Jak nacrawlovat 1,5 miliardy dokumentů pro fulltext

Ing. Jan Lukavský
vedoucí týmu vývoje
jan.lukavsky@firma.seznam.cz

SEZNAM.CZ

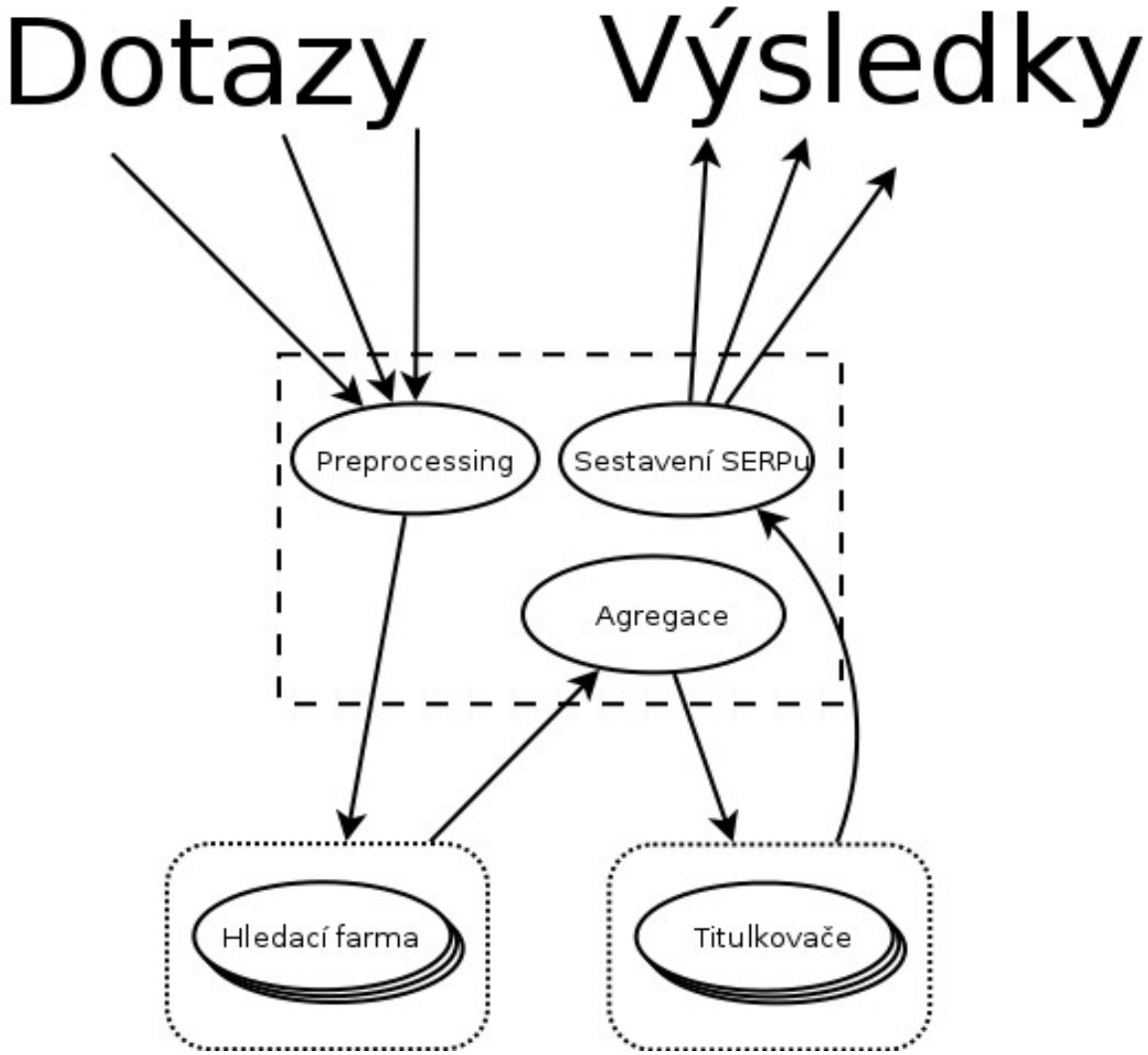
O čem to bude?

- Co je fulltextové hledání?
- Jaká je role crawleru pro fulltext?
  - jaké problémy crawler řeší?
- Kam uložit data?
  - SQL?
  - co je NoSQL?
- Jak data efektivně zpracovávat?
  - co znamená *efektivně*?
  - je Java skutečně *neefektivní* pro zpracování dat?

