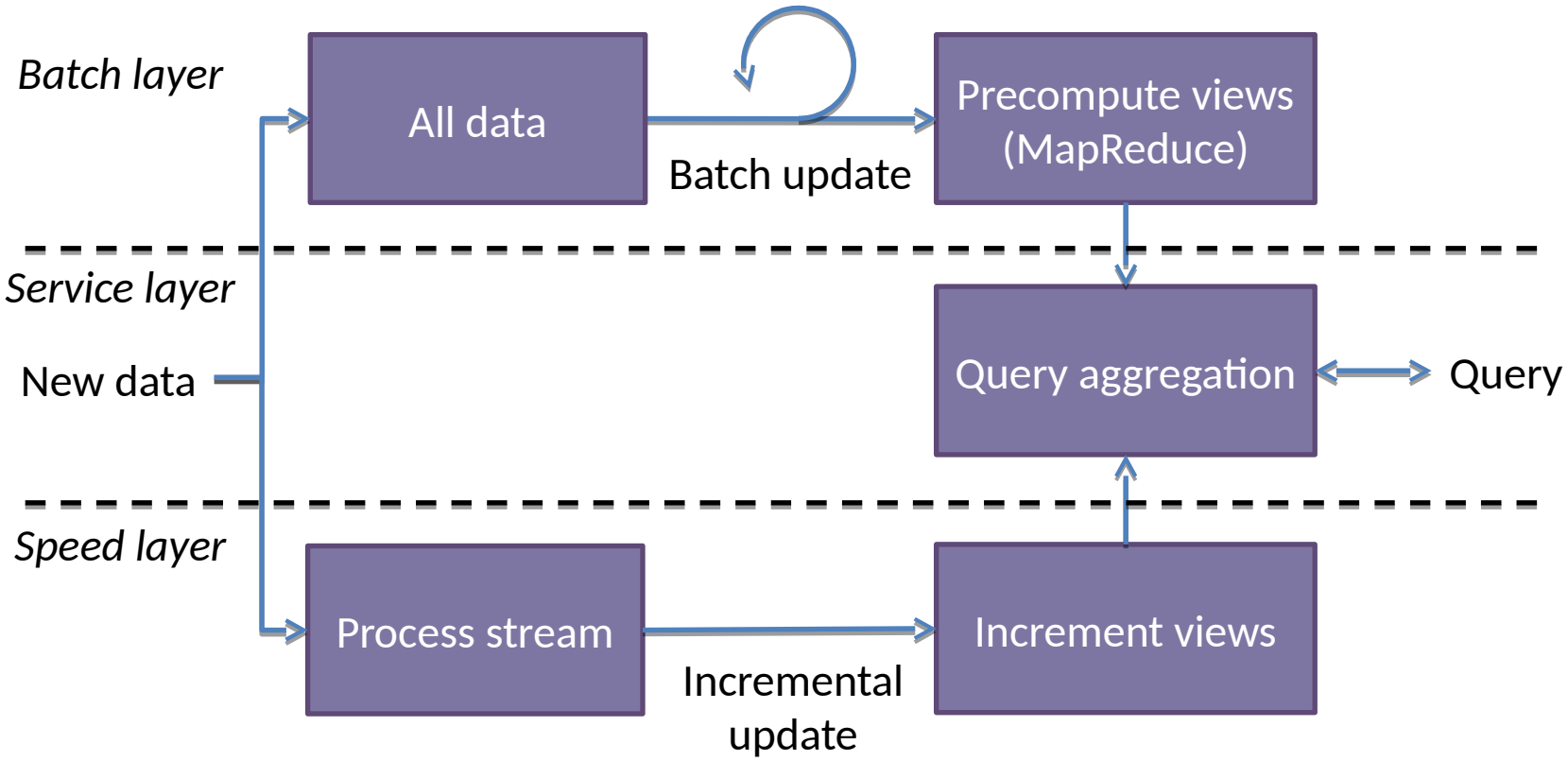
*Urban Sensing (2)



