# Evolúcia Big Data systémov

Marián Dvorský

Máj 2017

# Obsah

Čo sú Big Data?

V čom je problém?

Ako sa rieši?

Čo ešte treba vyriešiť?

# Koľko dát je na svete?

# Dáta

| | | |
|---|---|---|
| bajt | 1 | 8 bitov, 01001001, 256 rôznych hodnôt |

# Dáta

| | | |
|---|---|---|
| bajt | 1 | 8 bitov, 01001001, 256 rôznych hodnôt |
| kilo | 1,000 | text dvojparagrafového emailu |

# Dáta

| | | |
|---|---|---|
| bajt | 1 | 8 bitov, 01001001, 256 rôznych hodnôt |
| kilo | 1,000 | text dvojparagrafového emailu |
| mega | 1,000,000 | jedna megapixelová fotka |

# Dáta

| | | |
|---|---|---|
| bajt | 1 | 8 bitov, 01001001, 256 rôznych hodnôt |
| kilo | 1,000 | text dvojparagrafového emailu |
| mega | 1,000,000 | jedna megapixelová fotka |
| giga | 1,000,000,000 | jedna hodina HD videa |

# Dáta

| | | |
|---|---|---|
| bajt | 1 | 8 bitov, 01001001, 256 rôznych hodnôt |
| kilo | 1,000 | text dvojparagrafového emailu |
| mega | 1,000,000 | jedna megapixelová fotka |
| giga | 1,000,000,000 | jedna hodina HD videa |
| tera | 1,000,000,000,000 | veľkosť celého webu v 1997 |

# Dáta

| | | |
|---|---|---|
| bajt | 1 | 8 bitov, 01001001, 256 rôznych hodnôt |
| kilo | 1,000 | text dvojparagrafového emailu |
| mega | 1,000,000 | jedna megapixelová fotka |
| giga | 1,000,000,000 | jedna hodina HD videa |
| tera | 1,000,000,000,000 | veľkosť celého webu v 1997 |
| peta | 1,000,000,000,000,000 | YouTube upload každých pár dní v 2015 |

# Dáta

| | | |
|---|---|---|
| bajt | 1 | 8 bitov, 01001001, 256 rôznych hodnôt |
| kilo | 1,000 | text dvojparagrafového emailu |
| mega | 1,000,000 | jedna megapixelová fotka |
| giga | 1,000,000,000 | jedna hodina HD videa |
| tera | 1,000,000,000,000 | veľkosť celého webu v 1997 |
| peta | 1,000,000,000,000,000 | YouTube upload každých pár dní v 2015 |
| exa | 1,000,000,000,000,000,000 | Internet traffic za deň v 2015 |

# Dáta

| | | |
|---|---|---|
| bajt | 1 | 8 bitov, 01001001, 256 rôznych hodnôt |
| kilo | 1,000 | text dvojparagrafového emailu |
| mega | 1,000,000 | jedna megapixelová fotka |
| giga | 1,000,000,000 | jedna hodina HD videa |
| tera | 1,000,000,000,000 | veľkosť celého webu v 1997 |
| peta | 1,000,000,000,000,000 | YouTube upload každých pár dní v 2015 |
| exa | 1,000,000,000,000,000,000 | Internet traffic za deň v 2015 |
| zetta | 1,000,000,000,000,000,000,000 | všetky uložené dáta na svete v ~2012 |

# Ako spracovávať toľko dát?


Web | Images | Groups | Directory | News

Google Search | I'm Feeling Lucky

- Advanced Search
- Preferences
- Language Tools

Advertise with Us - Business Solutions - Services & Tools - Jobs, Press, & Help

# Problém: Web

Miliardy webových stránok, ~20TB.

Ako ich nájsť a stiahnuť?

**Ako vygenerovať index?**

Ako vyhľadávať v indexe daný dopyt?

Ako zoradiť výsledky podľa relevantnosti?

# Google riešenie

Uložiť dáta na stovky/tisíce “obyčajných” počítačov.

Spojiť ich sieťou (Ethernet).

Systém na distribuované spracovávanie.

# Big Data

Vyžadujú (omnoho) viac ako 1 počítač.

# Google cca. 2002

Ad-hoc systémy na generovanie indexov, analyzovanie záznamov.

Komplikované, logika zmiešaná so systémovými záležitosťami.

Dookola podobné programy s miernymi obmenami.

# Zlepšovák: MapReduce (2003)

Knižnica pre zjednodušenie písania takýchto systémov.

