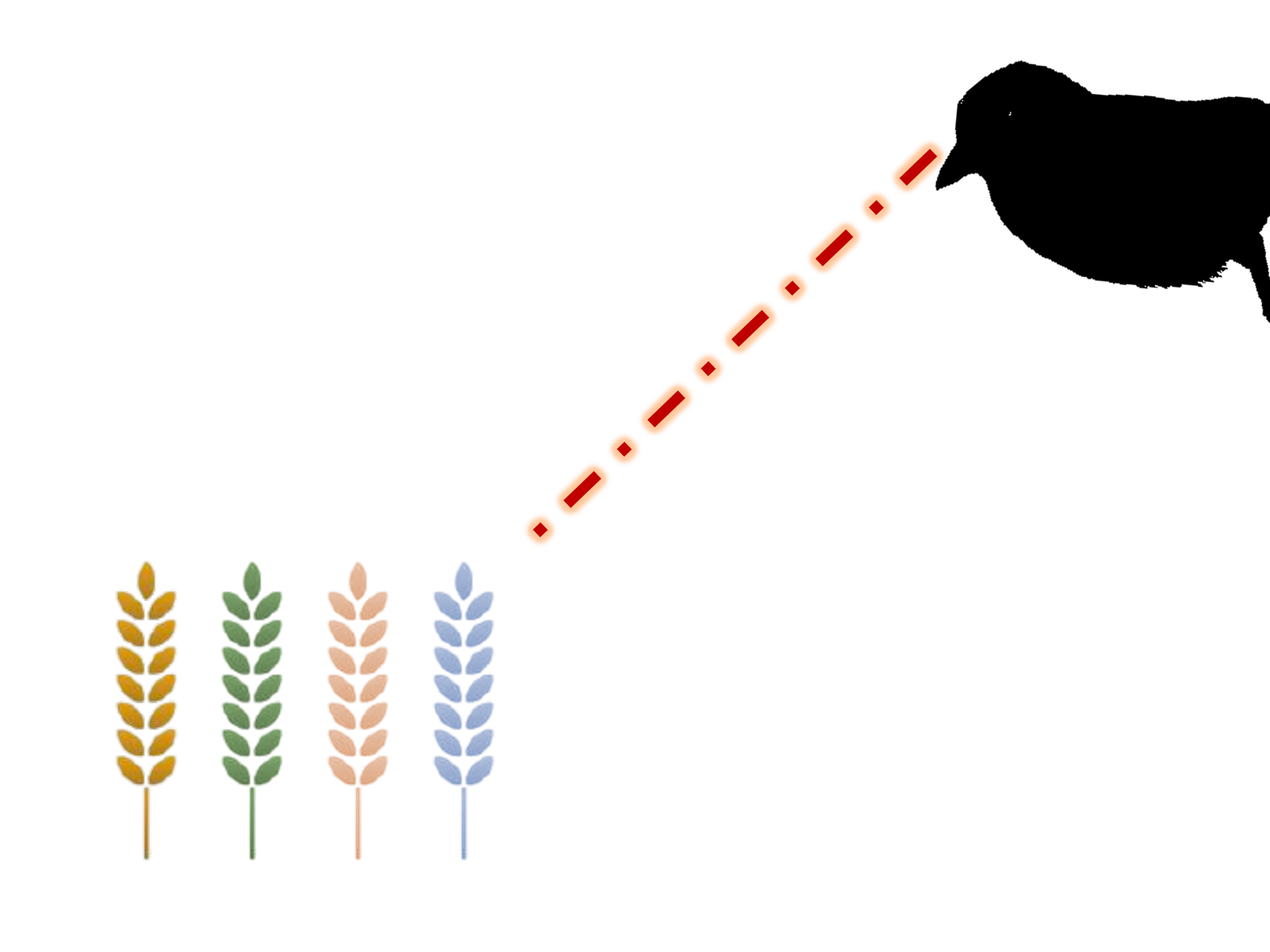
Kafka

@RoboNovotny

jeseň 2023



Čo je Kafka?

- Apache Kafka
- pôvodne v LinkedIn na zber clickstreamov

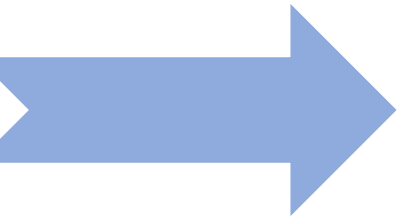
Čo je Kafka?

distribuovaný commit log

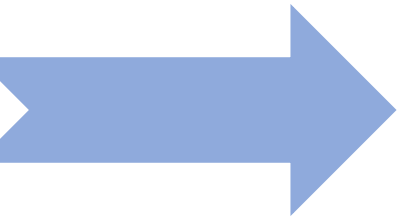
Čo je Kafka?

distributed
streaming platform

Producenti a konzumenti



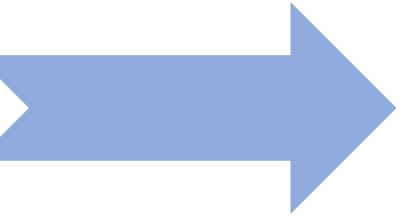
Záznamy a témy | records and topics



- kľúč (voliteľný)
- dáta

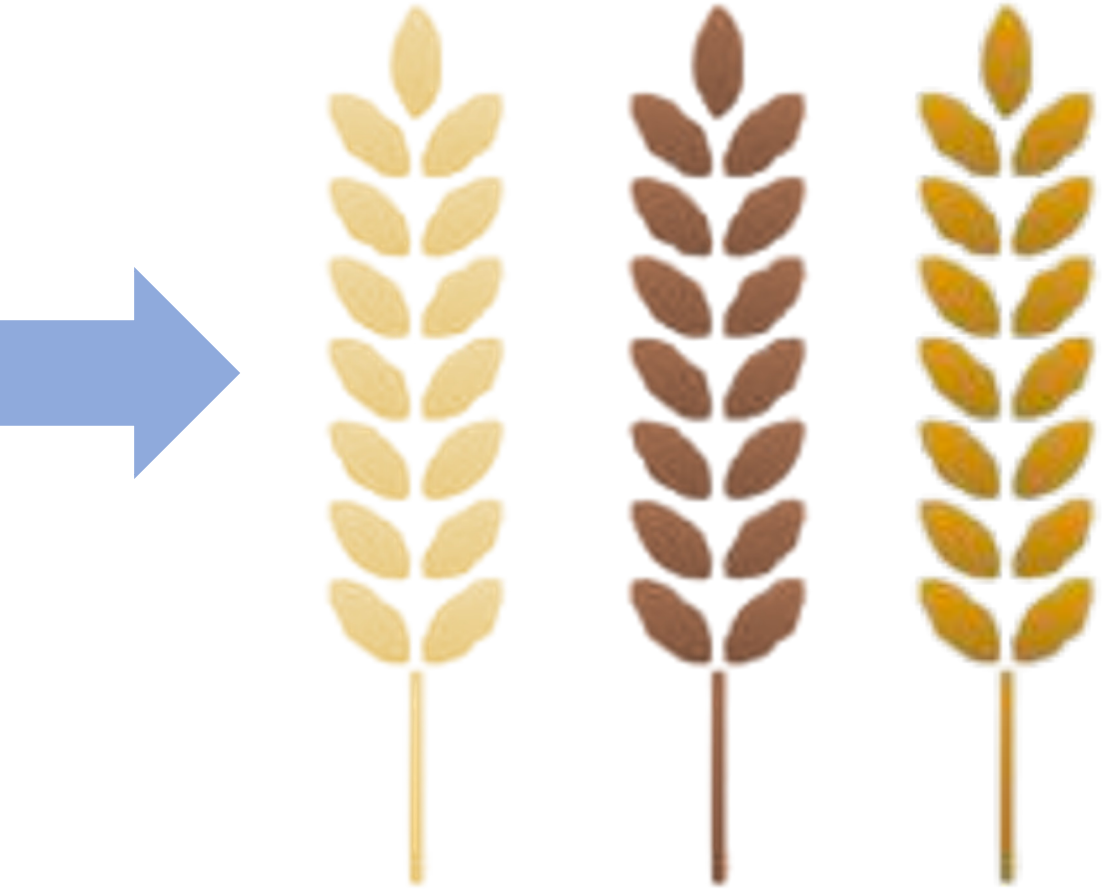
polia bajtov

Téma | topics



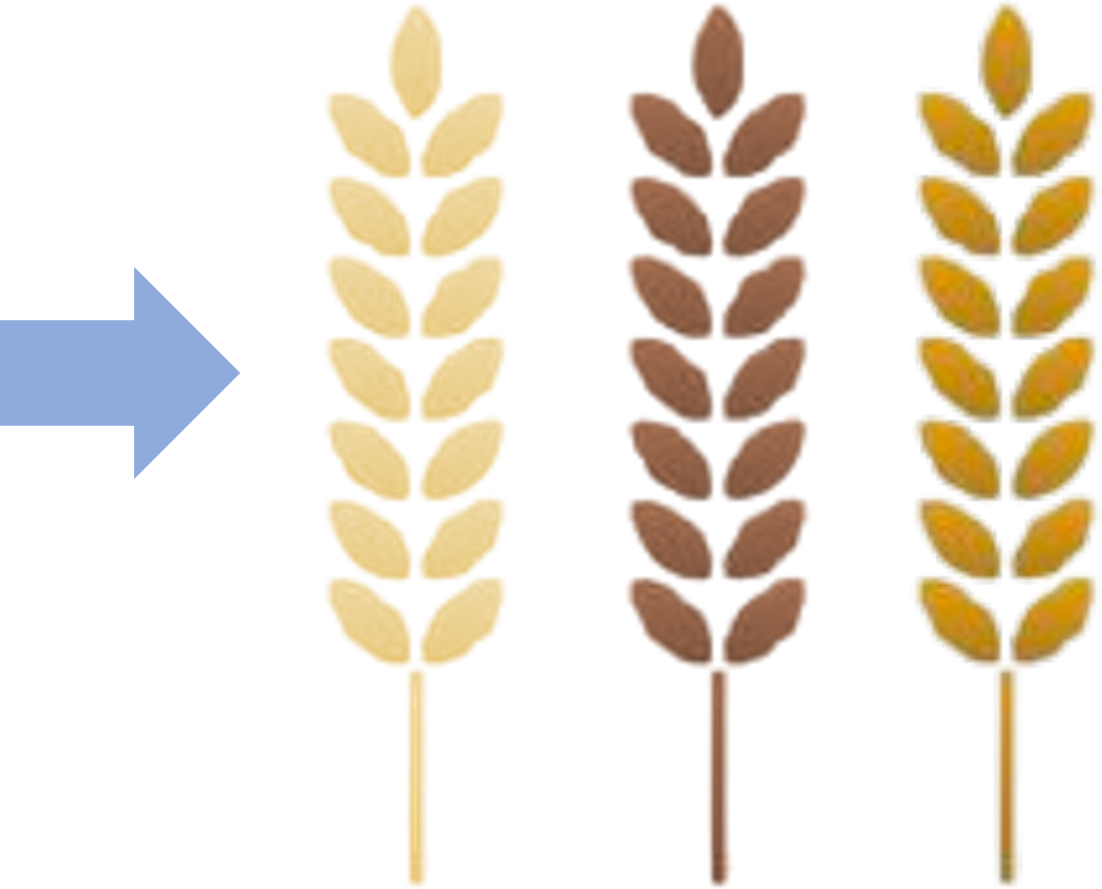
- len pridávame na koniec
- čítame zo začiatku

Konzumenti | Consumers



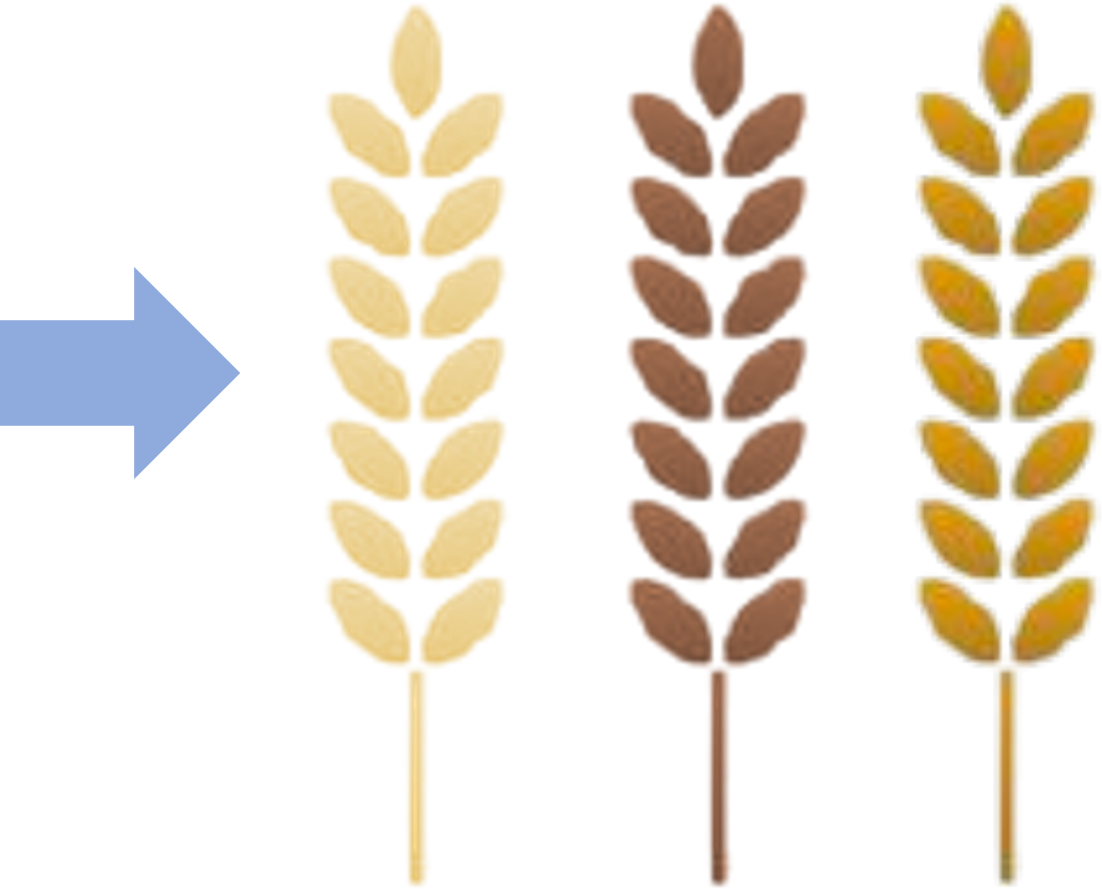
záznamy z topicu sa po konzumácii nestrácaju!