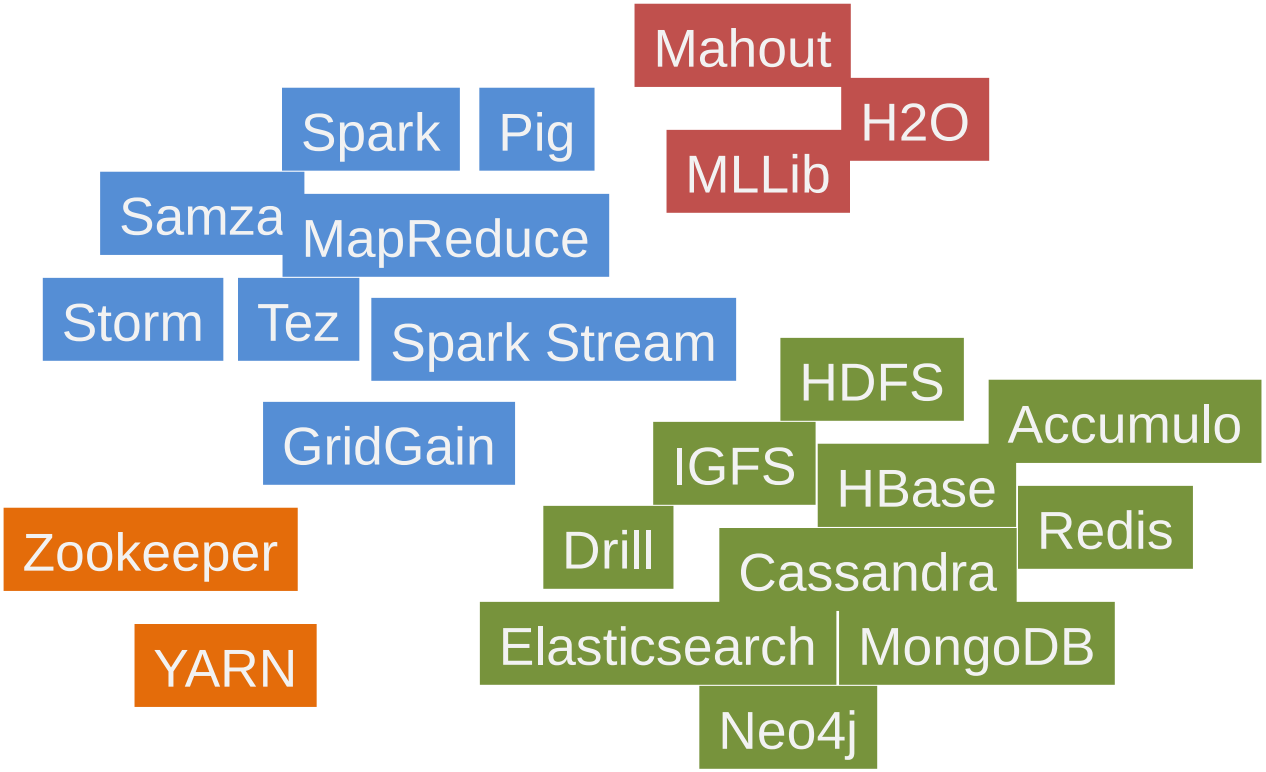
*Urban Sensing architektúra



λ-architektúra



Big Data = Big Explosion (1)



Big Data = Big Explosion (2)