Dáta: záznamy s dvoma údajmi: kľúč a hodnota (key/value).
Užívateľ napíše funkcie Map a Reduce.

Map: (k1, v1) -> list(k2, v2)
Reduce: (k2, list(v2)) -> list(v3)

Systém automaticky distribuuje Map na vstup a Reduce na výstup z Map.

# MapReduce

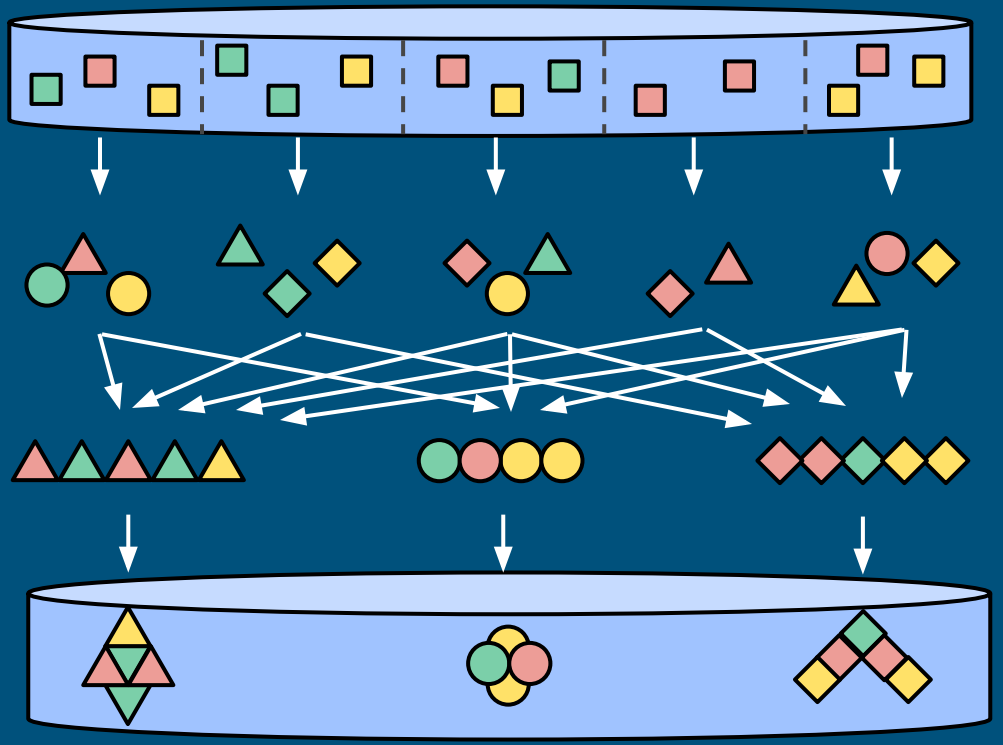
(Vstup)

Map

(Shuffle)

Reduce

(Výstup)



# Príklad: Index

*Vstup*: webové dokumenty
*Výstup*: pre každé slovo, zoznam dokumentov v ktorých sa nachádza

Map(doc_id, dokument) -> [(slovo$_0$, doc_id), (slovo$_1$, doc_id), ...]

Reduce(slovo, [doc_id$_0$, doc_id$_1$, ...]): utriediť a efektívne zakódovať doc_id

# Príklad: Štatistika o vyhľadávaní

*Vstup*: záznamy z vyhľadávania
*Výstup*: pre každý vyhľadávaný reťazec, ako často sa vyhľadával

Map(záznam) -> [dopyt, 1]