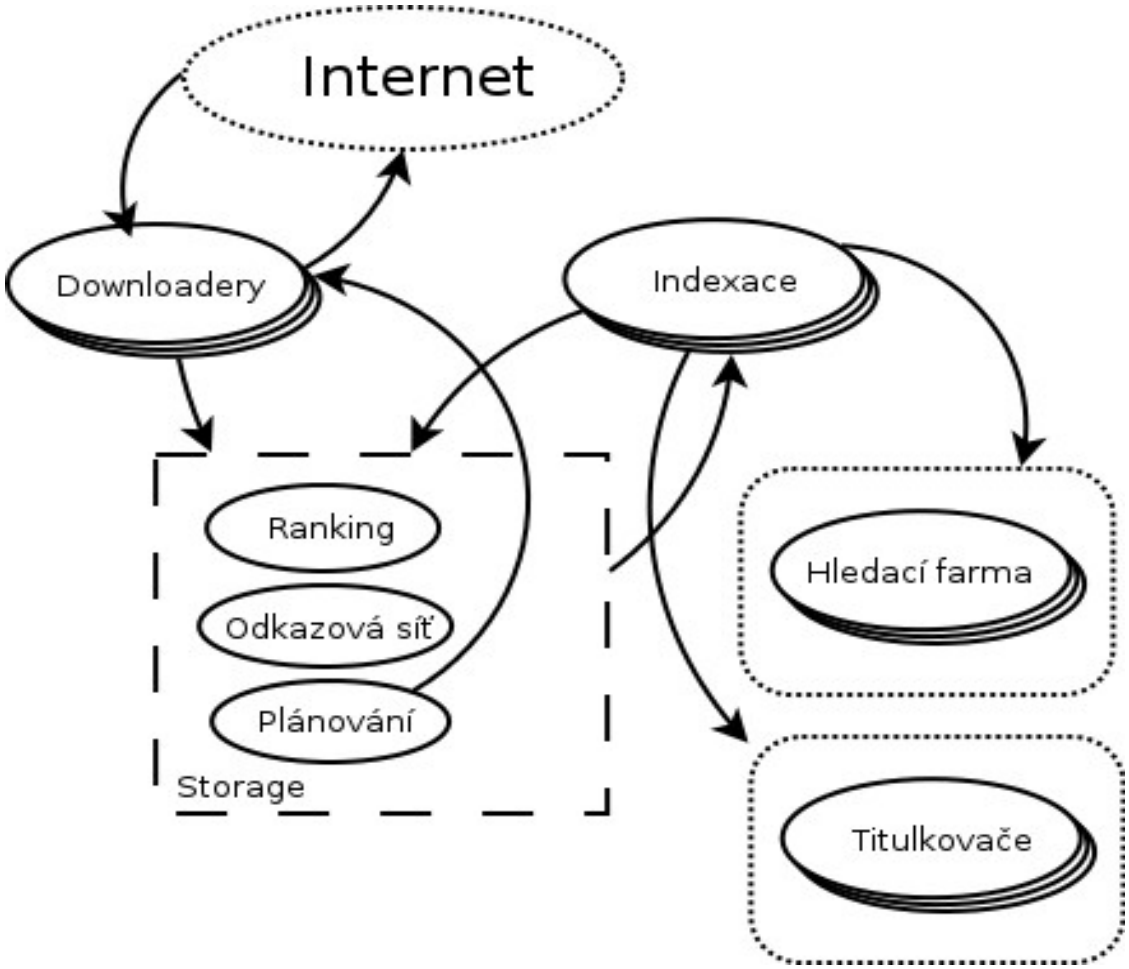
Co je fulltextové hledání?

- vyhledávání v nestrukturovaných datech
- jazyk "dotazů" je jiný než jazyk "výsledků"
  - uživatelé hledají něco jiného, než co zadali
  - zkratky, synonyma, diversifikace, ohýbání slov
- SEO spam
  - snaha uměle ovlivnit přiřozené výsledky vyhledávání

Frontend vyhledávače



Backend vyhledávače



Role crawleru

- "čmuchat" po webu a hledat zajímavý obsah
  - jak ho poznáme?
  - proč prostě "neočmuchat" všechno?
- počítat "statický" ranking
  - PageRank, informační obsah, …
- klasifikovat dokumenty
  - jazyk, porno, spam, tematizace, …
- predikovat signály z odkazové sítě a dalších signálů

Problémy s "čmucháním"

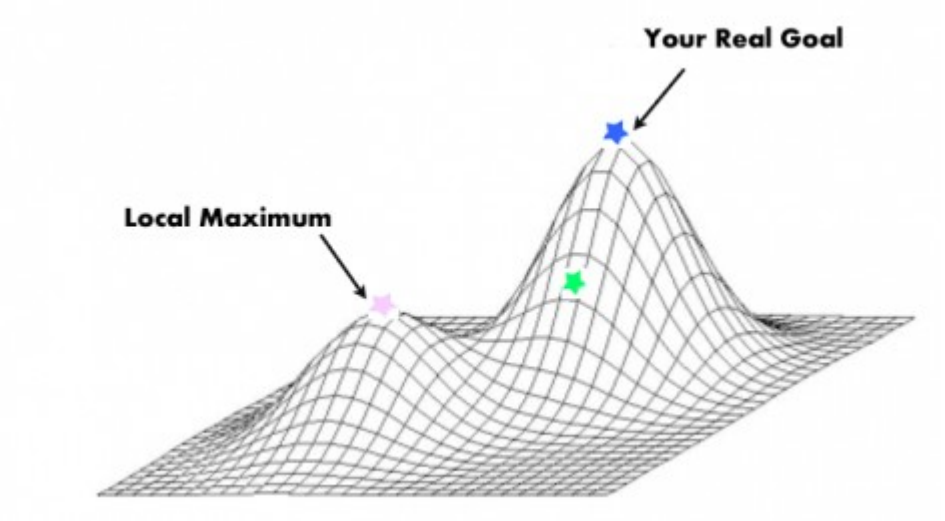
- jedná se o instanci diskrétního optimalizačního problému
  - stavový prostor se skládá z URL
  - sousední stavy reprezentují URL spojené hyperlinkem
- optimalizaci je potřeba provádět při splnění okrajových podmínek
  - kapacita crawleru (počet navštívených stránek)
  - rychlost odezvy webserverů (politeness policy)