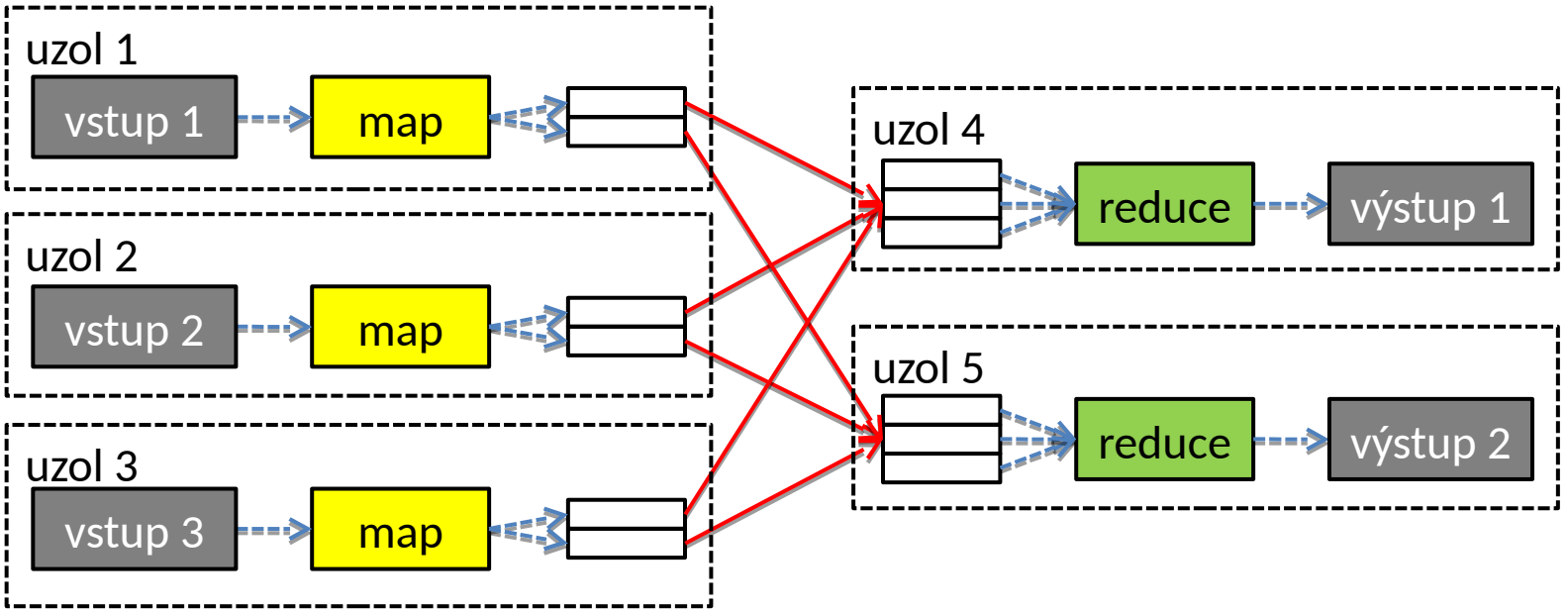
- výkon
- škálovateľnosť
- spoľahlivosť

- programátorské aplikačné rozhranie
- dátová konzistentnosť vs. spoľahlivosť

MapReduce (1)

map(k1, v1) -> list[(k2,v2)]

reduce(k2, list(v2)) -> list[v3]



MapReduce (2)

map(k1, v1) -> list[(k2,v2)]

```
public class TokenCounterMapper extends Mapper {
    public void map(Object key, Text value, Context context) throws
    IOException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
    context.collect(new Text(itr.nextToken()), new IntWritable(1));
    }
     }
}
```

MapReduce (3)

reduce(k2, list(v2)) -> list[v3]

```
public class IntSumReducer extends Reducer {
    public void reduce(Key key, Iterable values, Context context) throws
    IOException {
    int sum = 0;
    for (IntWritable val : values) {
    sum += val.get();
    }
    context.collect(key, new IntWritable(sum));
    }
}
```

Pig Latin (1)

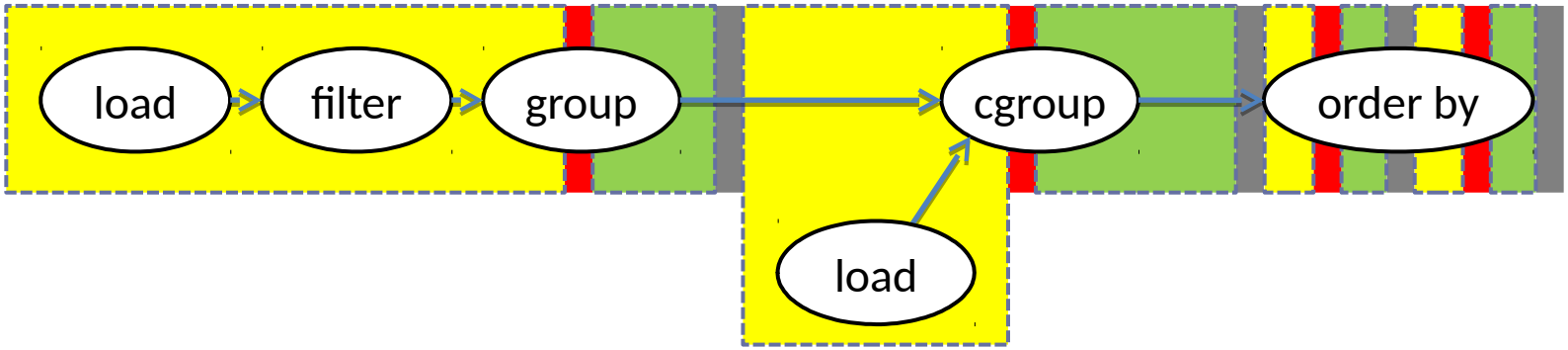
```
input_lines = LOAD '/tmp/all-pages-on-internet' AS (line:chararray);

words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
filtered_words = FILTER words BY word MATCHES '\\w+';

word_groups = GROUP filtered_words BY word;

word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS
    count, GROUP AS word;
ordered_word_count = ORDER word_count BY count DESC;

STORE ordered_word_count INTO '/tmp/number-of-words-on-internet';
```

Pig Latin (2)

- LOAD, STORE – perzistentný stav
- FOREACH – projekcia/transformácia dát
- FILTER, DISTINCT, LIMIT, SAMPLE – výber dát
- GROUP/COGROUP, JOIN, UNION – spájanie dát
- ORDER BY – usporiadanie

- kompilácia do MapReduce DAG
  - optimalizácia – zreťazenie operácií
  - PARTITION BY, PARALLEL

Pig Latin (3)



vnorené príkazy majú viac deklaratívny charakter

Apache Storm (1)