Reduce(dopyt, [1, 1, 1, ..., 1]): sčítať hodnoty

# MapReduce architektúra

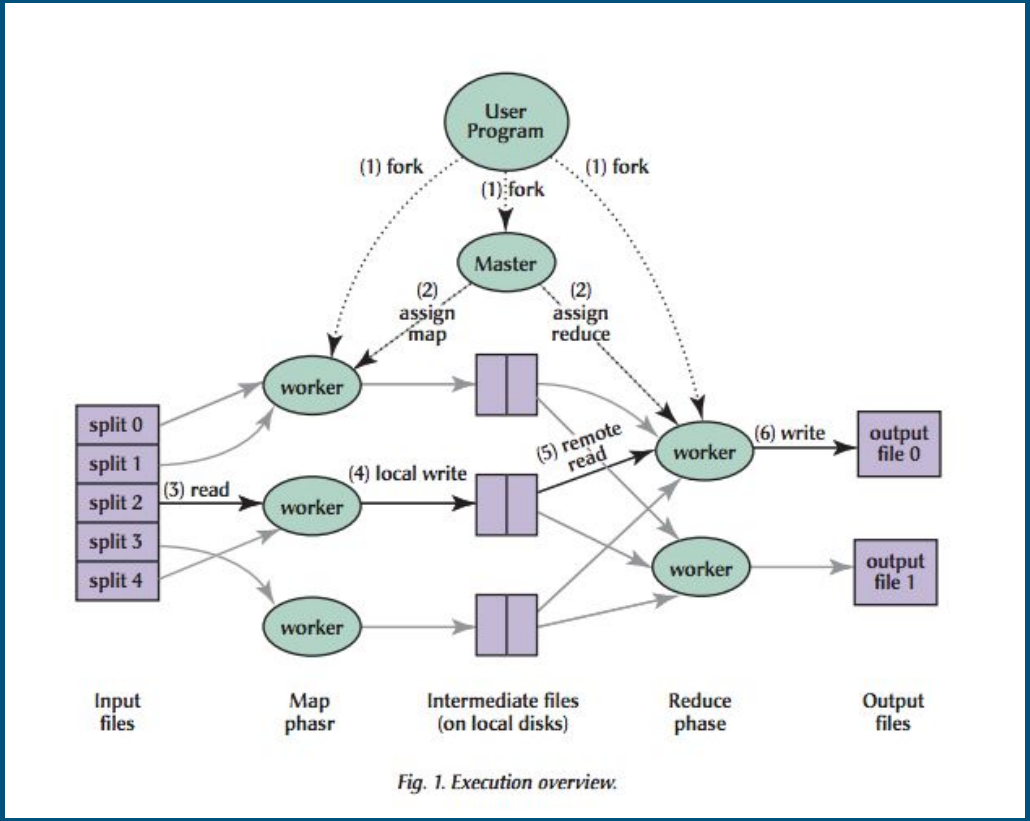



*Fig. 1. Execution overview.*

# MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

*Google, Inc.*

## Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a *map* operation to each logical "record" in our input in order to compute a set of intermediate key/value pairs, and then

# Úspech MapReduce

Google zverejnil článok v 2004 (dnes cez 3000 citácií).

Široké použitie vrámci Google (500 aplikácií prvý rok) vo všetkých oblastiach.

Nápad okopírovaný rýchlo po publikácií a rozšírený do open-source sveta.

# Hadoop (2004)

Na základe Google článku, programátori v Yahoo napísali vlastnú verziu.
Celý ekosystém open-source Big Data riešení.

| | |
|---|---|
| MapReduce | -> Hadoop |
| GFS | -> HDFS |
| Bigtable | -> HBase |

# Kritika databázovej komunity (2008)

*DeWitt, Stonebraker:* MapReduce: Veľký krok späť

Kritizovali primitívny spôsob spracovania dát (vs. SQL).

SELECT key, REDUCE(value) FROM (SELECT MAP(key, value) ...) GROUP BY key

Suboptimálna implementácia (žiadne indexy).
Chýbajúca funkcionalita oproti databázam.

Čiastočne oprávnená kritika, čiastočne nepochopenie problému.

# Problémy masívnych rozmerov

Problémy pri spracúvaní petabajtov dát na 1000 počítačoch sú jedinečné.

Všetkého je veľmi veľa, väčšina algoritmov O(N), prípadne O(N log N).

- Randomizované algoritmy.

Udalosti s nízkou pravdepodobnosťou sú na dennom poriadku.

- Všetko sa vždy kazí.

# Shuffle

Implementácia presunu dát medzi Map a Reduce (GroupByKey).

Ekvivalentné distribuovanému triedeniu.

Doteraz problém, ktorý sa aktívne rieši.

Experiment (2012): 50PB sort, 23 hodín, najväčšie triedenie vôbec.