Konzumenti | Consumers



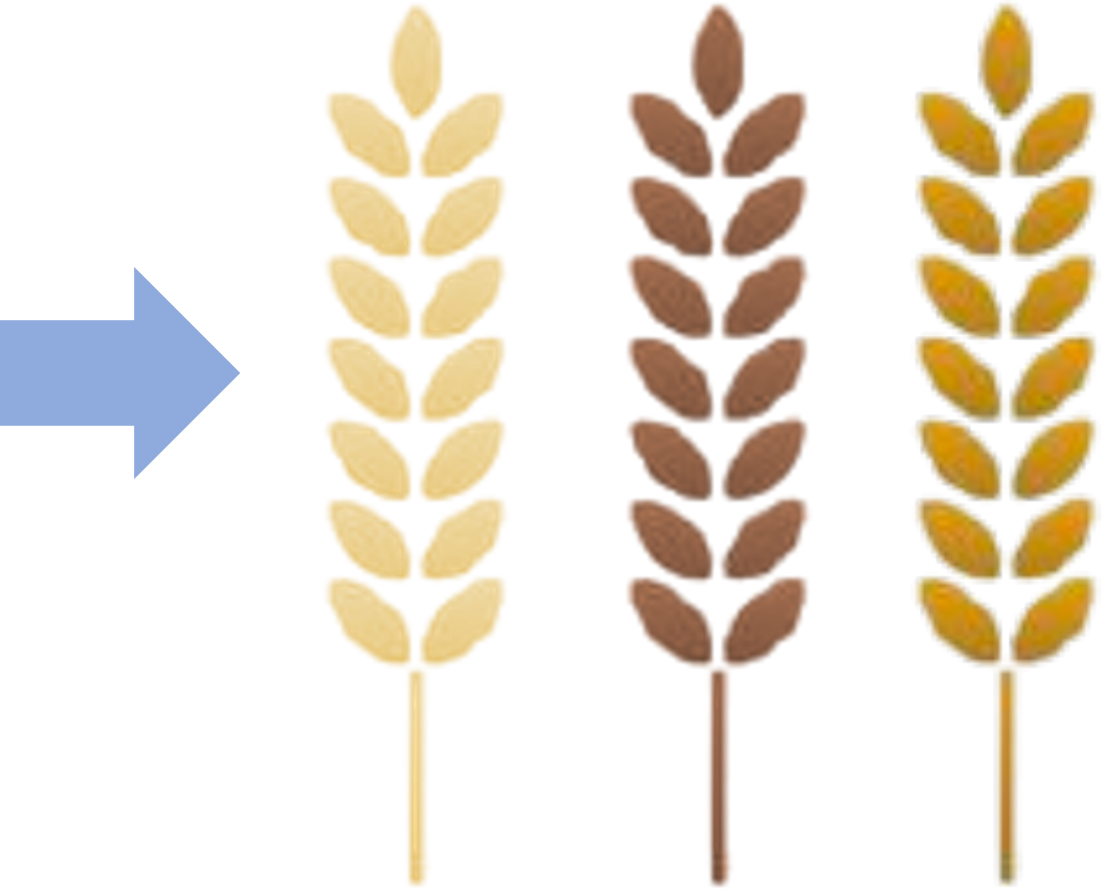
topic definuje pravidlá pre expiráciu záznamov

Konzumenti | Consumers



konzument môže čítať topic viacnásobne

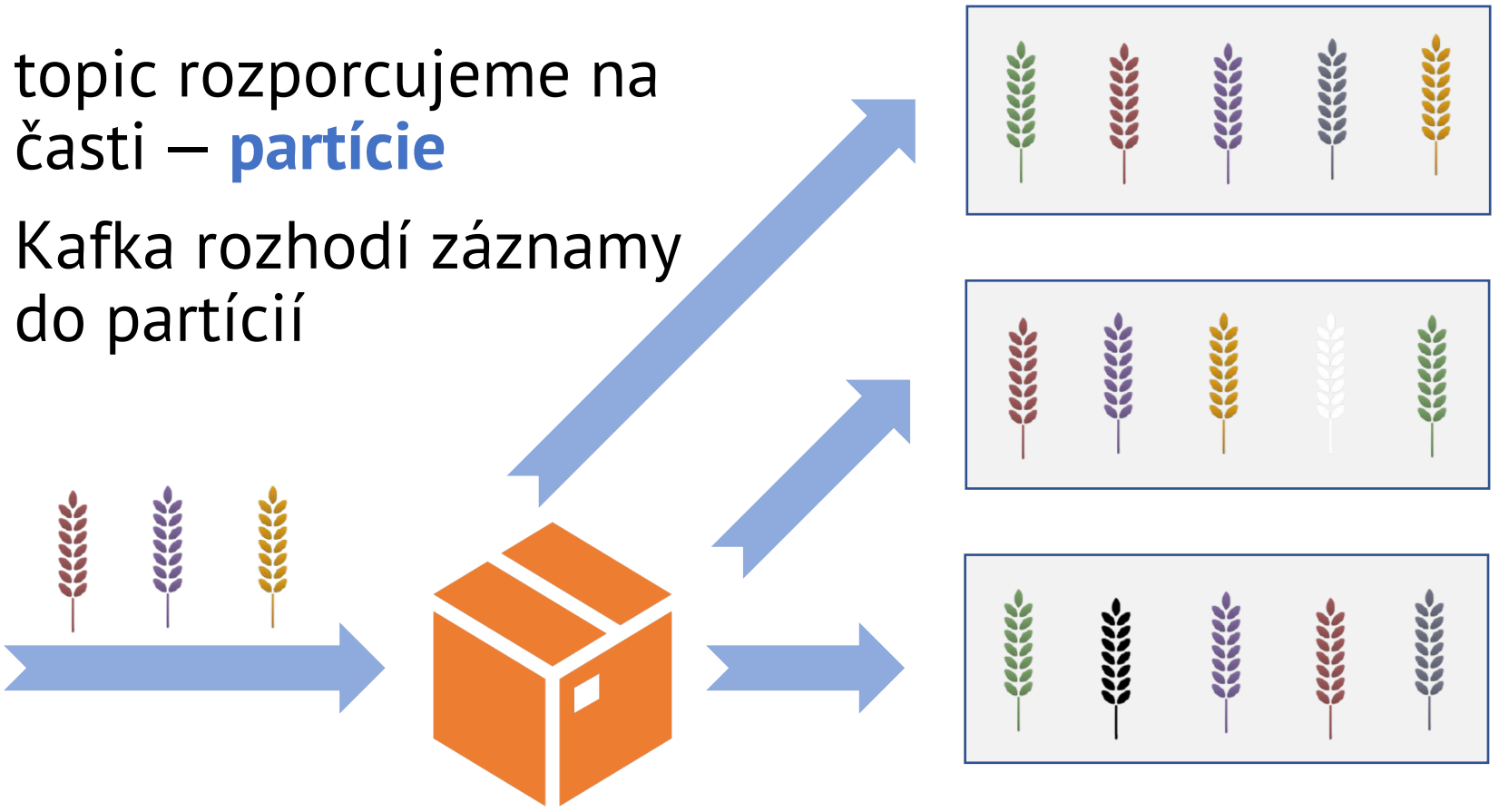
Konzumenti | Consumers



konzument si pamätá offset, od ktorého začne čítať záznamy

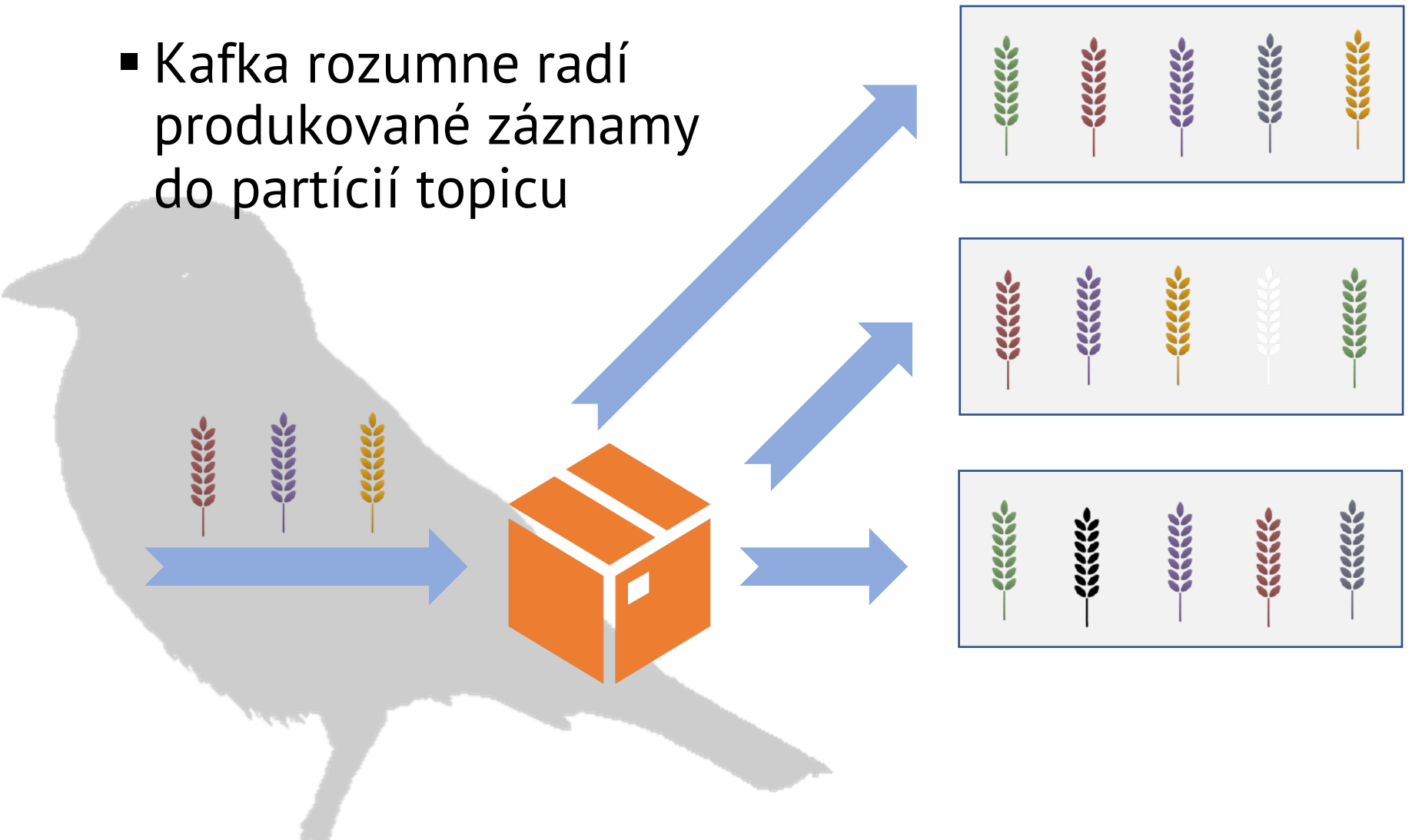
Ako škálovať?