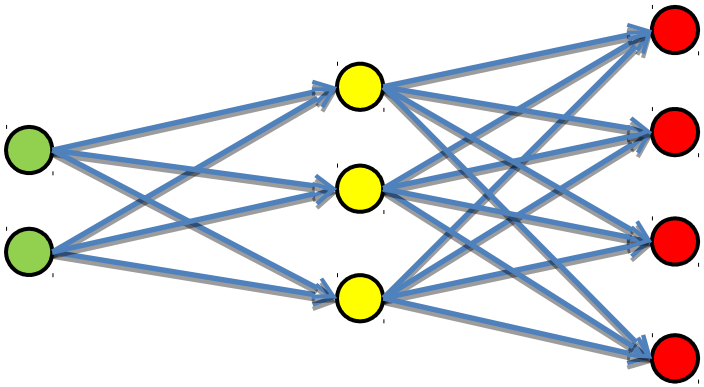
- dátový model (kolekcia máp – *n*-tíc s pomenovanými dátovými poľami, mapy môžu byť vnorené)

- spracovanie sa modeluje priamo ako DAG = topológia + funkcie
  - zdrojové uzly (Spout) – T next()
  - procesné uzly (Bolt) – T = process(T)

- systém zabezpečuje distribuovanie dát
- spoľahlivosť + manažment stavu koordinuje systém, uzly musia podporovať protokol transakcií

Apache Storm (2)

```
TopologyBuilder builder = new TopologyBuilder();

builder.setSpout("spout", new RandomSentenceSpout(), 2);
builder.setBolt("split", new SplitSentence(), 3).shuffleGrouping("spout");
builder.setBolt("count", new WordCount(), 4).fieldsGrouping("split", new
    Fields("word"));
```



Apache Storm - Trident (3)

```
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);
```

dátový prúd + funkcionálne programovanie

Apache Spark (1)

```
val textFile = spark.textFile("hdfs://...")
val counts = textFile.flatMap(line => line.split(" "))
                     .map(word => (word, 1))
                     .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

dátový prúd + funkcionálne programovanie

Apache Spark (2)