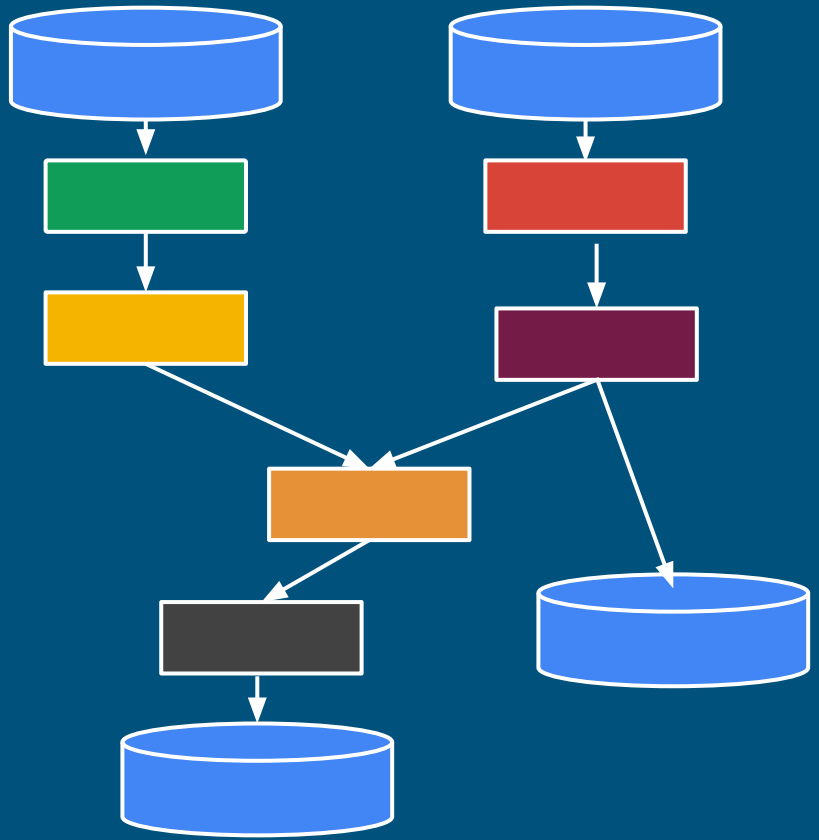
# Flume (2007)

Veľké množstvo aplikácií kombinuje MapReduce výpočty.

Flume: Rozhranie vyššej úrovne.
Optimizátor prekladá Flume program na MapReduce.

Od 2012 sa noví zamestnanci učia už len Flume.

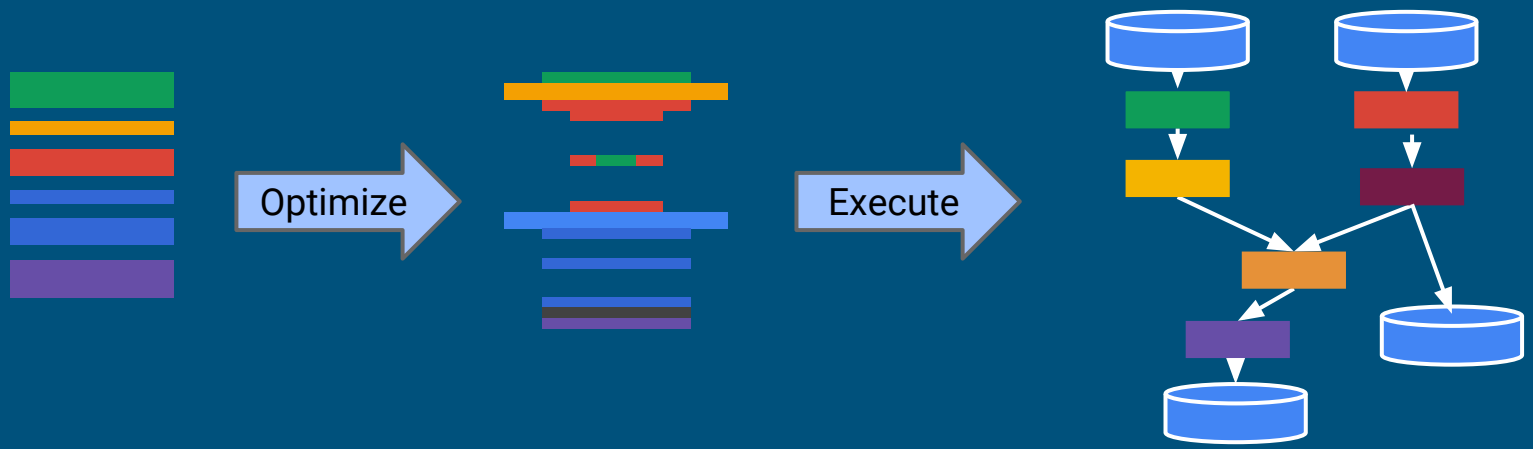
# Flume Architektúra



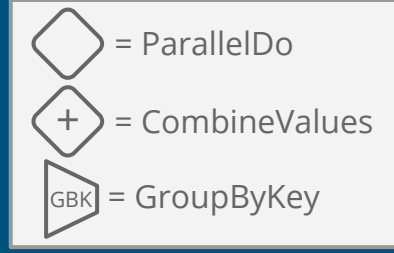
- Graf dátových transformácií.
- Po hranách tečú PCollections, kolekcie typovaných dát.
- Optimizovaný a vykonávaný ako celok.