- topic rozporcujeme na časti – **partície**
- Kafka rozhodí záznamy do partícií



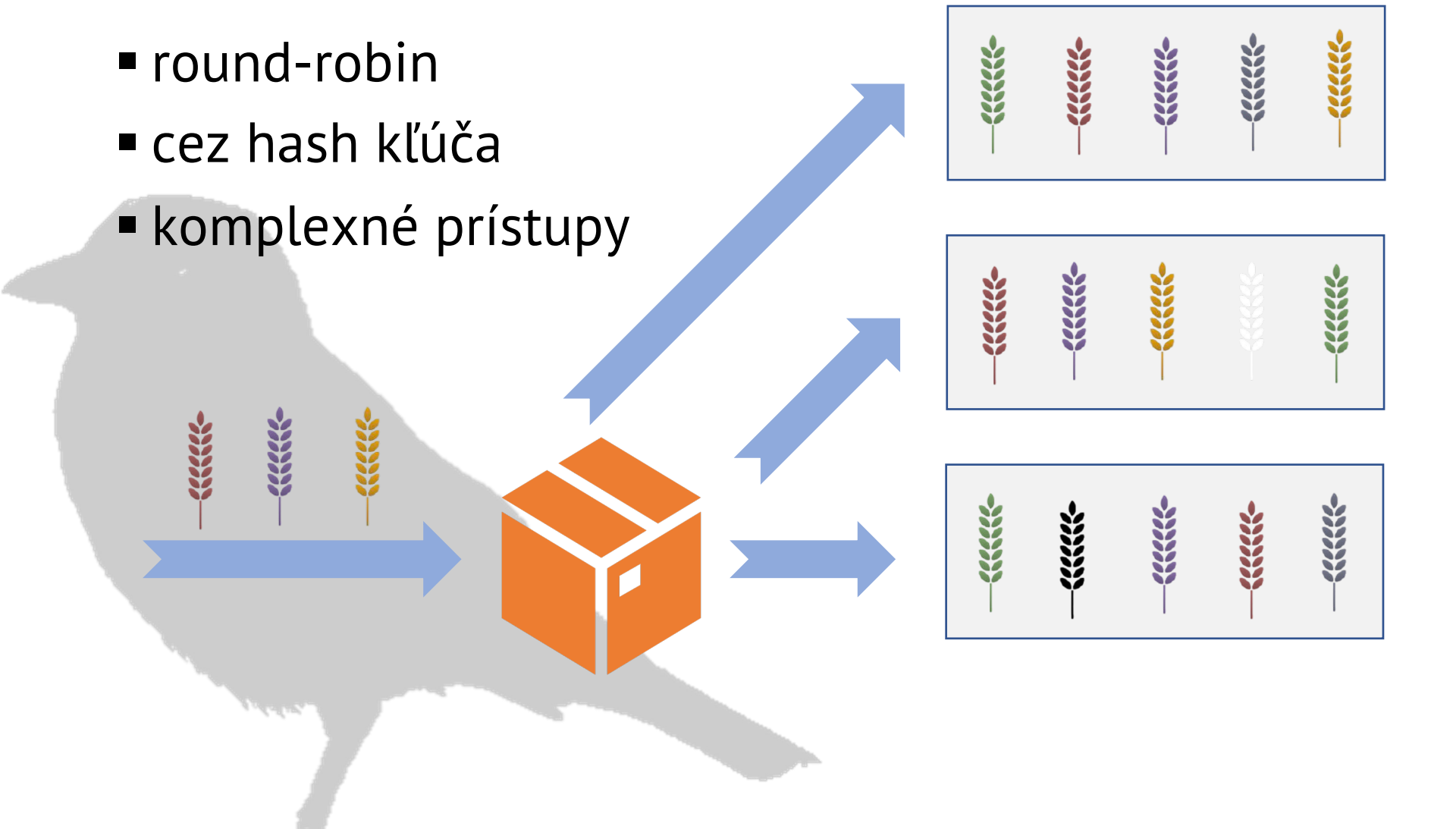
Partitioner: radenie záznamov

- Kafka rozumne radí produkované záznamy do partícií topicu



Partitioner: radenie záznamov

- round-robin
- cez hash kľúča
- komplexné prístupy



Škálovanie konzumentov

konzumenti sa zhluknú
do **consumer group**

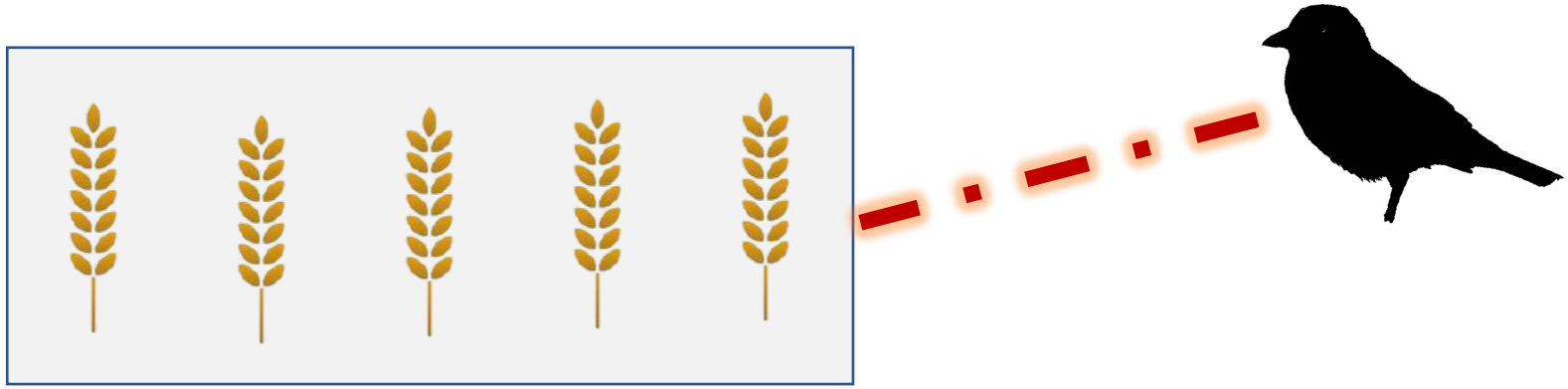
členovia grupy si delia záznamy z topicu

Škálovanie konzumentov

- členovia groupy si delia záznamy z topicu
- viacero grúp na topicu dosiahne broadcast

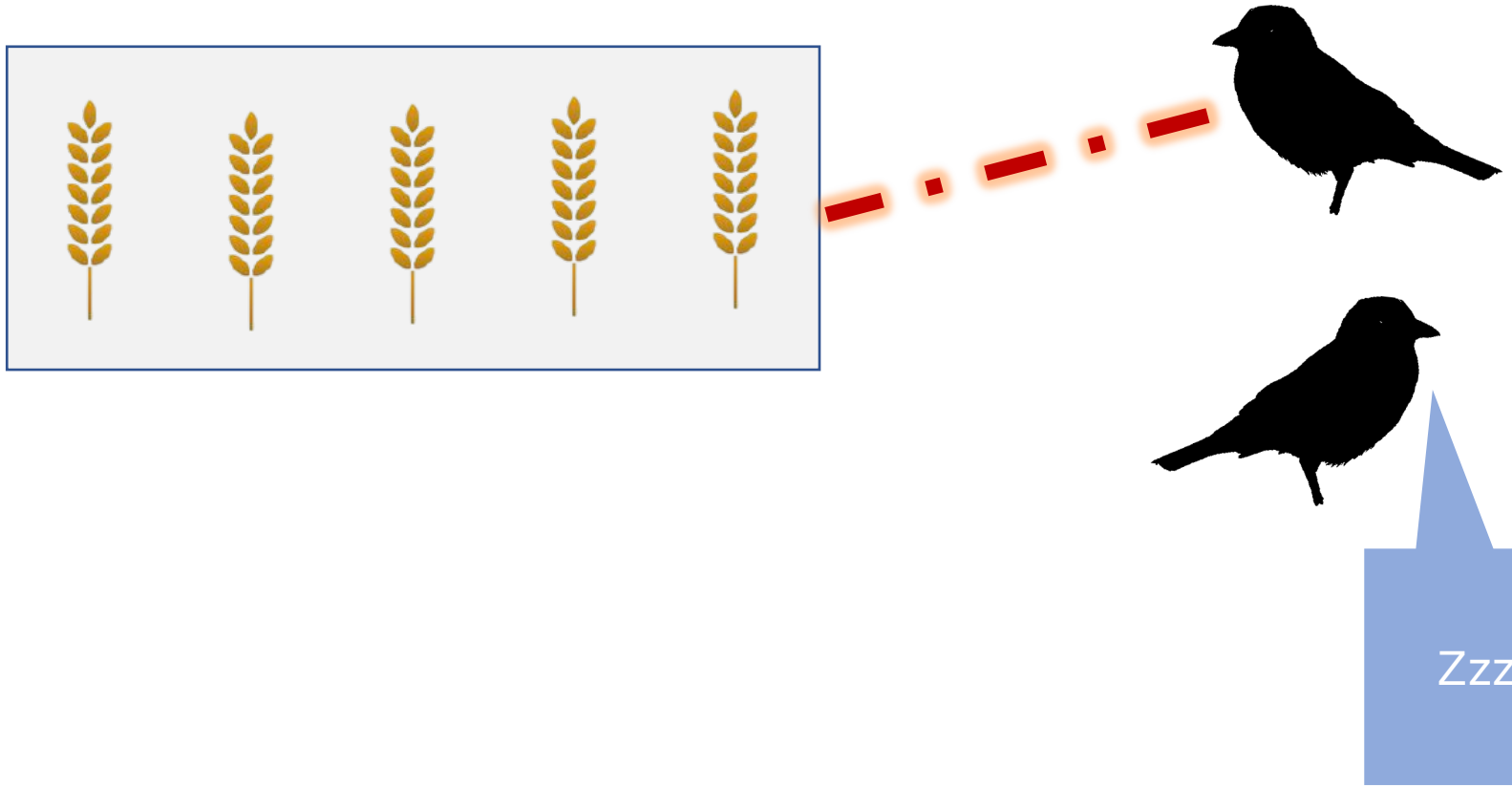
Príklad:

1 topic, 1 partícia,
1 consumer group, 1 člen



Príklad:

1 topic, 1 partícia,
1 consumer group, 2 členovia

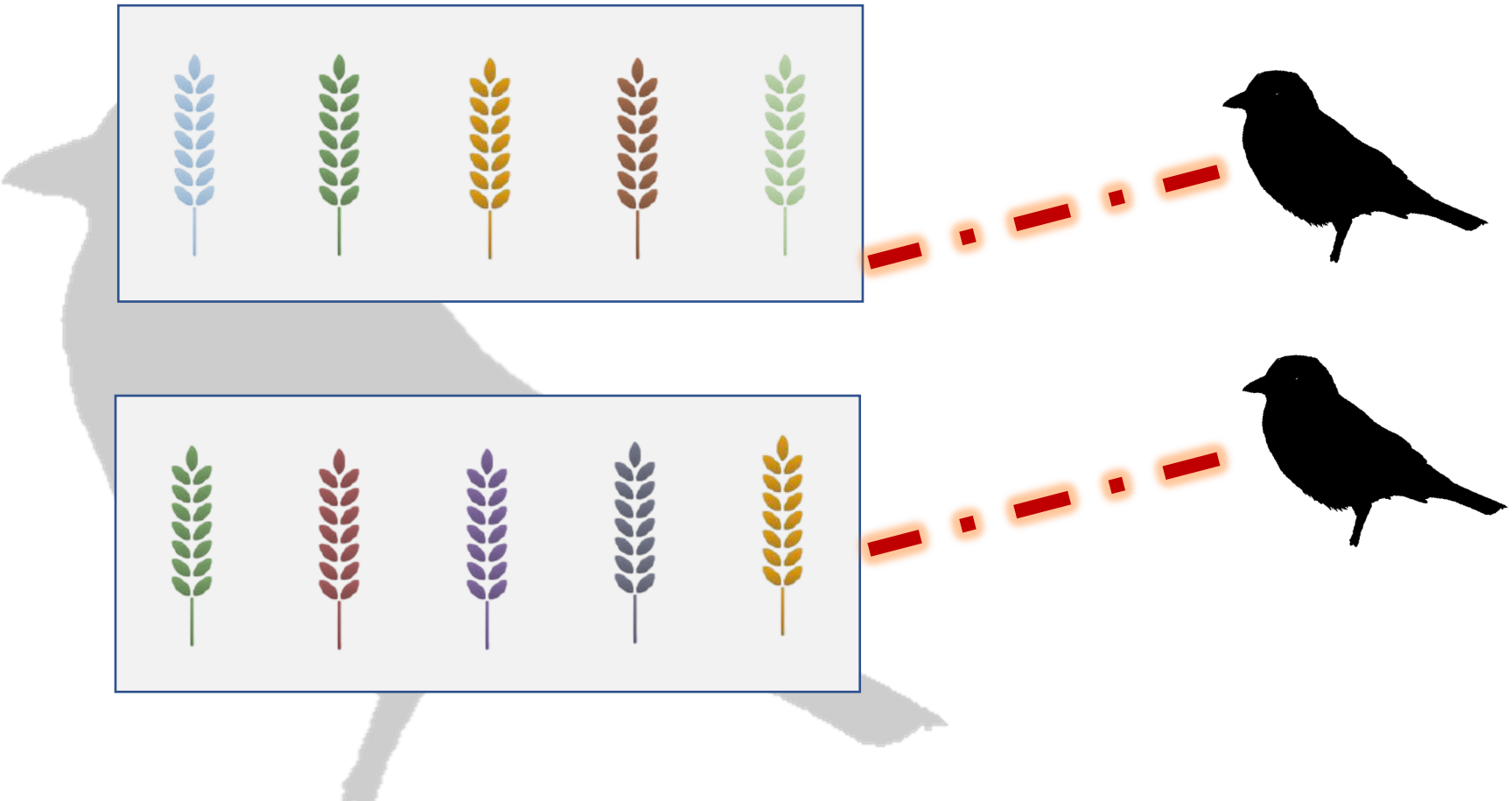


Pravidlo

Z 1 partície konzumuje najviac 1 člen consumer groupy

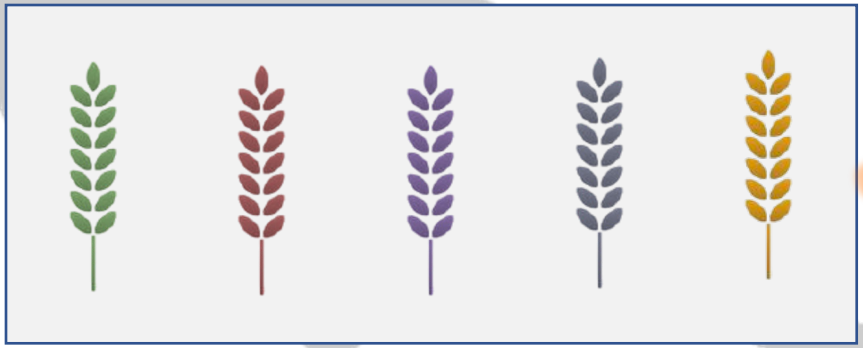
Príklad:

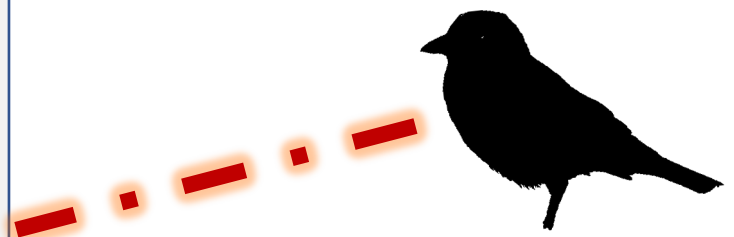
1 topic, 2 partície,
1 consumer group, 2 členovia



Príklad:

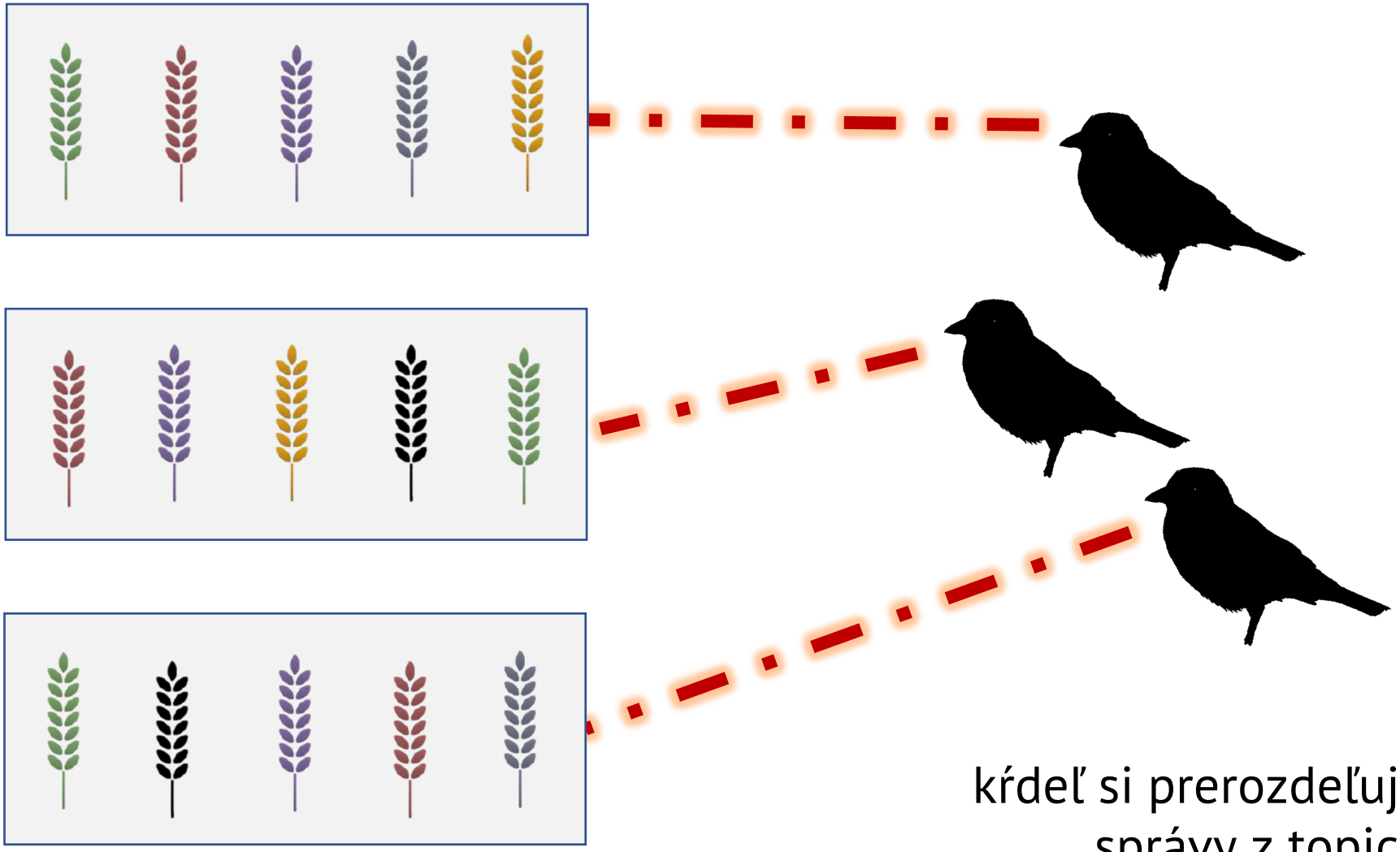
1 topic, 2 partície,
1 consumer group, 2 členovia





Príklad:

1 topic, 3 partície — 1 consumer group, 3 členovia



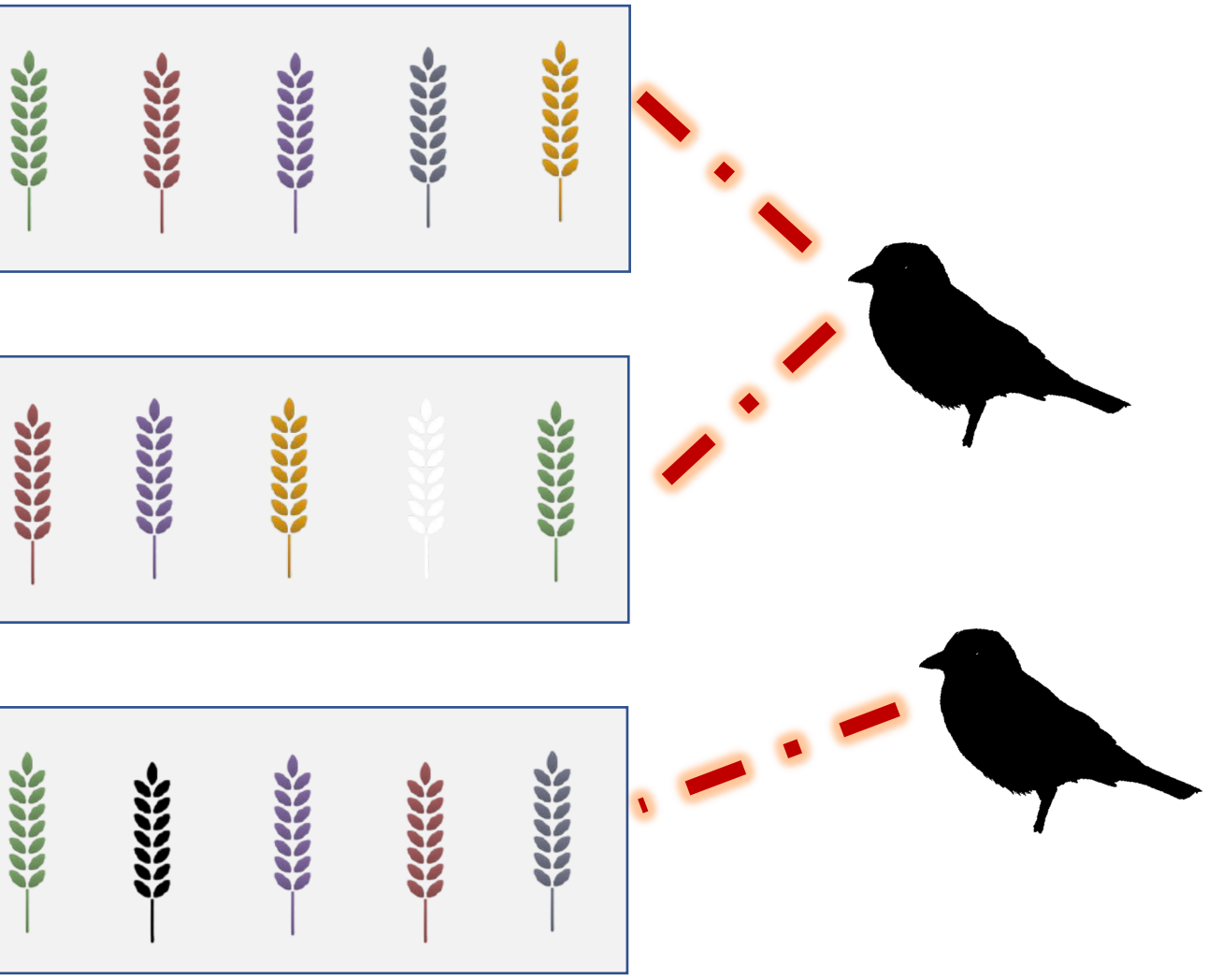
kŕdeľ si prerozdeľuje správy z topicu

Partition Assignor: priraďovanie partícií konzumentom

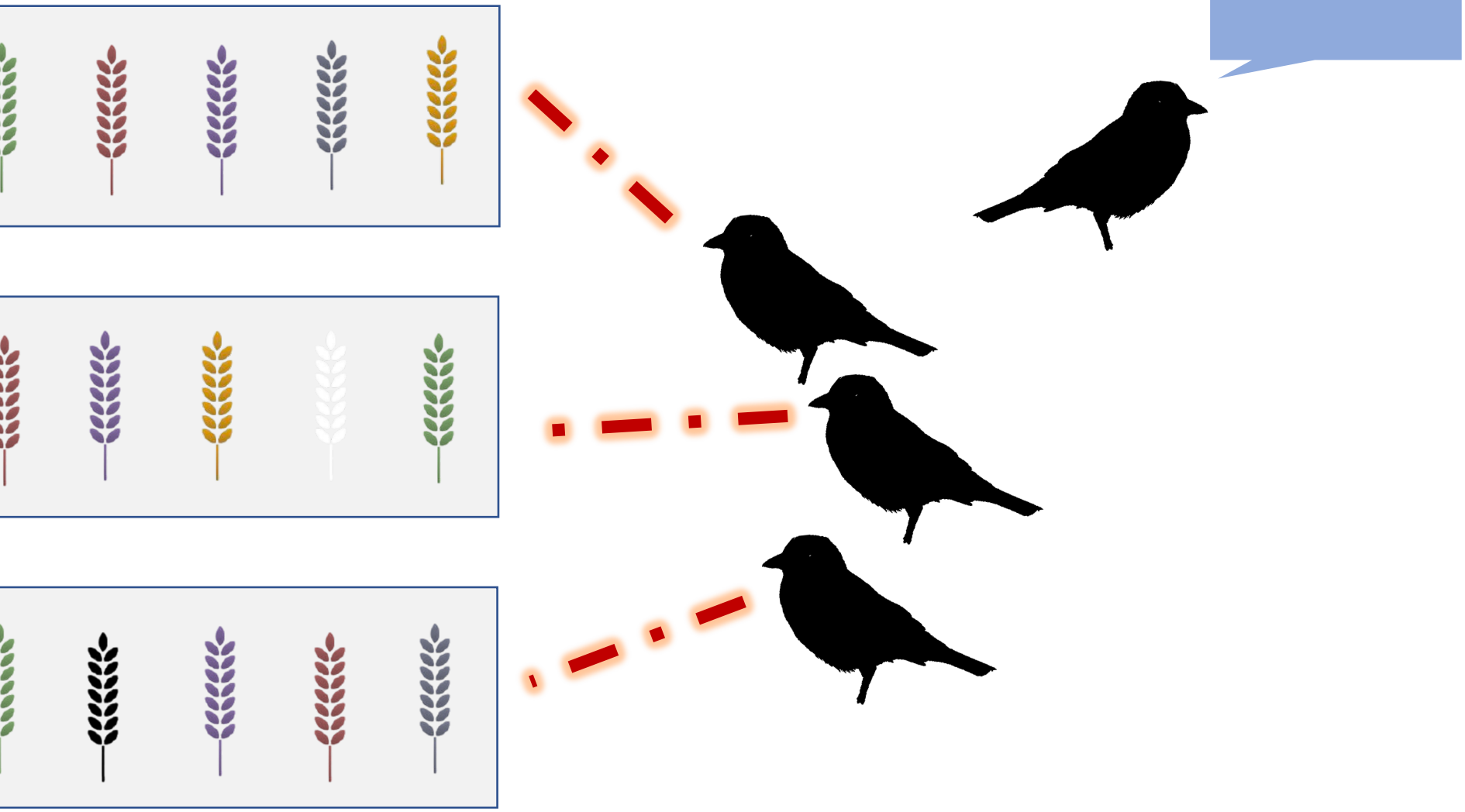
Pravidlo: Partícia je konzumovaná najviac 1 členom grupy

Pravidlo: Člen grupy konzumuje žiadnu, jednu, alebo viacero partícií.