Crawler jako optimalizační problém

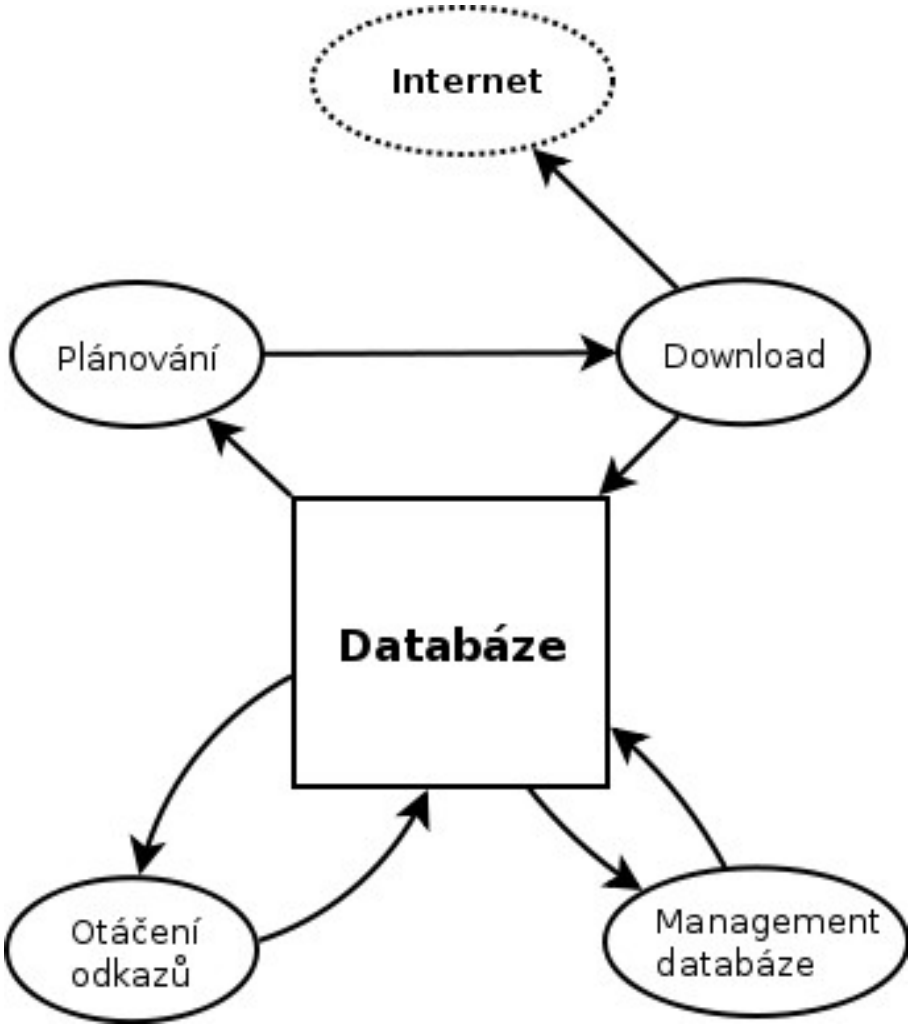
- je potřeba stanovit optimalizační kritérium
  - to je trochu problém ….
  - aktuálně používáme ***sum(score(doc))***
  - *selection policy*
- optimalizace probíhá iterativně
  - výběr dokumentů (podle *selection policy*)
  - stažení
  - nalezení nové *crawler frontier*
  - opakování