# Flume Architektúra

```
PCollection<KV<String, Long>> sums =
  IO.read(...)
  .parallelDo(new KeyByUser())
  .count();
```



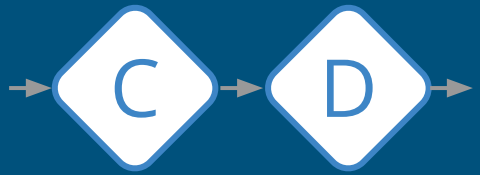
# Optimizácie

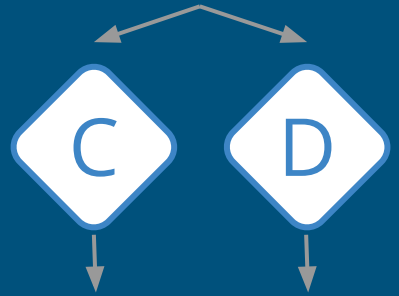



consumer-producer

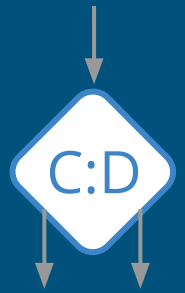
sibling

predtým

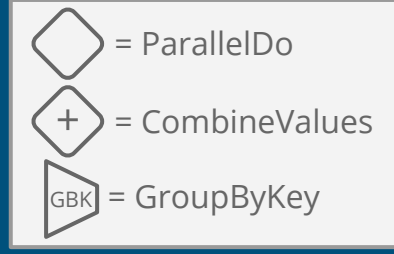