Partícií > konzumentov v skupine?

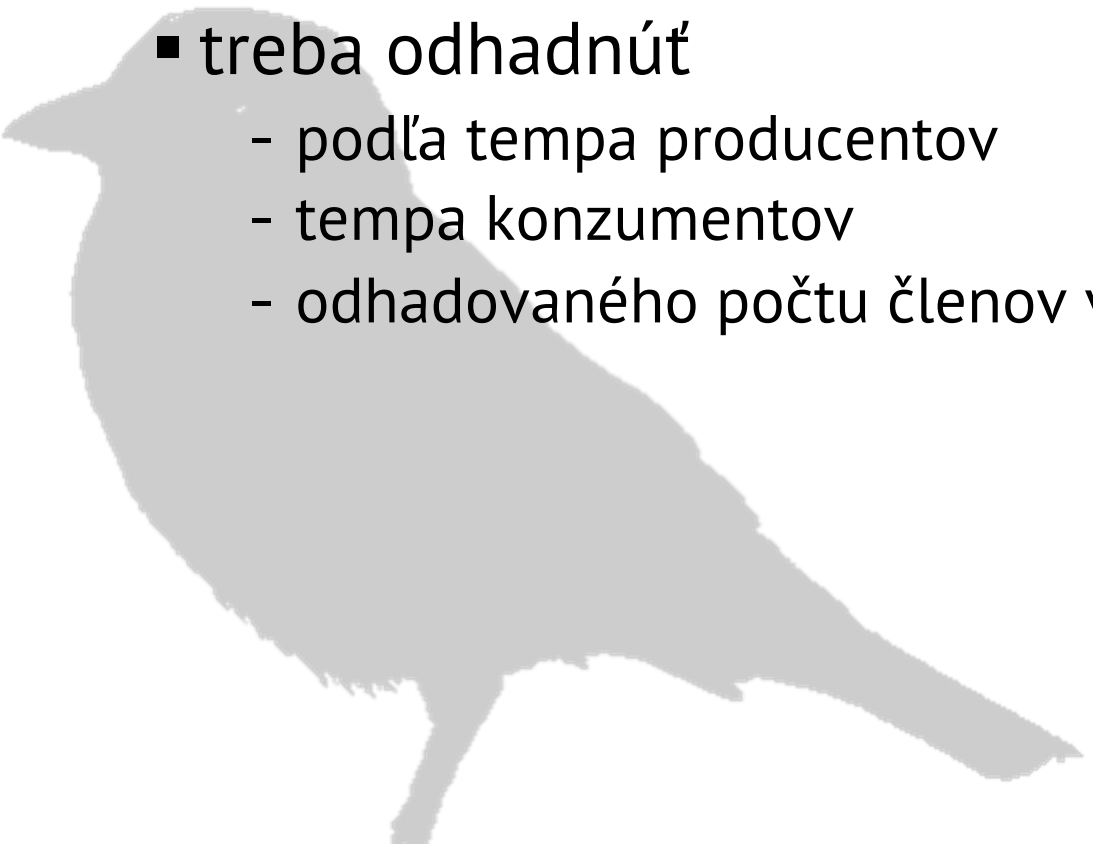


Partícií < konzumentov v skupine?



Partície = mechanizmus škálovania

- partície nastavujeme pri tvorbe topicu
- navýšiť je možné, znížiť nie
- treba odhadnúť
  - podľa tempa producentov
  - tempa konzumentov
  - odhadovaného počtu členov v consumer grupe



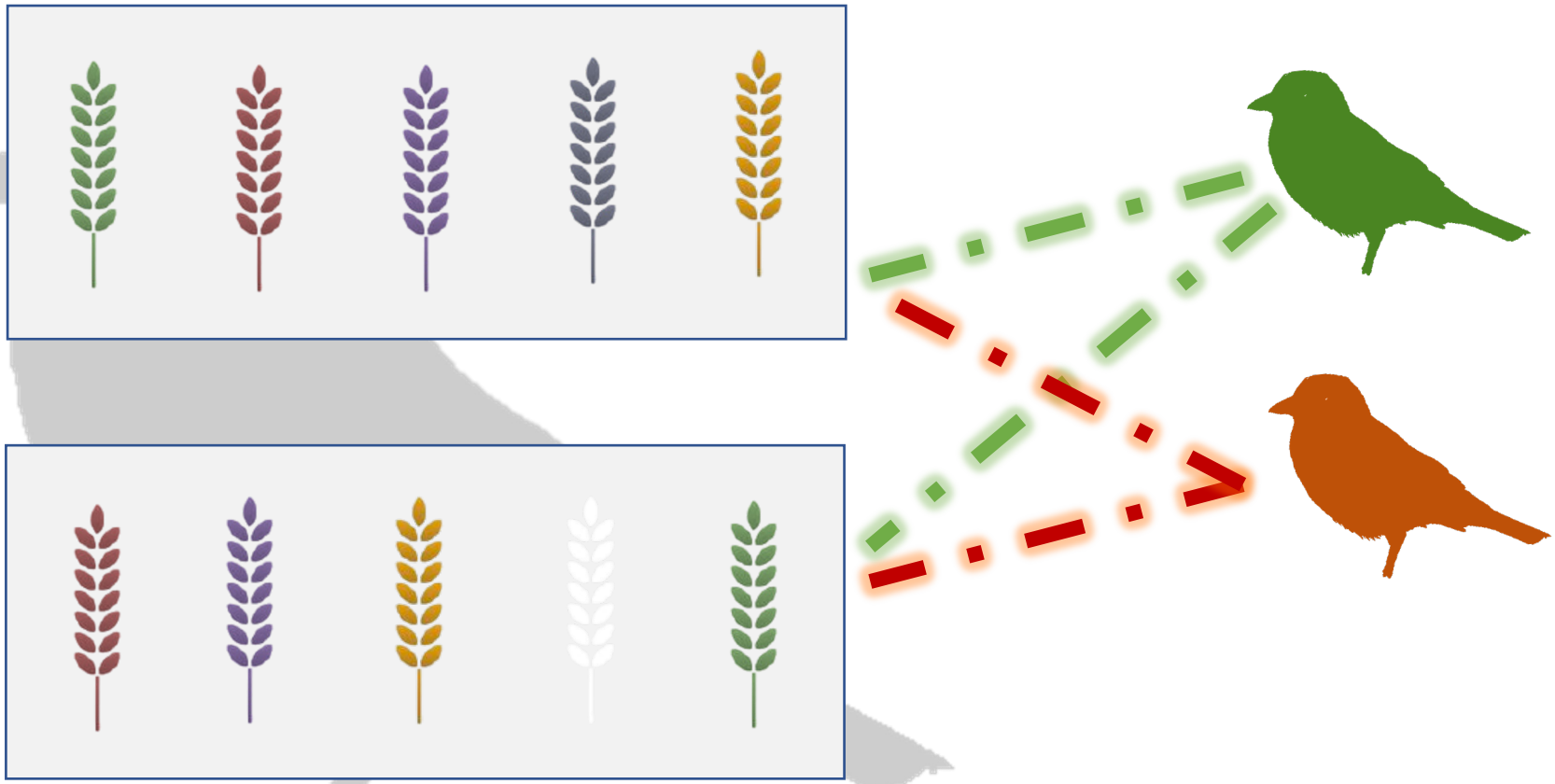
Škálovanie konzumentov

- členovia groupy si delia správy z topicu
- viacero grúp na topicu dosiahne broadcast

Príklad:

1 topic, 1 partícia –

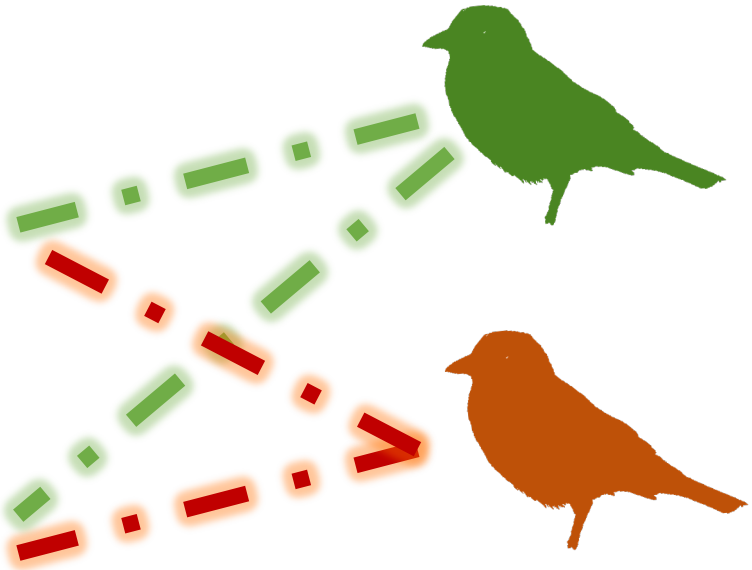
2 consumer groupy po 1 člene



Príklad:

1 topic, 1 partícia –

2 consumer groupy po 1 člene



Broadcast!

Rýchli konzumenti

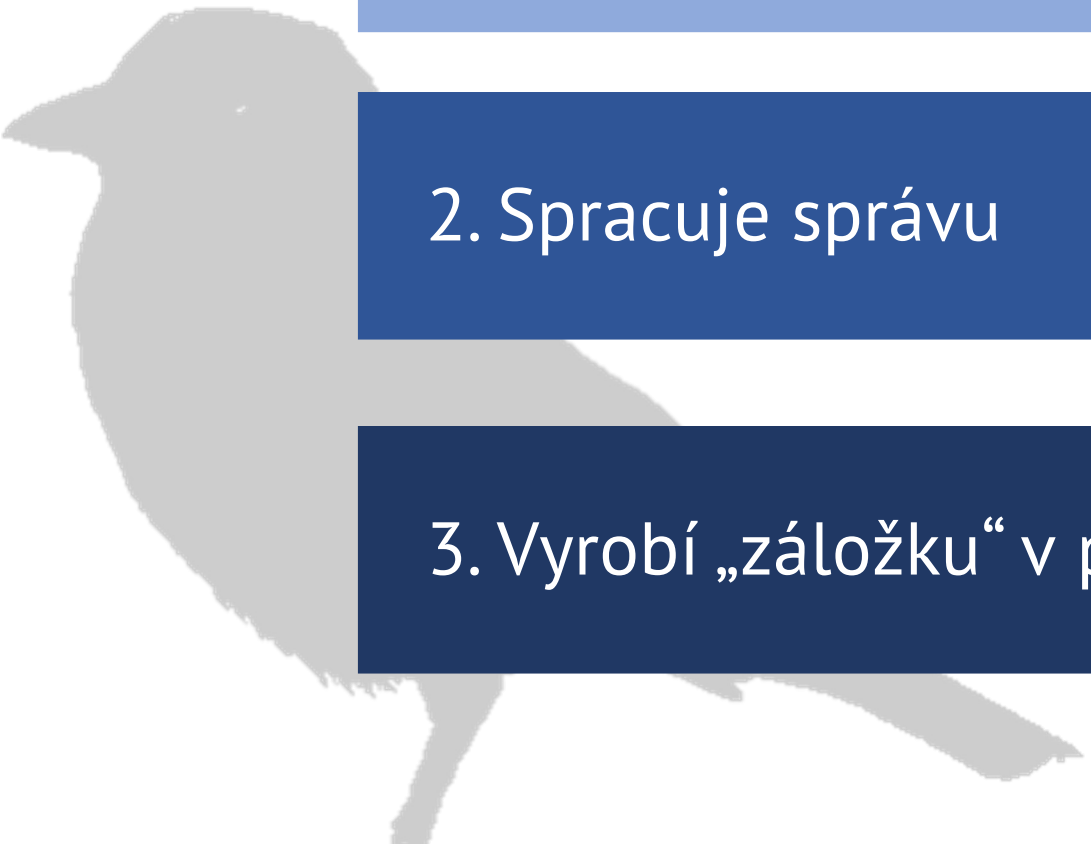
- Kafka nemá acknowledgement konzumentov
- „Tu máte partície, čítajte si od ktorého ofsetu chcete“.