```
val points = spark.textFile(...).map(parsePoint).cache()
var w = Vector.random(D)
for (i <- 1 to ITERATIONS) {
    val gradient = points.map(
            p => (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x).
            reduce(_ + _)
    w -= gradient
}
```

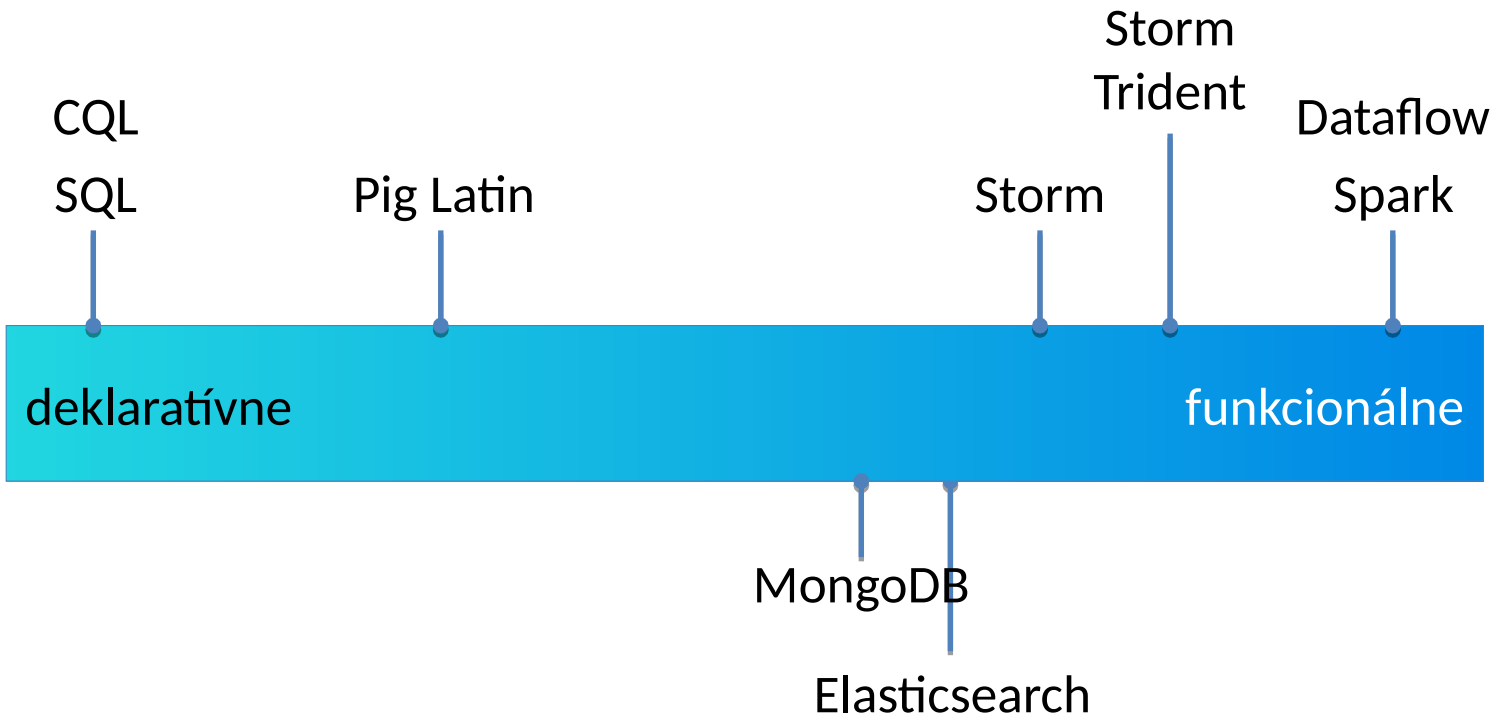
distribuovanie premenných

MongoDB

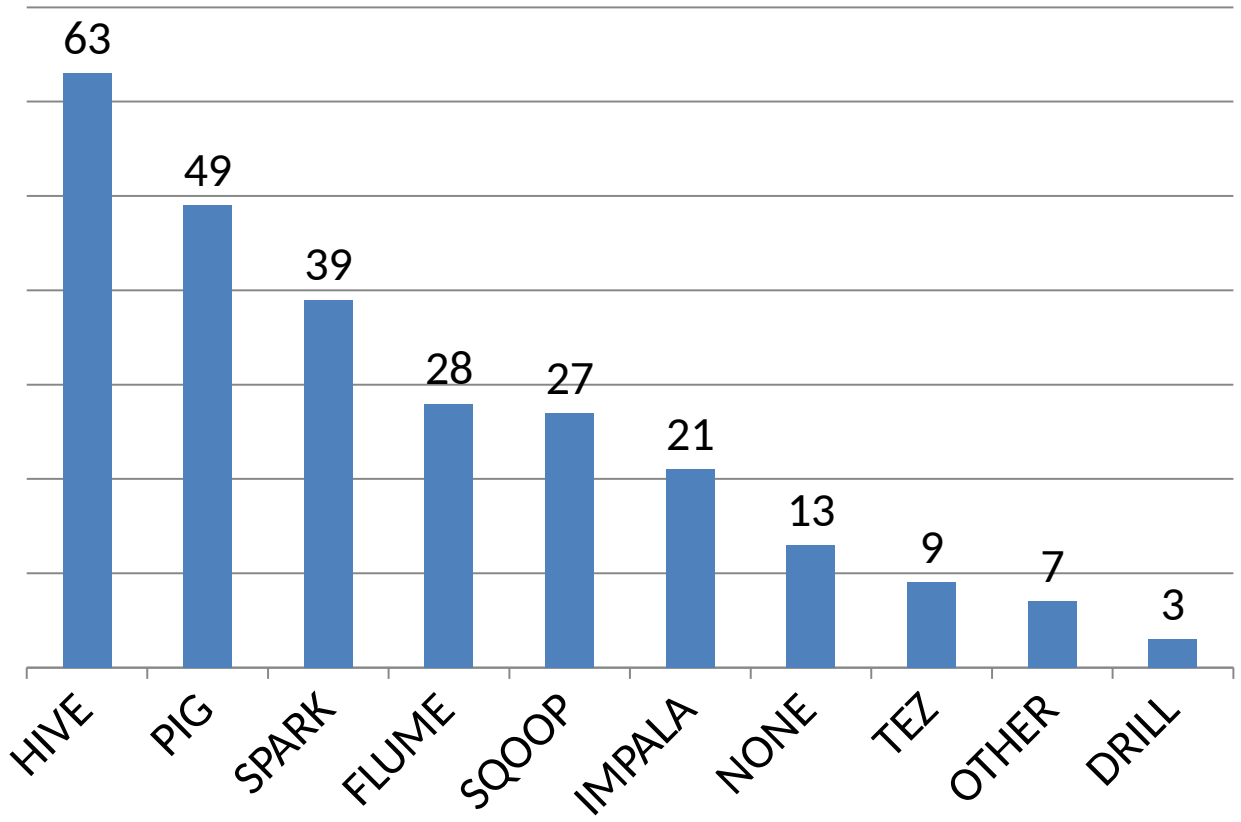
```
db.orders.aggregate([
        { $match:   { status: “A” } },
        { $group:   { _id: “$cust_id”, total: { $sum: “$amount” } } },
        { $sort:    {amount: 1} }
])
```

dátový prúd + „funkcionálne“ programovanie

Aplikačné programové rozhrania



Big Data = Big Explosion



DZone 2015 survey

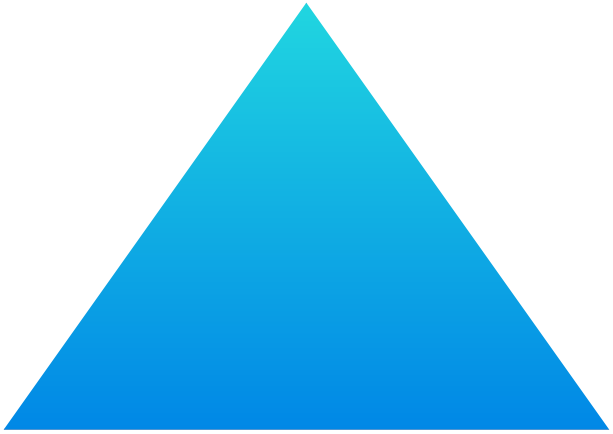
Funkcionálny model pri návrhu

- kompozícia funkcií T = *f*(T, lambda)
- T = {v_1, v_2, …, v_n}, T = v_1, v_2, …
- filter(V) → boolean
- map(V_1) → (K, V_2), reduce(V_1,V_2) → V_3
- subtract, join
- window
- save(state), read(state)

- časová zložitosť + pamäťová zložitosť
- komunikačná zložitosť
- perzistentná zložitosť

Consistency – Availability – Partition