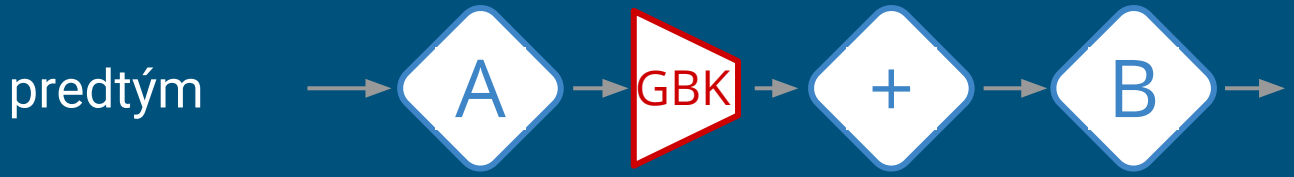



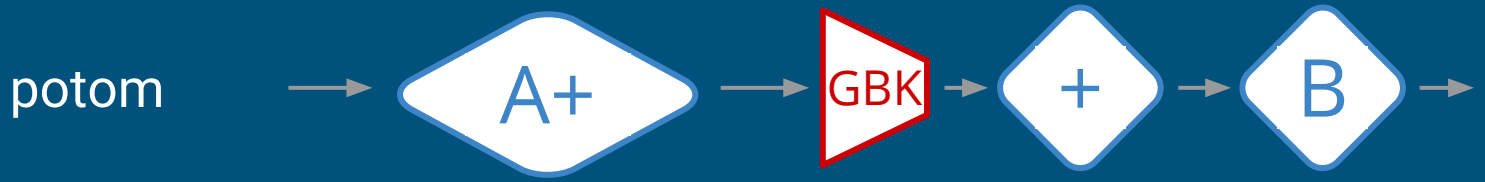

potom

# Combiner Lifting



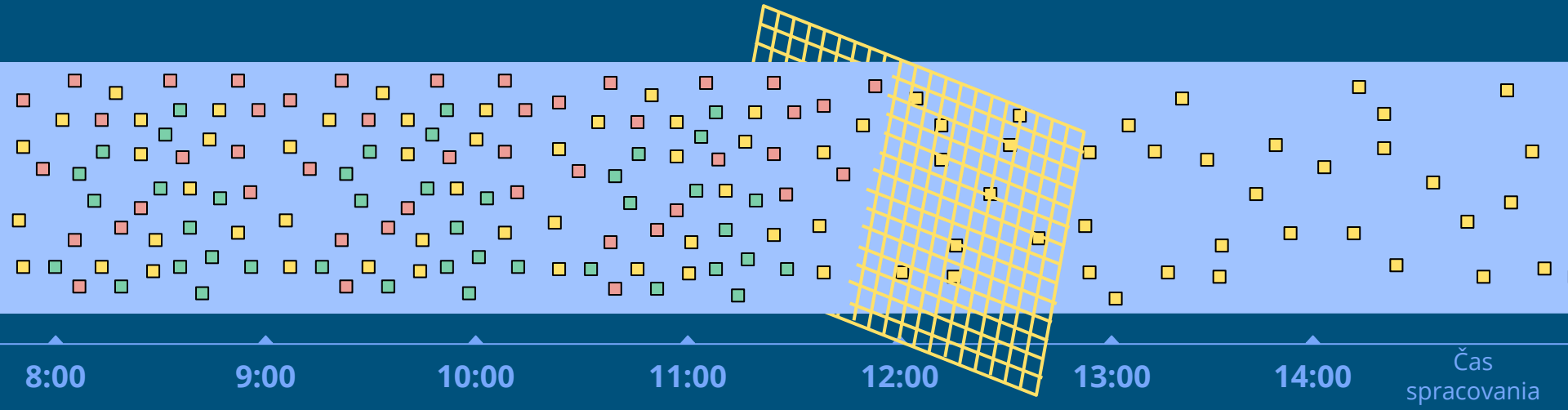
# Spark a Flink (2009)

Nezávislé na sebe vyvýjané open-source projekty s API podobným Flume.

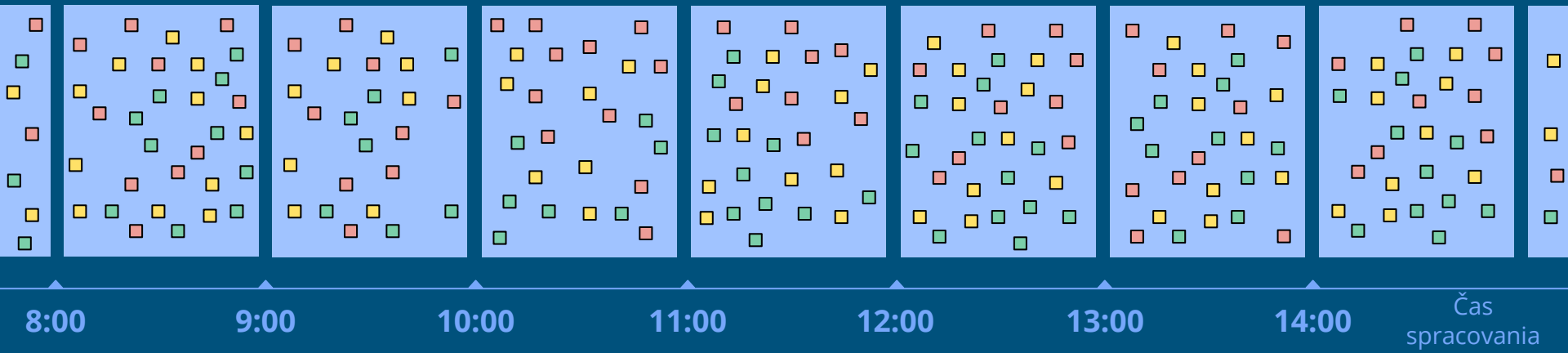
Apache Spark z UC Berkeley.

Apache Flink z TU Berlin.