Prostý konzument priradený k partícii číta správy

- latest: „odteraz čítam nové správy“
- earliest: od najstaršieho-najmenšieho offsetu

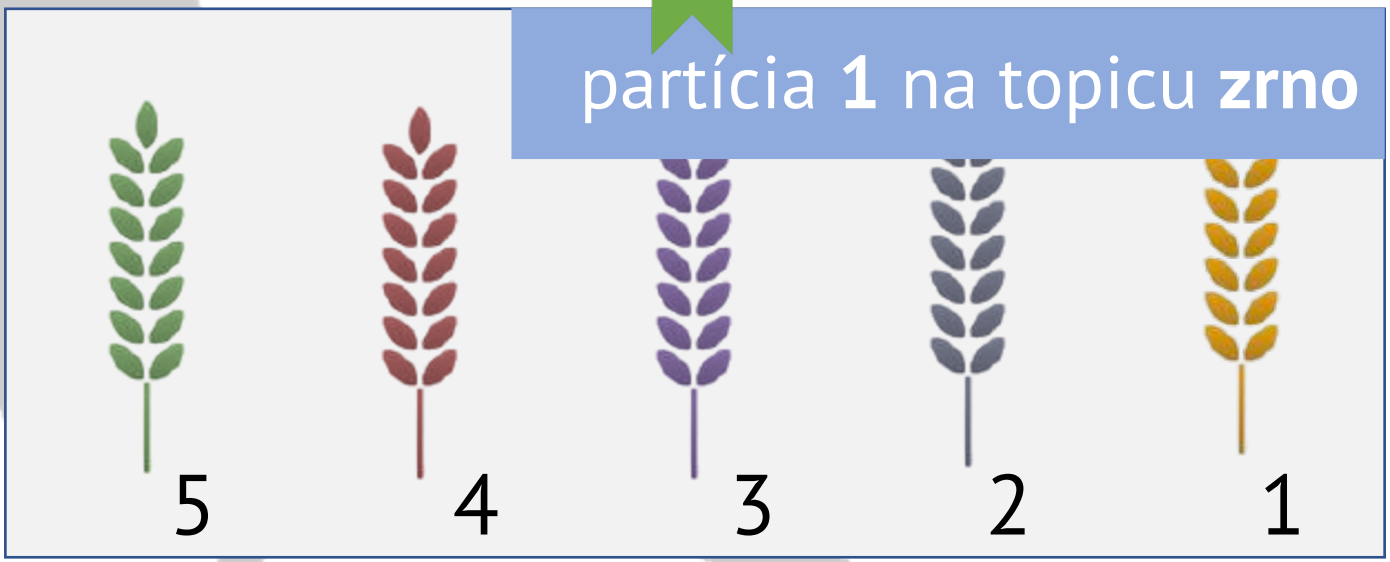
Inteligentný konzument

1. Prečíta správu z partície
2. Spracuje správu
3. Vyrobí „záložku“ v partícii na danom ofsete

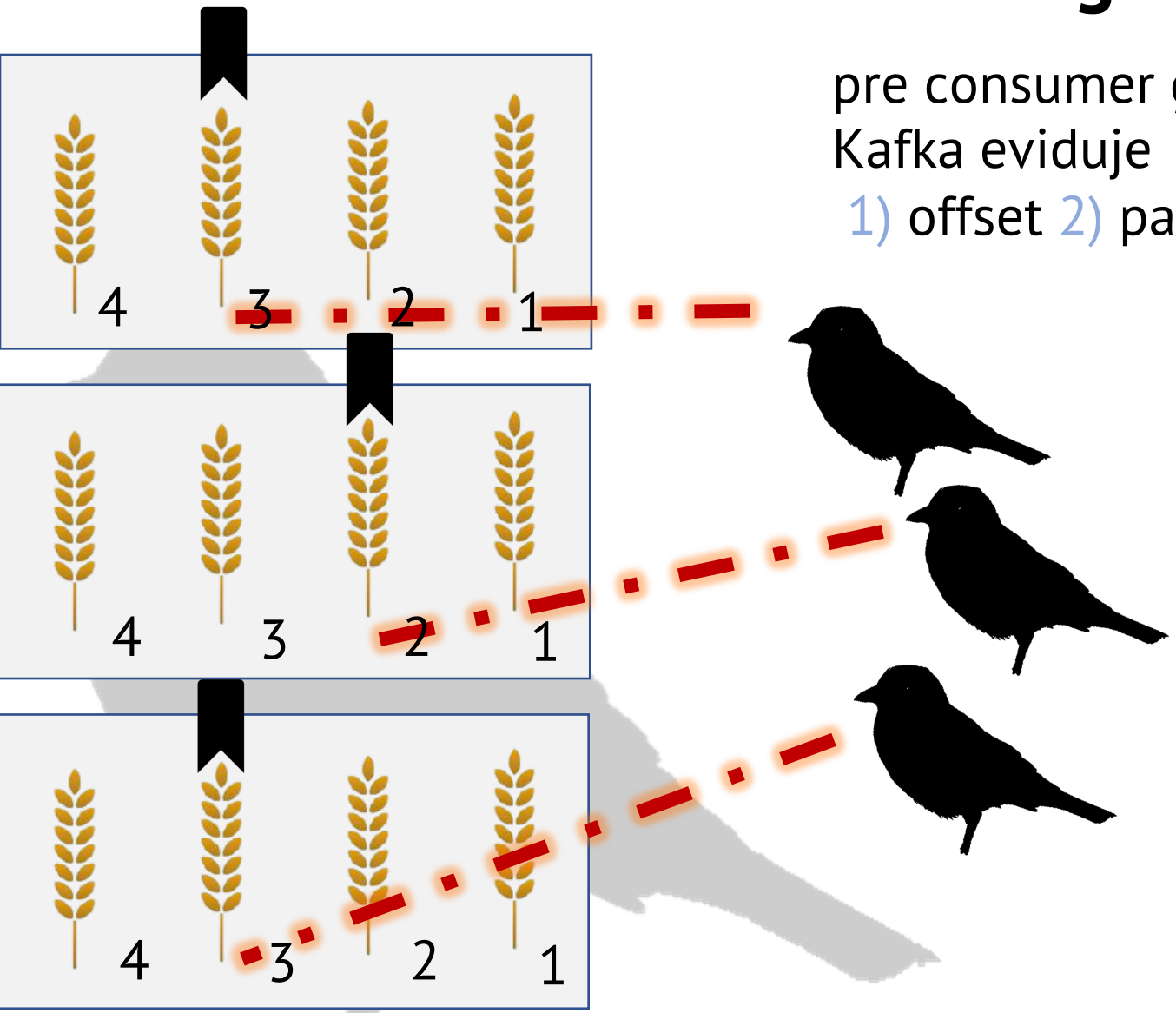


konzument spracuje 3 správy a

do Kafky
za zelenú consumer groupu
komitne offset 3
do partície **1**
na topicu **zrno**



Viacero členov v consumer grupe



pre consumer grupu
Kafka eviduje
1) offset 2) partície 3) topicu

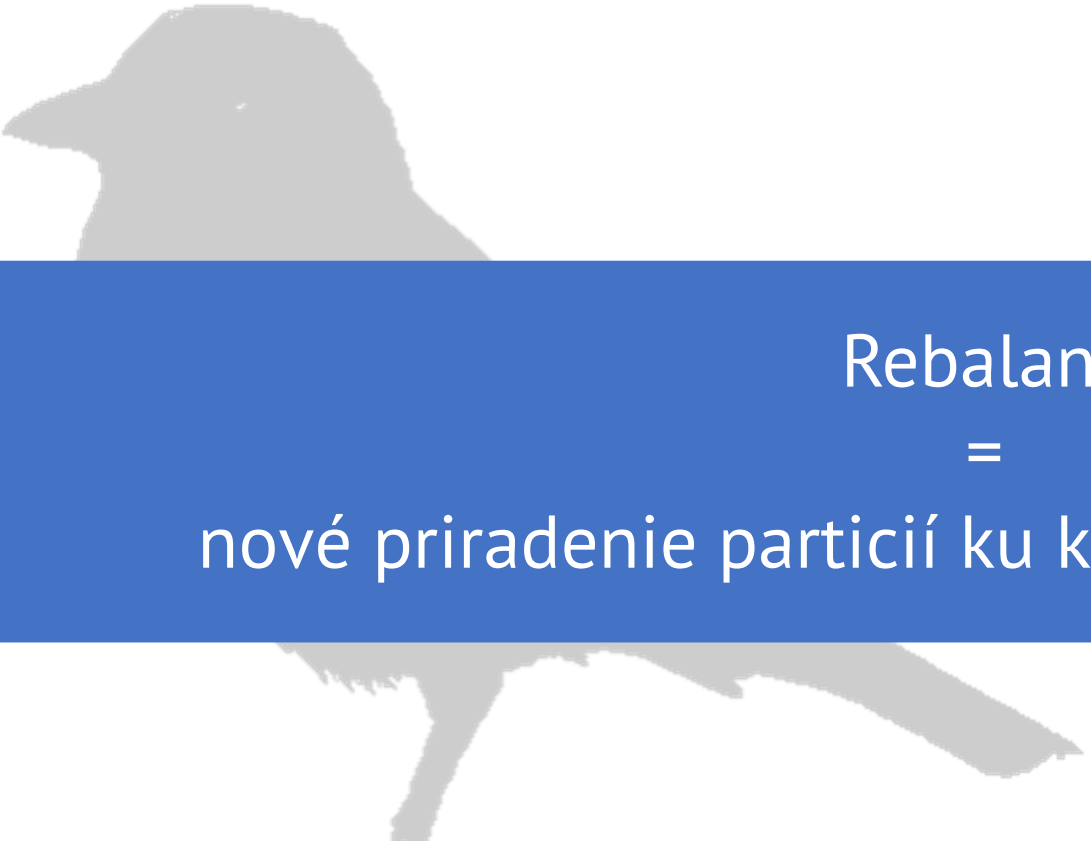
Spoľahliví konzumenti

- konzument v skupine môže **commitnúť** offset v partícii topicu

„Haló, Kafka?! Hlásim, že za šedý kŕdeľ som z tretej partície spracoval správy po offset 3“