Crawler jako optimalizační problém

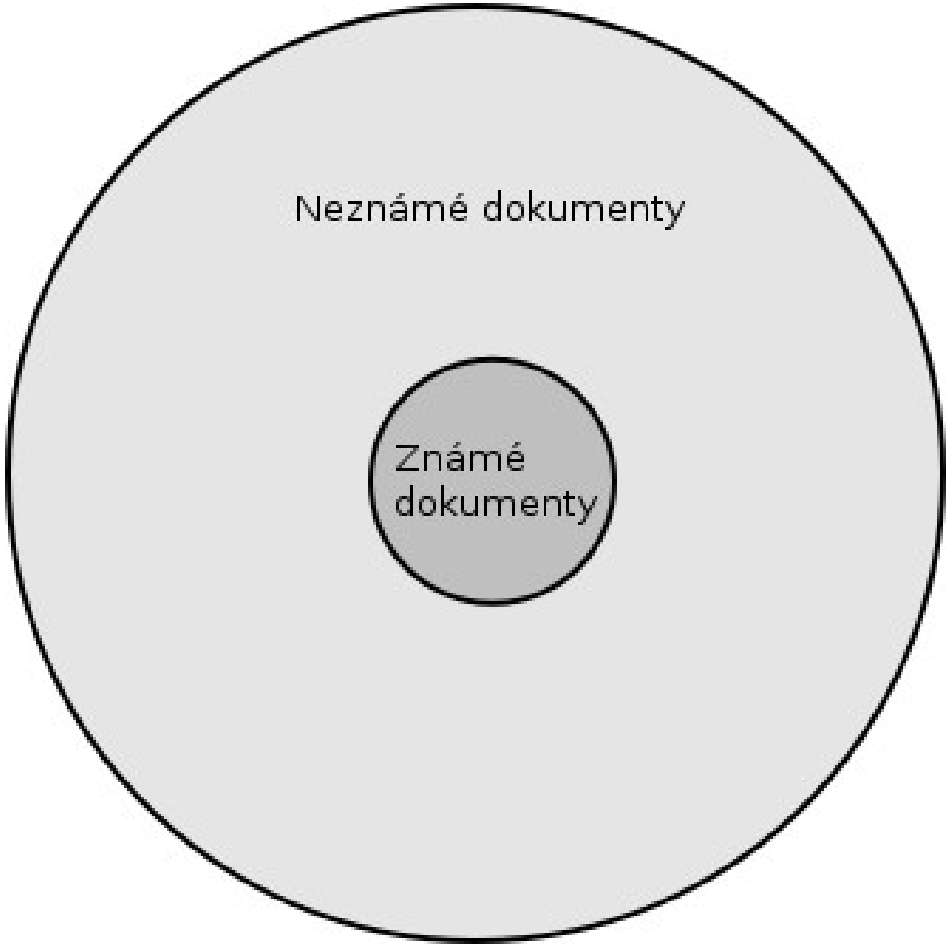
- co lokální extrémy?
  - problém *focusovaných* crawlerů
  - je potřeba vhodně zařadit část kapacitu crawleru na nefocusované procházení webu



Architektura crawleru



Crawler frontier



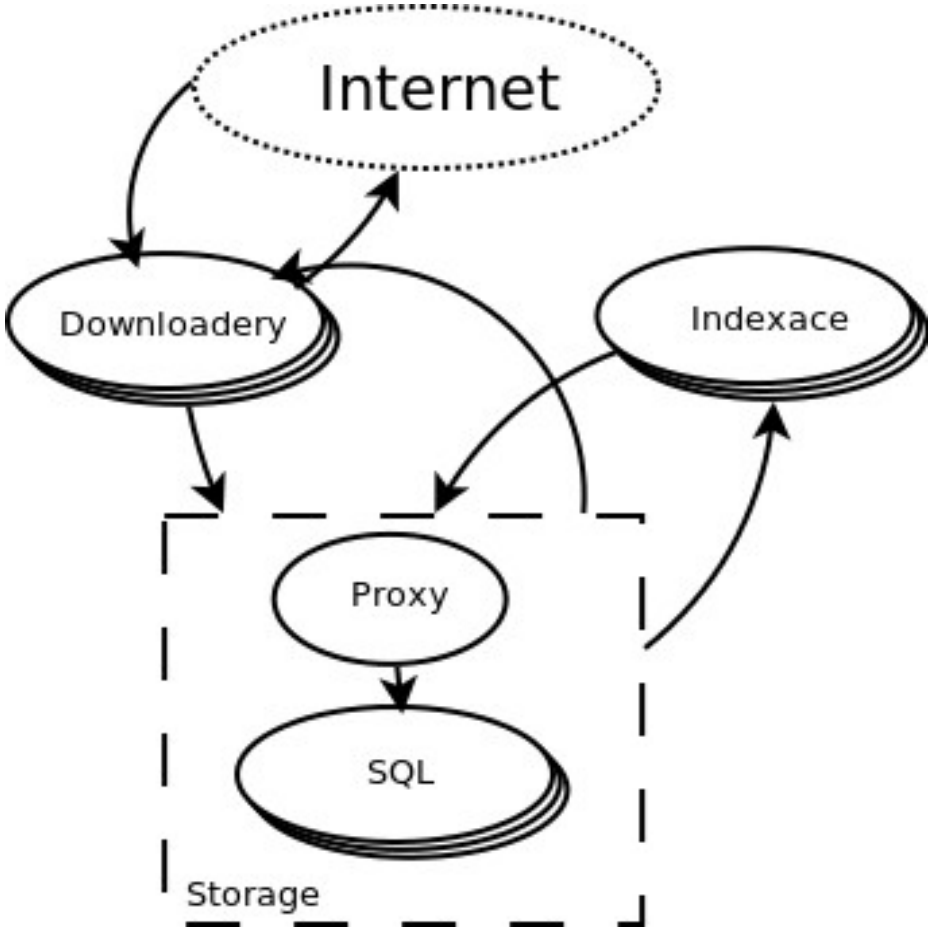
Kam uložit data?

- 1.5G dokumentů
  - ~ 40 TiB HTML dat a extrahovaných textů
  - ~ 15 TiB odkazová síť
  - ~ 25 TiB metadat
  - ~ 80 TiB databáze
- nutno designovat **škálovatelný** systém
  - vertikální škálování (scale-up)
  - horizontální škálování (scale-out)

Vertikální škálování

- celý systém běží na jednom fyzickém HW
- zvětšování kapacity nákupem větších (a dražších) disků, nakupování více pamětí
- budování superpočítačů
- => pro rozumně uvažující komerční firmy příliš drahé
- co s tím?