Konzistentnosť – **C**
všetci klienti majú
„správny“ pohľad na dáta



Dostupnosť – **A**
všetci klienti môžu vždy
zapisovať a čítať dáta

Spoľahlivosť – **P**
systém pracuje správne aj
pri výpadkoch uzlov

Konzistentnosť (1)

- konzistentnosť ovplyvňuje replikáciu dát a podporu transakcií

- „silná“ konzistentnosť – po potvrdení zápisu všetky uzly vrátia aktualizovanú hodnotu
  - zápis cez primárny uzol, ktorý atomicky replikuje dáta na záložné uzly
  - čítanie cez všetky uzly
  - škálovateľnosť a robustnosť

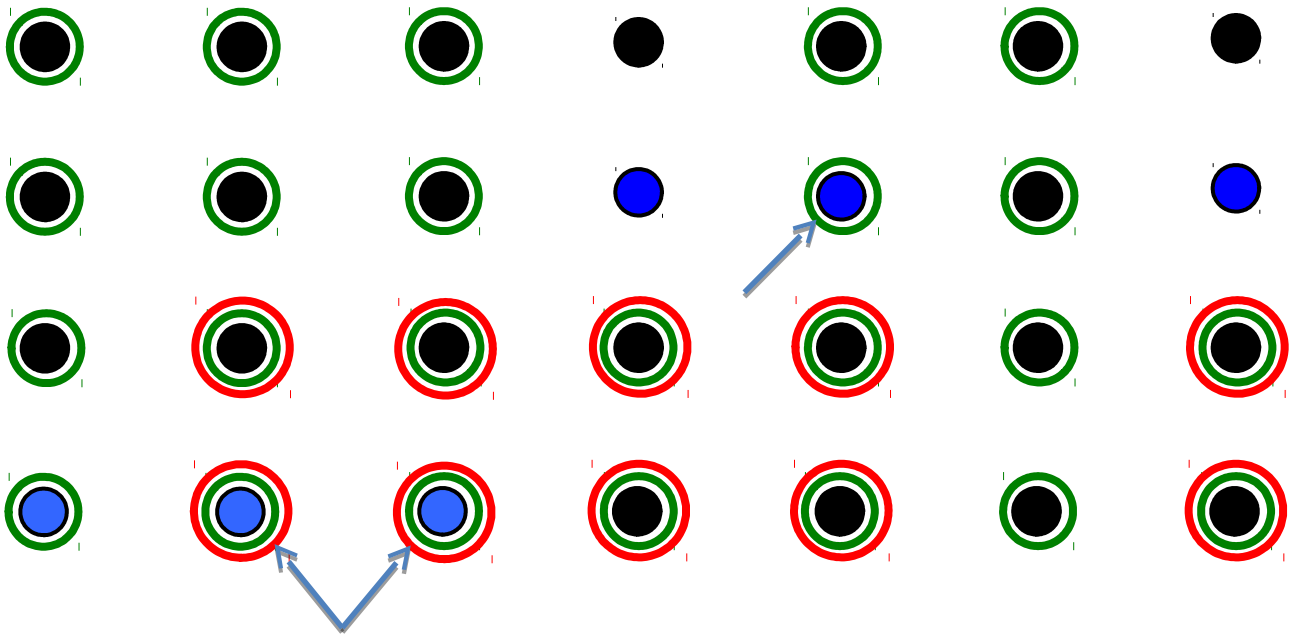
- „slabá“ konzistentnosť – niektoré uzly sú aktualizované asynchrónne, po čase musí systém skonvergovať do konzistentného stavu

Konzistentnosť (2)

- ak máme n kópií, definujeme kvórum pre zápis QW a pre čítanie QR pre ktoré platí:
  - QW > n/2
  - QW + QR > n
- klient môže komunikovať s ľubovoľným uzlom, ktorý zostaví kvórum a inicializuje operáciu
  - iba jeden uzol môže inicializovať zápis
  - aspoň jeden uzol pozná aktuálnu hodnotu

Konzistentnosť (3)

- príklad pre n = 7, QW = 5, QR = 3

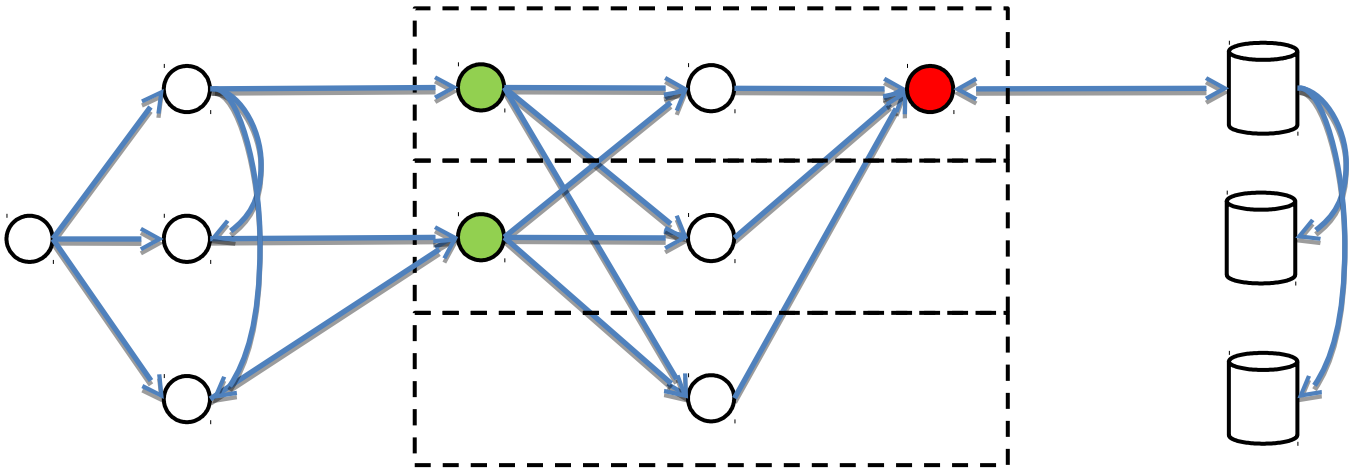