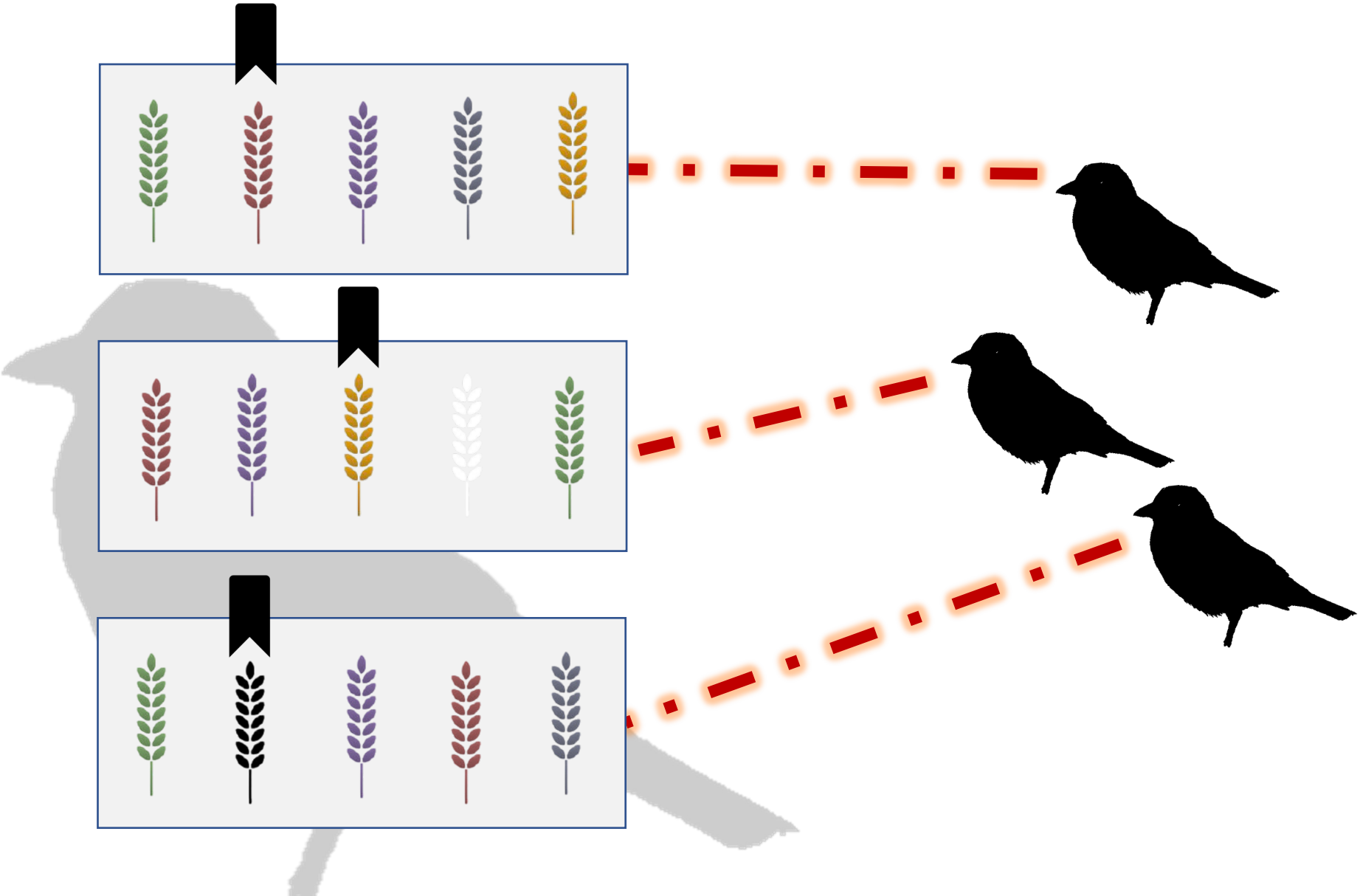
Rebalans

- konzumenti môžu odpadávať
- ale konzumenti môžu aj pribúdať

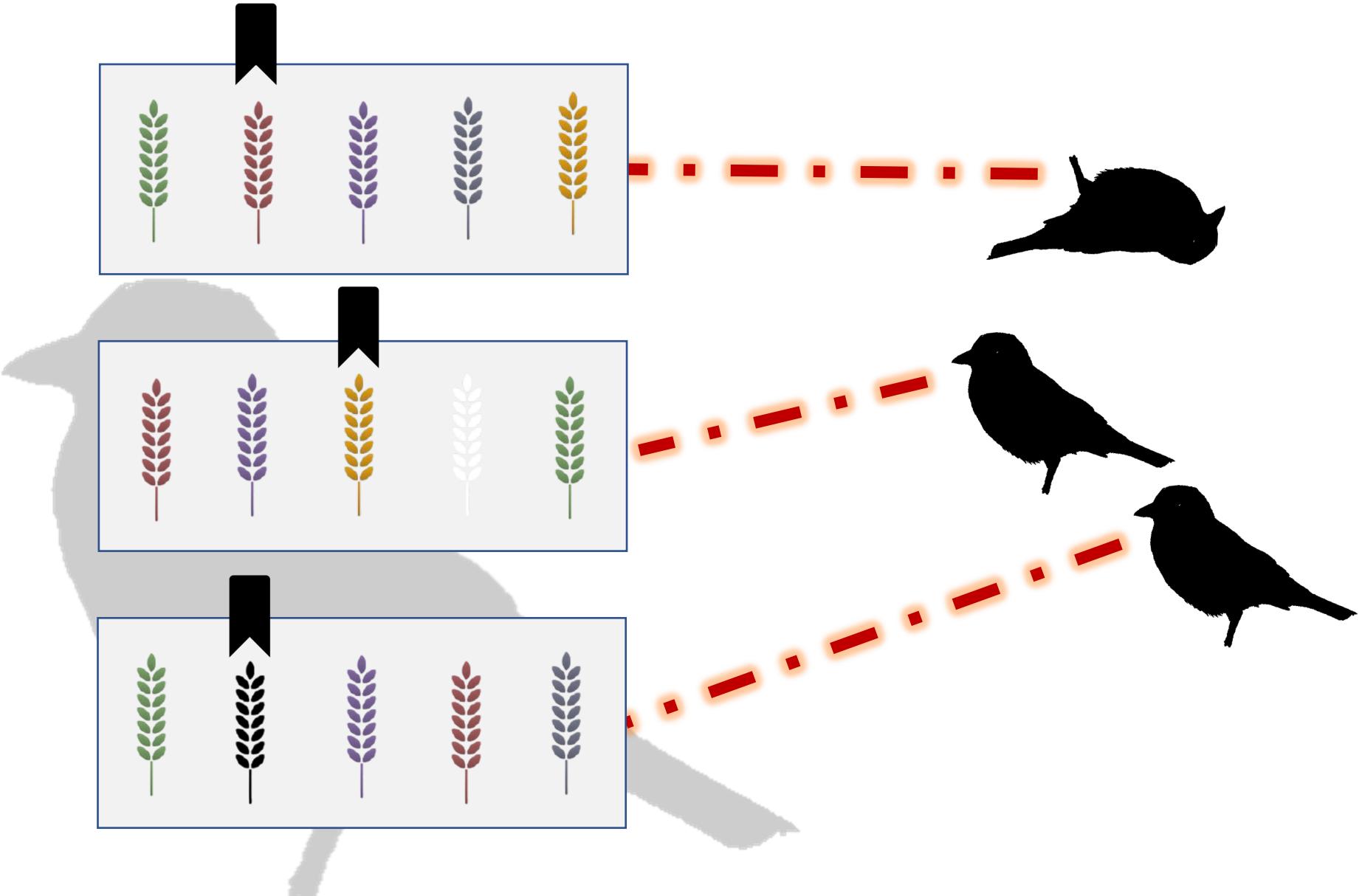


Rebalans
=
nové priradenie particií ku konzumentom v grupe

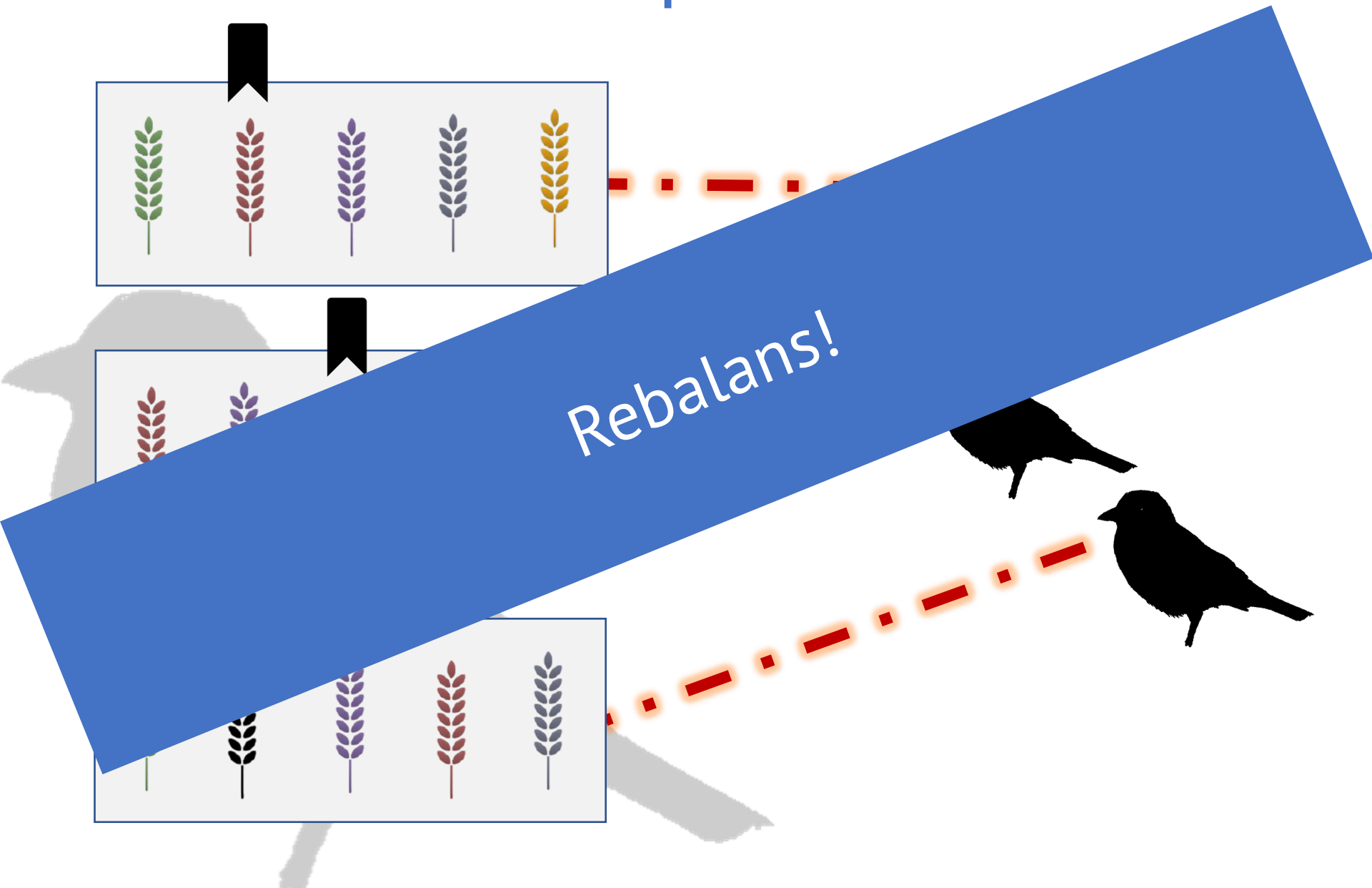
Pôvodný stav: 3 partície, 3 konzumenti v groupe