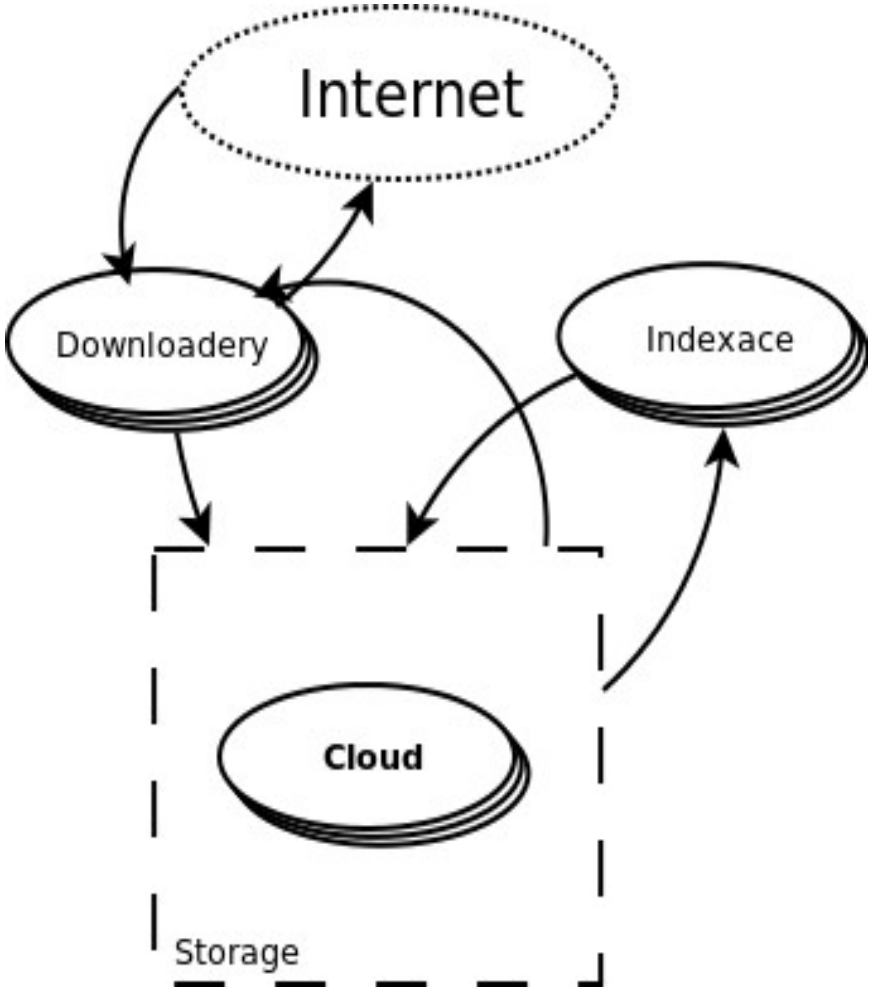
Sharding



Sharding

- relativně škáluje
  - záleží na volbě partitionování shardů
- globální analýzy jsou velmi komplikované
  - neustále dokola se řeší komunikace mezi shardy
  - neustálé problémy s error handlingem
  - problém je **síťový partitioning**
- **neexistuje systém, který toto řeší?**
  - naštěstí existuje