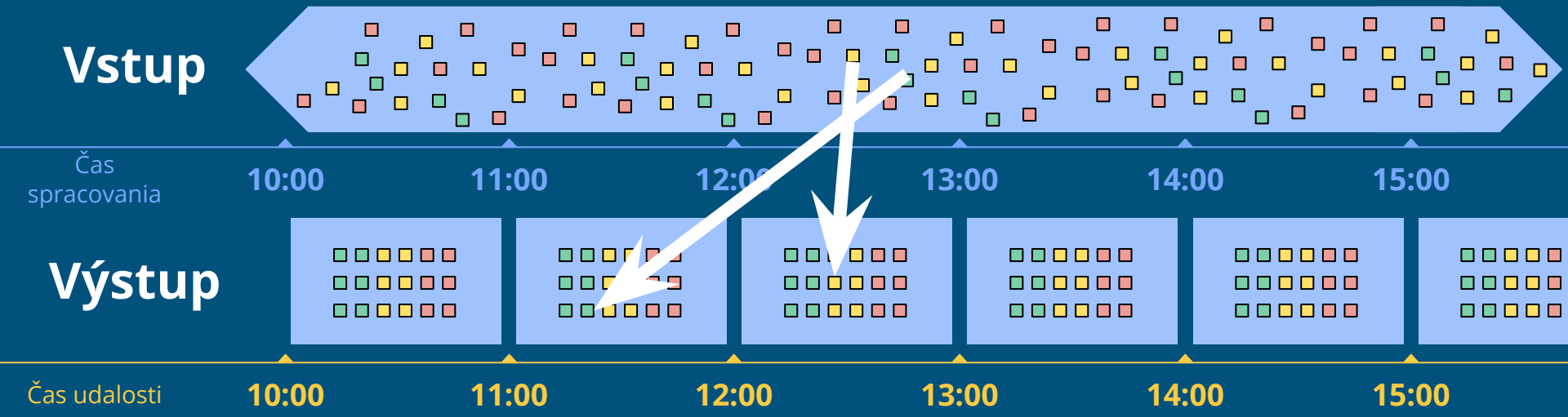
# Neohraničené prúdy dát (streaming)



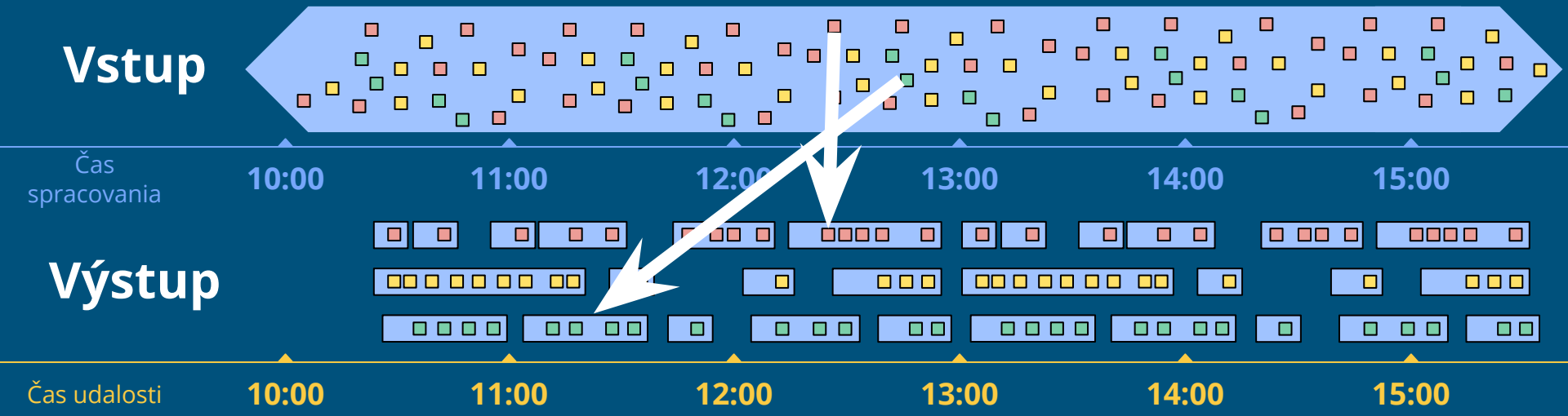
# Agregácie v čase spracovania



# Agregácie v čase udalostí



# Agregácie vrámci aktivít (sessions)



# Millwheel (2007)

Samostatný systém na spracovanie neohraničených prúdov dát.

Nízka latencia výsledkov oproti MapReduce.
Dôraz na správnosť výsledkov.

Založený na Bigtable.

# Dataflow (2013)

Výsledok evolúcie MapReduce, Flume, Millwheel.