Konzistentnosť (4)

- spracuj „aspoň raz“– spoľahlivosť
- spracuj „práve raz“ – agregovanie dát
- systém musí zachovať usporiadanie dát

- záznamom je priradené jedinečné ID transakcie, ktoré monotónne rastie
- perzistentne je uložená dvojica (ID transakcie, stav)
- ak sa ID opakuje, stav nie je potrebné meniť, inak sa aktualizuje na novú hodnotu

Konzistentnosť (5)



fronta správ

aký model pre replikáciu?

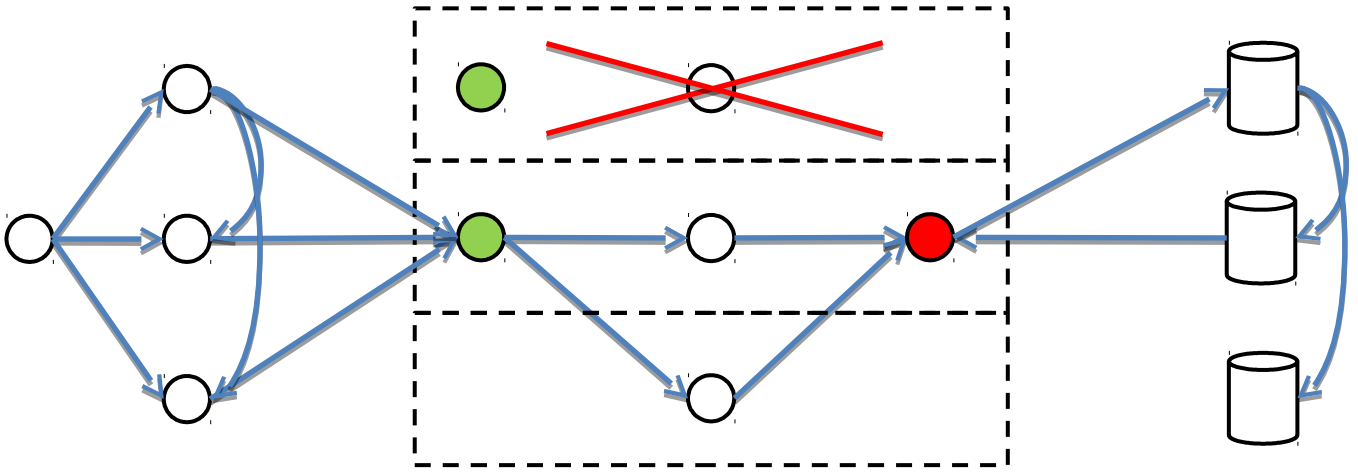
in-memory distribuované výpočty

aký model pre distribuovanie správ?

distribuovaná databáza

aký model pre replikáciu?

Konzistentnosť (6)



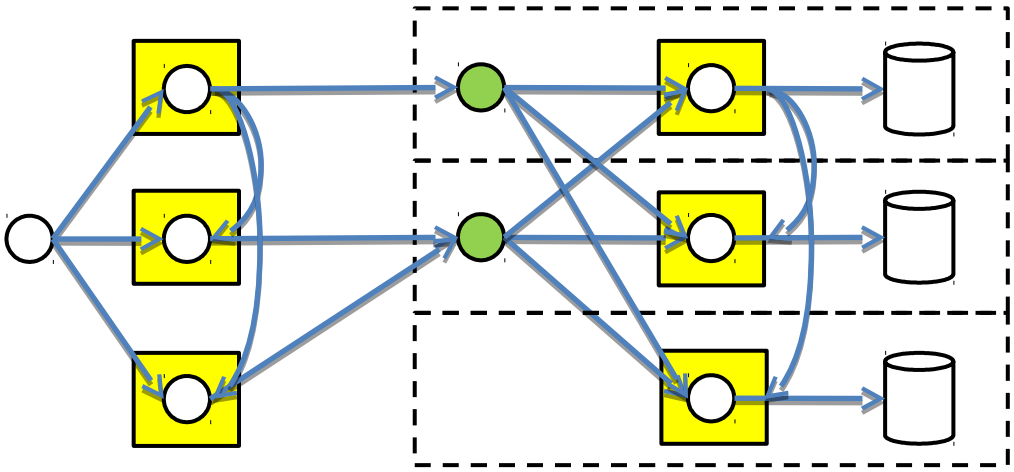
fronta správ

in-memory distribuované výpočty

distribuovaná databáza

interakcie a závislosti

Potrebujeme distribuované databázy?



fronta správ

in-memory distribuované
výpočty + lokálne úložiská

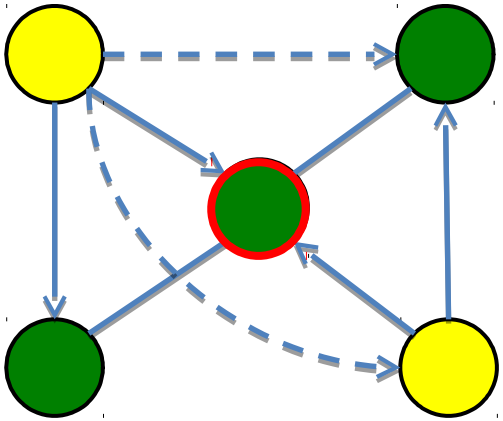
Konzistentnosť (8)

- „silná“ konzistentnosť vyžaduje viac komunikácie medzi uzlami čo vedie k zníženiu dostupnosti – výkonu – dátovej priepustnosti
- pre Big Data aplikácie sa často uprednostňuje výkon