Horizontální škálování



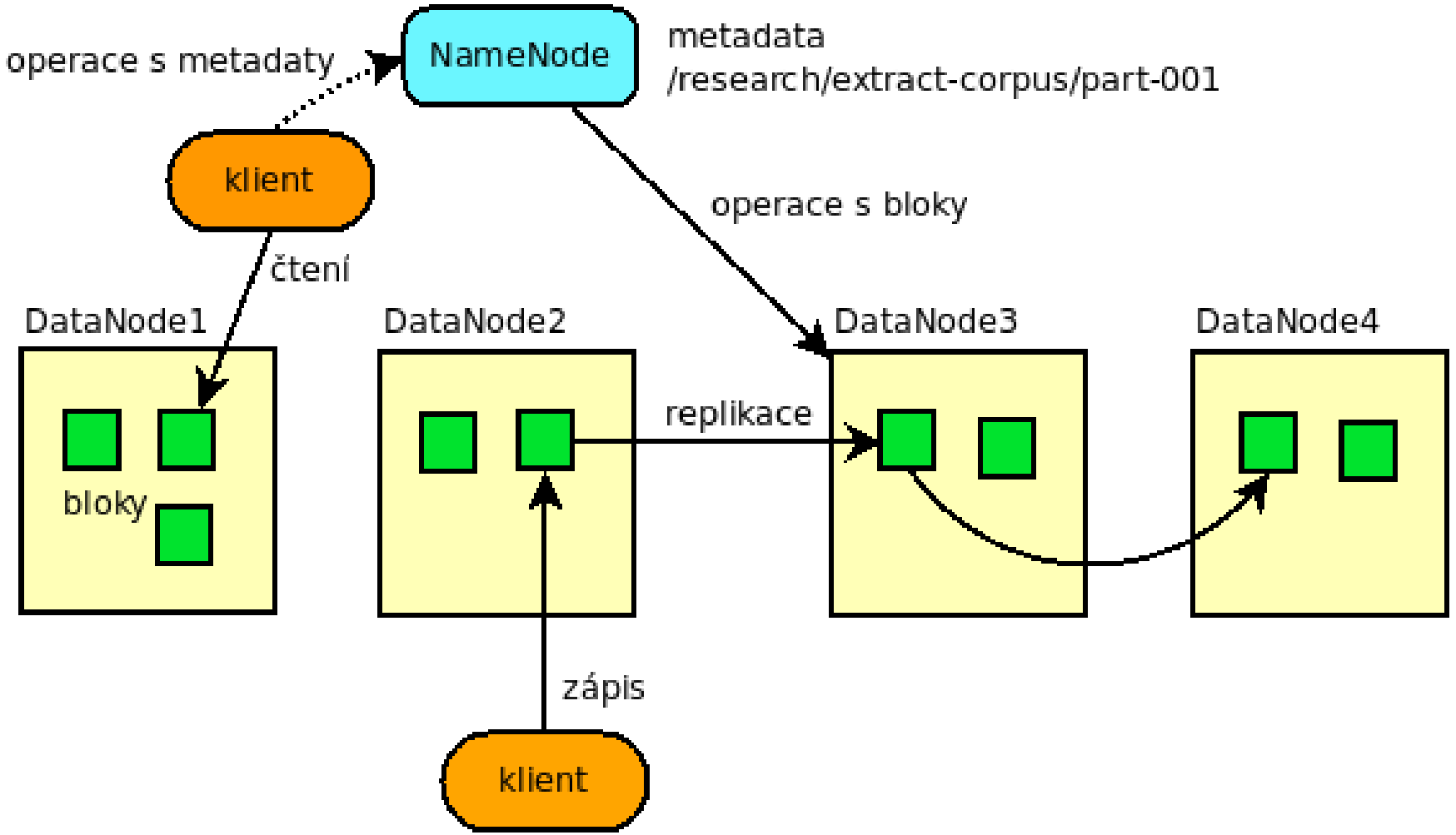
Horizontální škálování

- systém škáluje přidáním výpočetního uzlu
  - "commodity hardware"
  - nezaměňovat s hardwarem na odpis, jedná se o výkonné servery
- větší nároky na síťovou komunikaci
  - výpočetní nody si musí vyměňovat data
- větší nároky na toleranci k výpadku uzlu
  - čím více výpočetních nodů, tím větší šance, že stroj odejde v průběhu výpočtu

Hadoop - HDFS

- reimplementace GoogleFS
- distribuovaný, replikovaný, fault tolerant systém
- standardní filesystém rozdělený na **NameNode** (alokační tabulka) a **DataNode** (slave držící data)
- master (NameNode) zajišťuje kooperaci DataNodů
- řídí replikace a replica placement policy