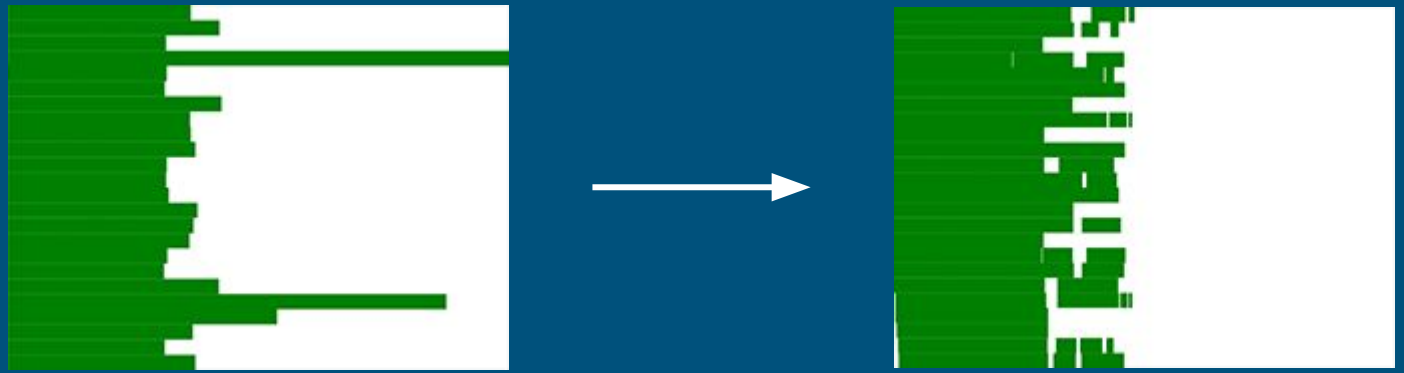
Rovnaké rozhranie pre ohraničené a neohraničené dáta, podobné Flume.

```
PCollection<KV<String, Long>> sums = pipeline
  .apply(TextIO.Read.from(...))
  .apply(ParDo.of(new ParseEventFn()))
  .apply(Window.into(FixedWindows.of(Minutes(2))))
  .apply(Sums.integersPerKey())
```

Spravovaná služba na Google Cloud.
Silný dôraz na auto-konfiguráciu.

# Liquid sharding

Rozdelenie na úlohy nie je vopred fixované, môže sa meniť.



# Apache Beam (2016)

Nový Apache projekt.

Dataflow rozhranie nezávislé od implementácie.

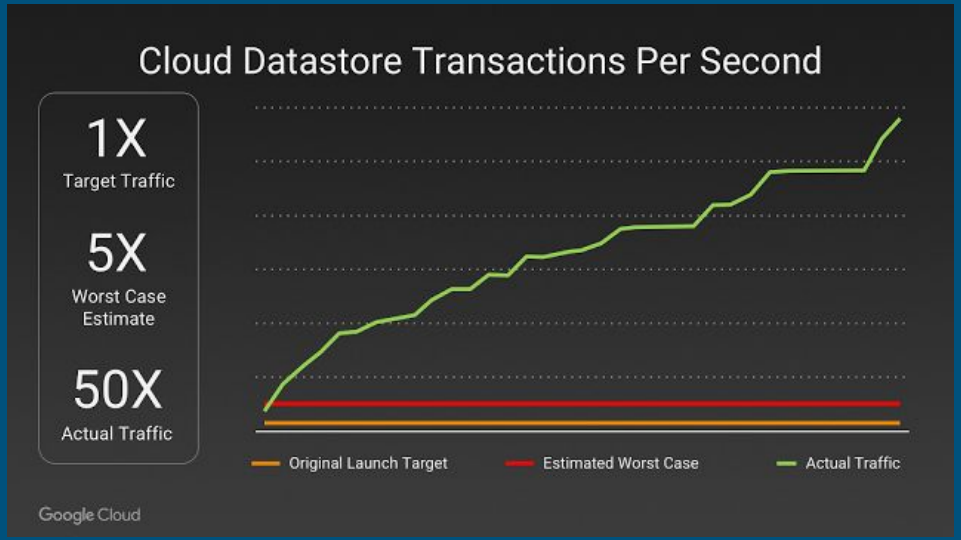
Implementácie na Spark, Flink, Apex, Dataflow.

# Cloud computing

Amazon AWS, Google Cloud Platform, Microsoft Azure a iné.

Umožnuje hocikomu prístup k veľkej výpočtovej sile.

Príklad: Pokémon GO.



# Čo sa ešte rieši

Big Data systémy stále komplikované na použitie.

- Automatizácia konfigurácie.
- Nástroje.

Konsolidácia rozhraní (SQL).

Konvergencia Big Data systémov a databáz.

# Zhrnutie

- BigData problémy vznikli vďaka Internetu.
- Mnoho počítačov a špecializované systémy.
- Jedinečné problémy škálovateľnosti a odolnosti voči chybám.
- MapReduce spustil Big Data revolúciu, ale už sa skoro nepoužíva (Flume/Dataflow, Apache Spark/Flink).
- Apache Beam: rozhranie zjednocujúce batch/streaming.
- Demokratizácia výpočtovej sily vďaka Cloud computing.
- Veľa nevyriešených problémov.