- **„slabá“ konzistentnosť pri návrhu a implementácii nemusí byť problém**
- **ale... problémom môže byť spoľahlivosť pri prevádzke**

Spoľahlivosť

- zlyhanie uzla/uzlov
  - primárny/sekundárny
  - dočasné/trvalé
  - všetky naraz
- obmedzená komunikácia
  - rozdelenie systému



- nie len spoľahlivosť fyzickej siete!

Konzistentnosť + spoľahlivosť (1)

- pri návrhu a implementácii je potrebné voliť kompromis (CAP!) – veľa implementačných detailov
- každý systém „deklaruje“ nejaký model konzistentnosti a spoľahlivosti

1. systém nepracuje tak ako ho navrhli
   - implementačná chyba
2. nie je jasné ktoré prípady zlyhania systém:
   1. vie detegovať
   2. automaticky toleruje
   3. vie tolerovať so zásahom administrátora
   - neúplná dokumentácia

Konzistentnosť + spoľahlivosť (2)

- regresné testovanie
- testovacia sada dobre definovaných prípadov zlyhania
- analýza incidentov z reálnej prevádzky
- vhodné nástroje
  - Jepsen
  - ... „call me maybe“ <váš obľúbený systém> https://aphyr.com/tags/jepsen

"Your recent Amazon purchases, Tweet score and location history makes you 23.5% welcome here."