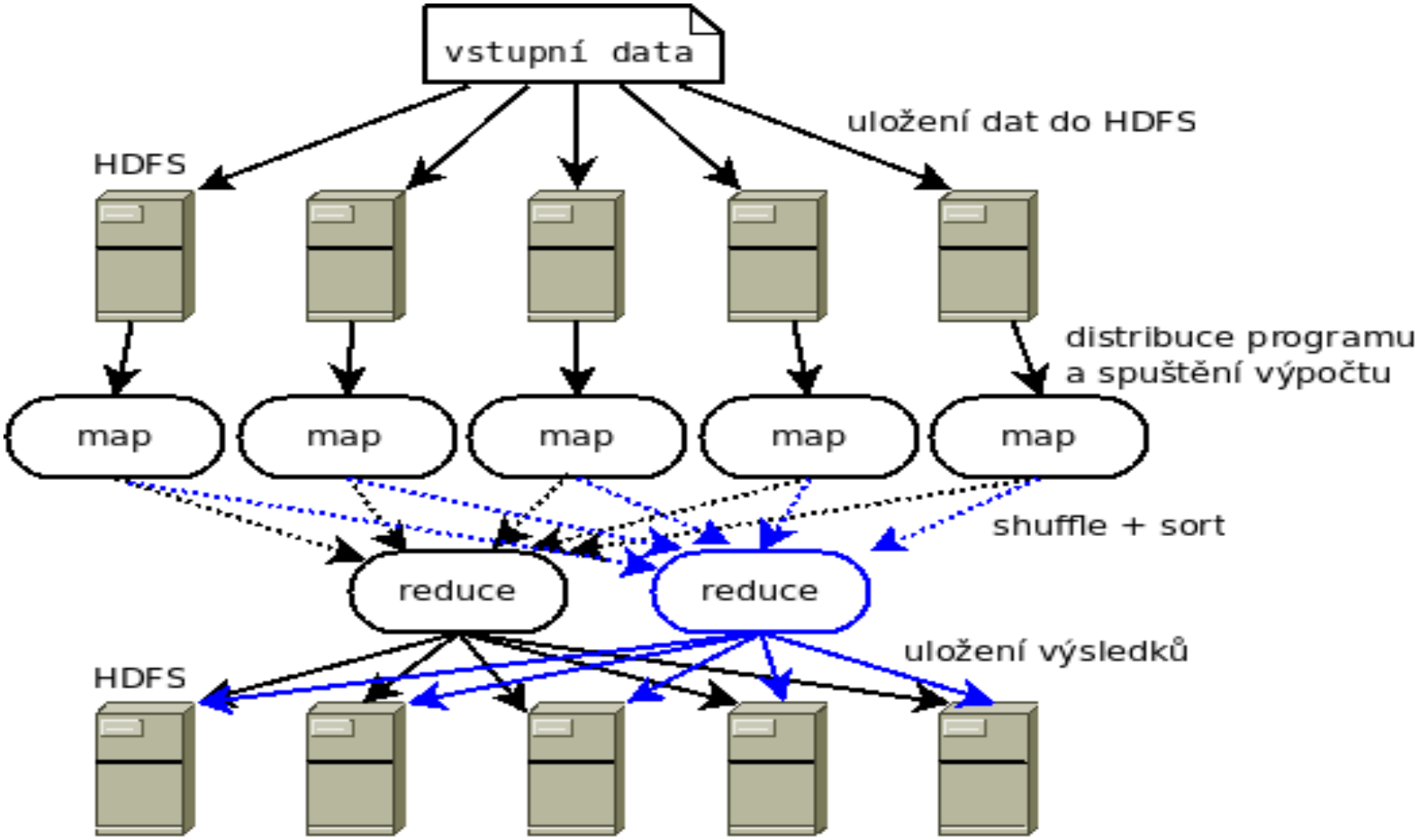
Hadoop - HDFS


operace s metadaty
NameNode
metadata
/research/extract-corpus/part-001
klient
operace s bloky
čtení
DataNode1
DataNode2
DataNode3
DataNode4
replikace
bloky
zápis
klient

Hadoop - Map / Reduce

- distribuované výpočty nad HDFS
  - framework zajišťuje korektní dokončení výpočtu i v prostředí failujících výpočetních uzlů
- jednoduché pro programátora
  - dvousečná zbraň, komplikované algoritmy se špatně designují a koordinují
  - většina algoritmů nejde napsat na jeden Map/Reduce job
- existují nadstavby a obaly

Hadoop - Map / Reduce



Map / Reduce – problémy

- z principu **batch-oriented**
  - neumožňuje interaktivní odezvu
- málo flexibilní
  - Map a Reduce fáze jsou "zadrátované" v systému
- komplexní algoritmy komplikované pro programátora
  - nutné správně řídit data-flow
  - netriviální design patterny jsou náchylné na chyby
- iterativní algoritmy jsou neefektivní
  - například výpočet PageRanku

Map / Reduce – co dále?

- Hadoop YARN
  - "operating system for the cloud"
- umožnil vznik spoustě dalších systémů
  - typicky pracují s vyššími konstrukty (operátory), než prosté Map a Reduce
  - Spark, Tez, Storm, Giraph, ...
- zjednodušují programátorský pohled a zrychlují vývoj
- zatím většina ve velmi ranné fázi

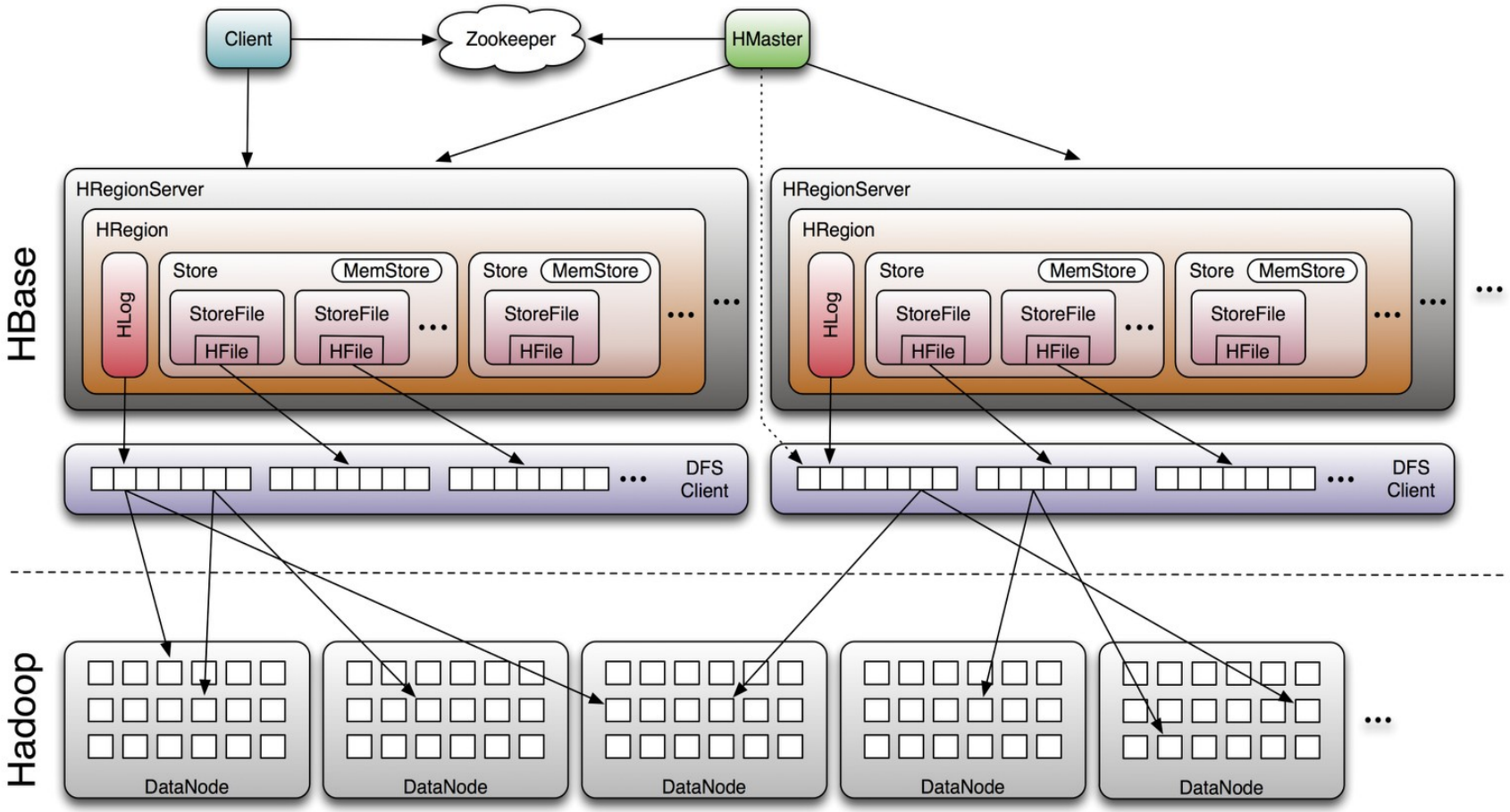
Co je NoSQL?

- dnes již trochu zavádějící, NoSQL databázím se dělají SQL vrstvy :-)
- odlišení od RDMS, většinou existují pouze primární indexy
- **CAP** teorém
  - Consistency, Availability, Partition tolerance
- RDMS operují na CA hraně
- pro horizontální škálovaní je **nutná** P hrana
  - můžeme volit CP nebo AP
- Cassandra, HBase, MongoDB, ...

HBase

- logicky se dá chápat jako nekonečně dlouhá řídká matice
- klíčem k datům trojice (klíč, sloupec, timestamp)
- umožňuje neúplné dotazy
  - například (klíč, sloupec, *), (klíč, *, *), (klíč*, sloupec, *)
- matice je horizontálně rozdělená na *regiony*
- regiony obsluhují **RegionServery**
  - jednodušeně jde o velké cache servery
- data uložena v HDFS

HBase


Client
Zookeeper
HMaster
HBase
HRegionServer
HRegion
HLog
Store
MemStore
StoreFile
HFile
DFS Client
Hadoop
DataNode

Proč Java?

- je Java dostatečně efektivní?
  - z praxe se ukazuje, že designové rozhodnutí pro Javu (JVM) bylo správné
  - efektivita za běhu vs. efektivita při vývoji
- je levnější nahradit drobnou neefektivitu za běhu nákupem HW
  - horizontálně škálovatelné algoritmy k tomu přímo vybízejí
- v Javě je vývoj rychlejší, než v low-level jazycích
  - časově kritické operace lze i tak psát nativně
- většina času se "propálí" při IO operacích
  - čtení z disku, síťová komunikace

A jak tedy nacrawlovat 1,5G dokumentů?

- obrnit se trpělivostí :-)
- nevynalézat kolo – je mnohem lepší a efektivnější použít existující (ideálně) open source řešení
  - levnější je podílet se na vývoji s open source komunitou, než psát vlastní řešení na zelené louce
- frameworky bohužel nejsou všespásné
  - i s nejlepším frameworkem se narazí na problémy se škálovatelností a stabilitou

Problémy velkých dat

- při vývoji algoritmů nemáte k dispozici produkční prostředí
  - produkční nasazení potom s sebou nese několik iterací, neboť velká data důkladně prověří skutečnou stabilitu kódu
- špatný návrh algoritmu vede k "hotspotování"
  - nerovnoměrné vytěžování clusteru, různé nody jsou různě zatížené
  - systém je pouze tak rychlý, jak rychlý je nejpomalejší člen

1,5G dokumentů

- ~ 500 serverů
  - ~ 10000 jader
  - ~ 5 PiB raw prostor
  - ~ 25 TiB RAM
- 30 miliard záznamů v databázi
- desítky analytických a agregačních úloh denně
- desítky rankovacích a prediktivních úloh týdně
- 200 M denně stažených dokumentů

1,5G dokumentů

- 180 miliard odkazů mezi dokumenty
- 30 miliard záznamů v databázi
- zhruba 700 milionů dokumentů zaindexováno
- z toho 150 milionů dokumentů se objeví v SERPu
- 18 milionů dotazů denně
- 3,5 milionů unikátních dotazů denně
- 17 milionů unikátních dotazů týdně

Otázky?

http://vyvojari.seznam.cz/

http://seznam.sprace.cz/

...najdu tam, co neznám!

Děkuji za pozornost

Ing. Jan Lukavský, vedoucí týmu vývoje, jan.lukavsky@firma.seznam.cz