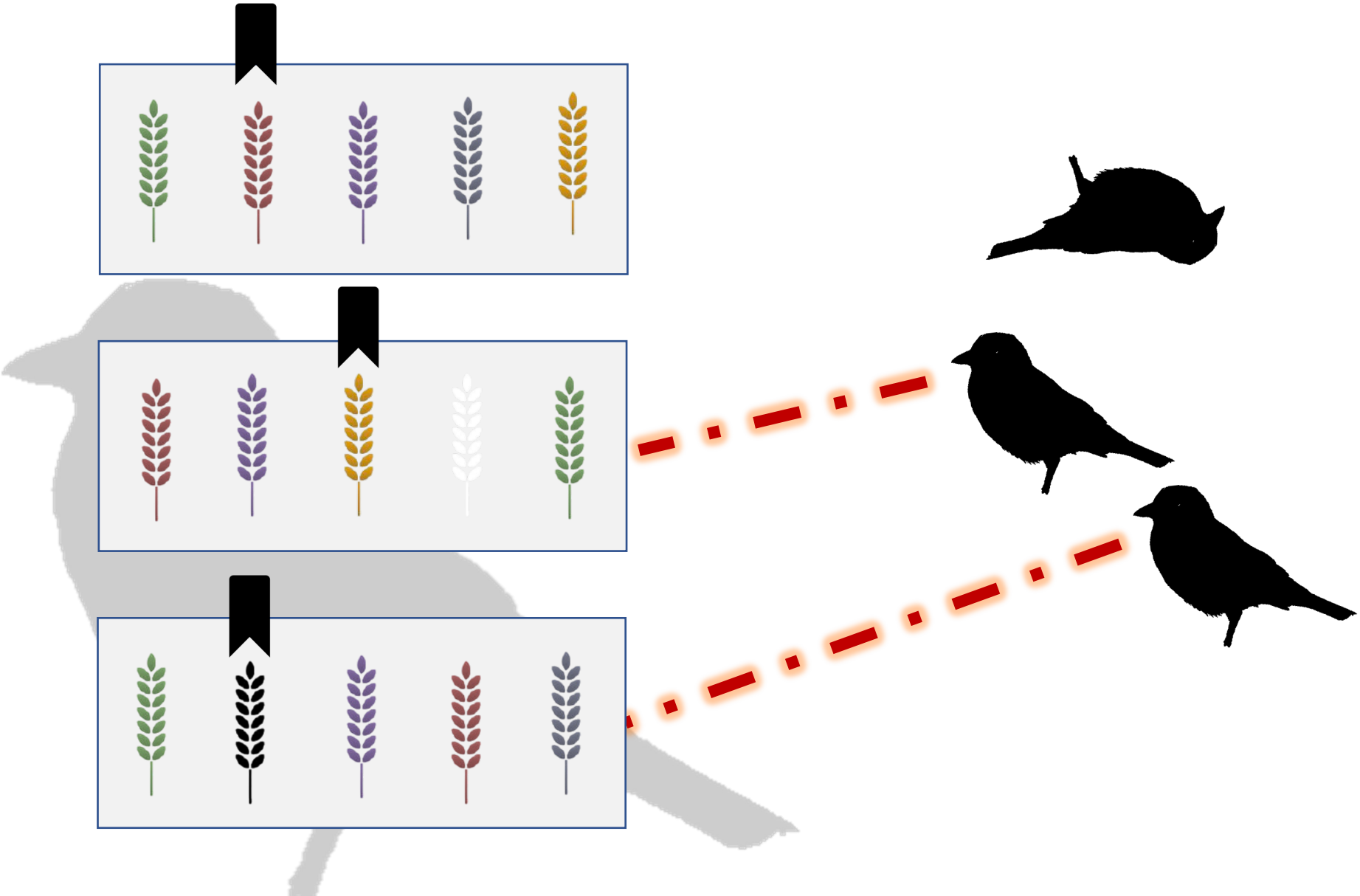
Zmena! 1 konzument odpadne



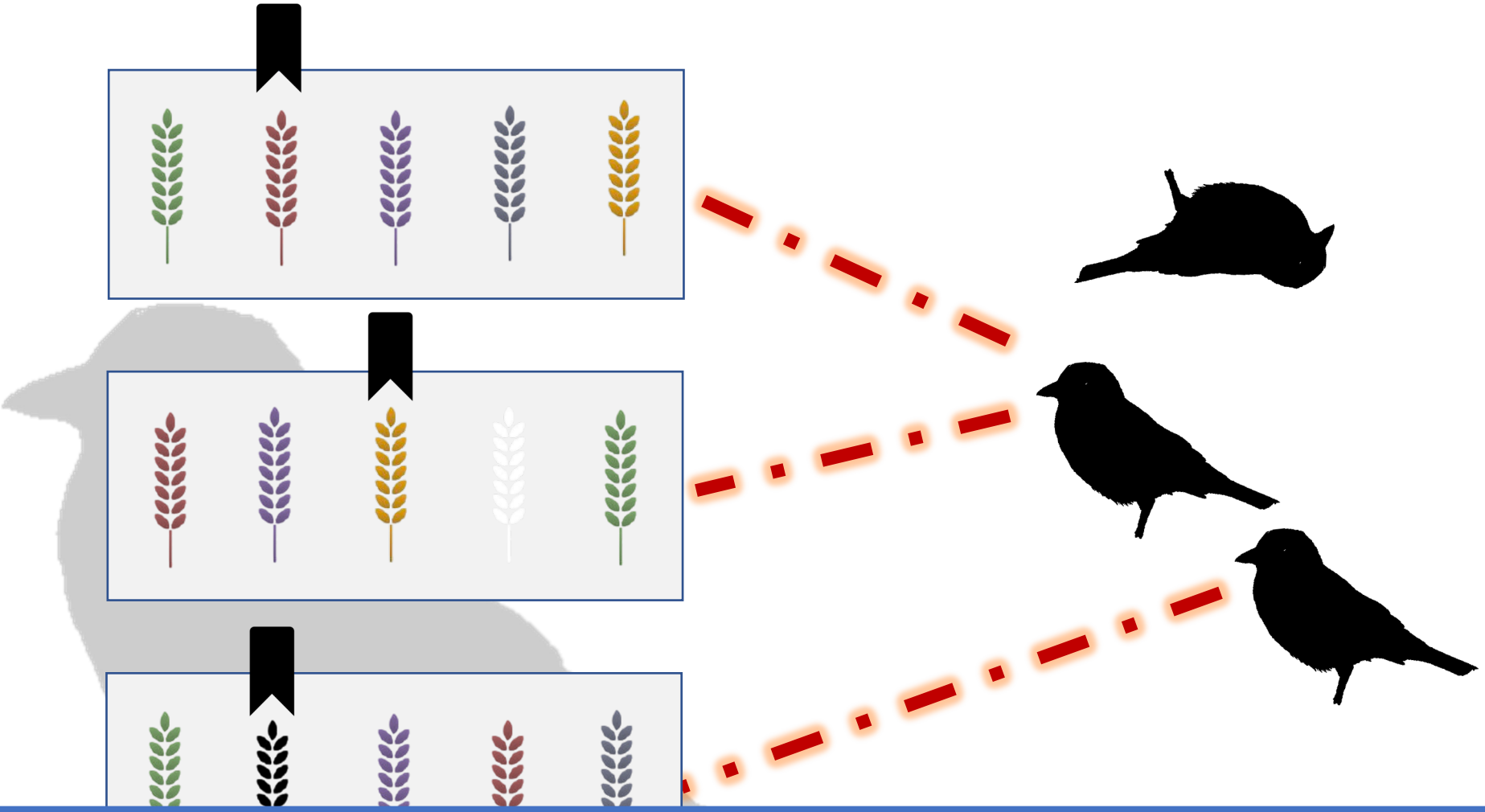
Zmena! 1 konzument odpadne



Rebalans: 3 partície na dvoch konzumentov



Rebalans: dodržíme pravidlá pre priradenie

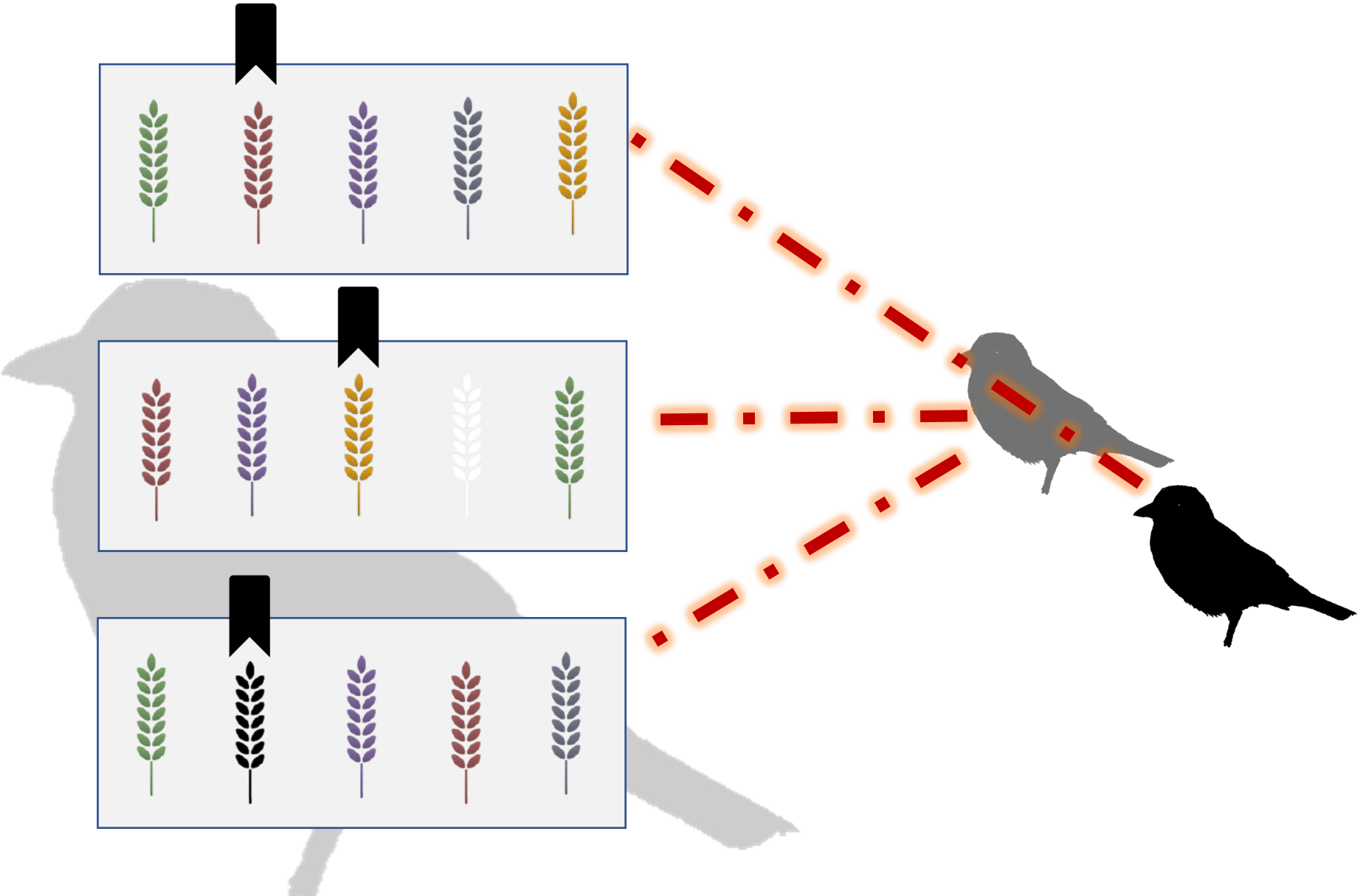


1 konzument získal 2 partície

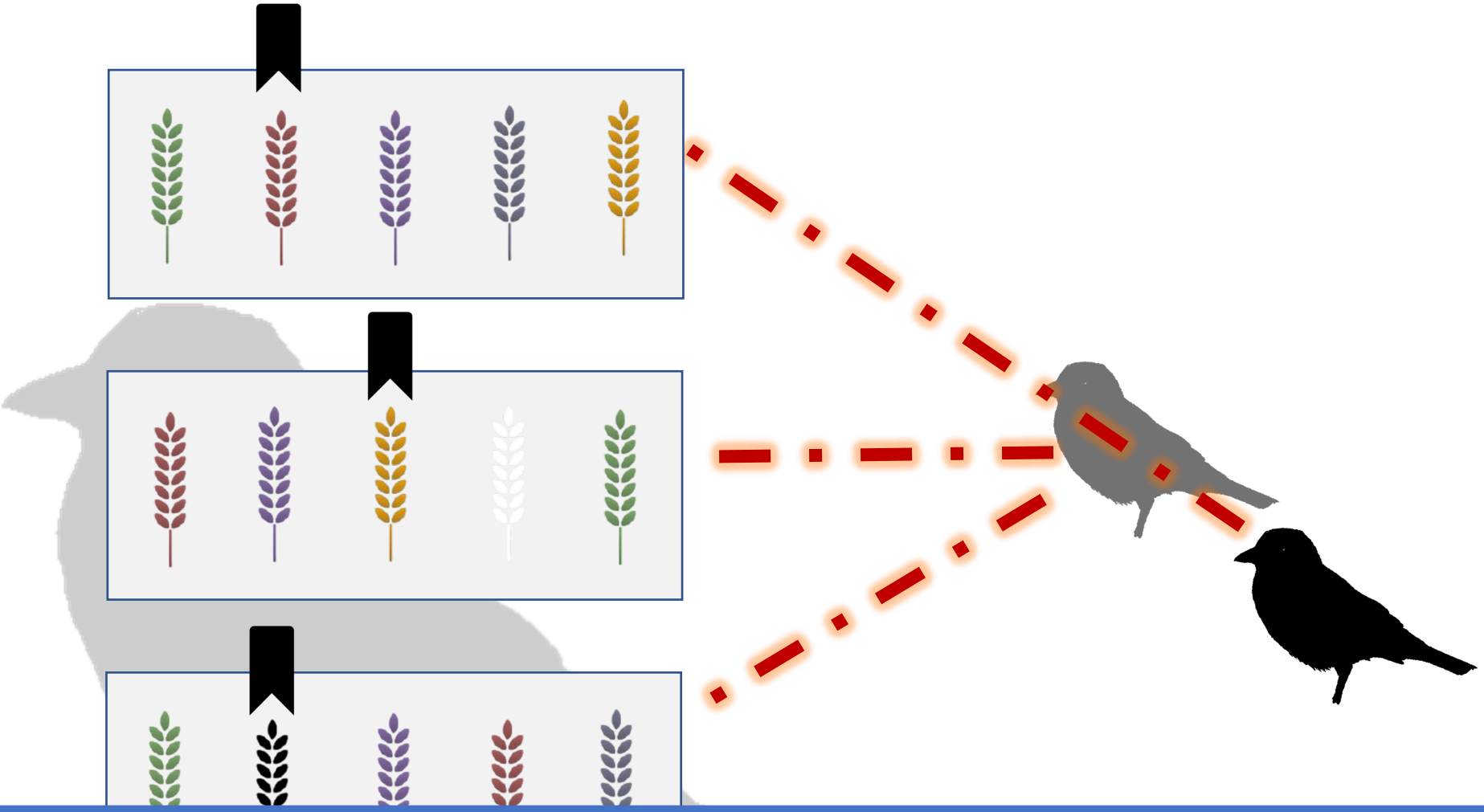
Rebalans

- priradenie medzi konzumentami a partíciami sa môže úplne premiešať
- vďaka komitnutým ofsetom to nie je problém!

Rebalans: premiešanie partícií



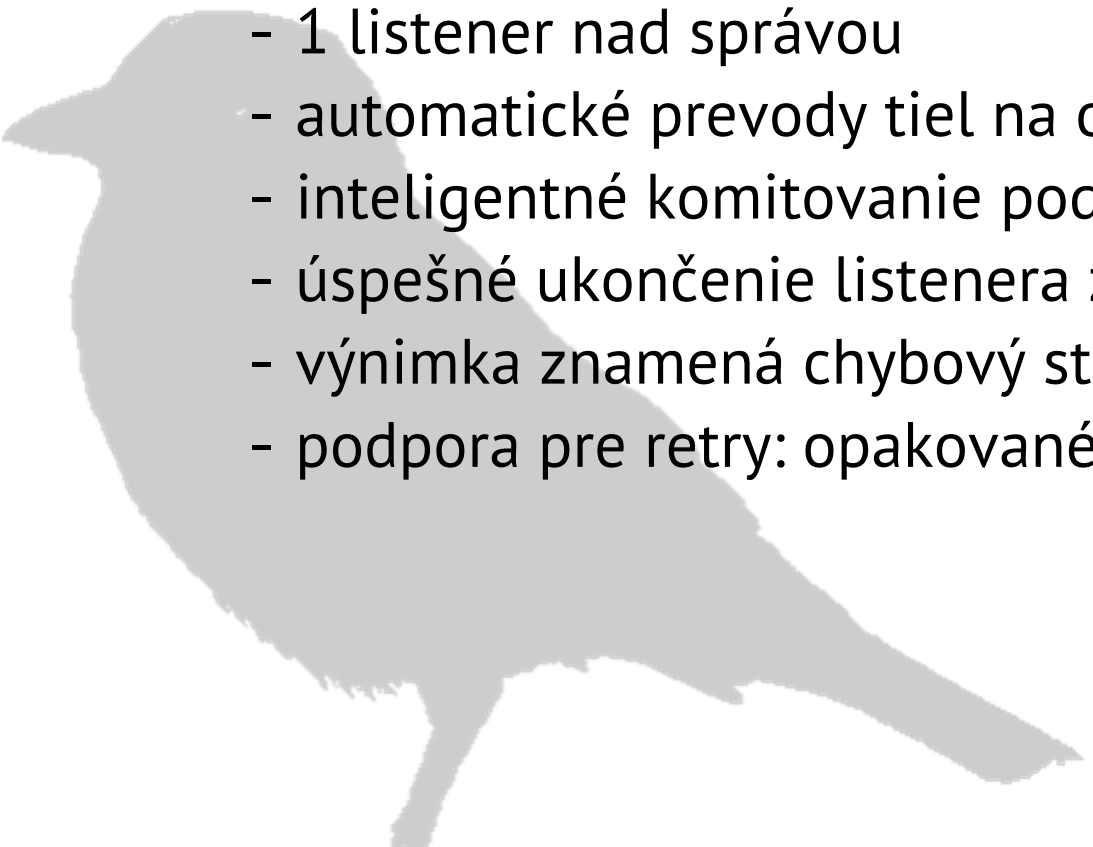
Rebalans: premiešanie partícií



Vďaka komitnutým offsetom člen grupy pokračuje po inom členovi grupy

Stratégie knižníc

- knižnice ponúkajú rozumné automatické prístupy
- Spring Boot + Kafka:
  - 1 listener nad správou
  - automatické prevody tiel na objekty
  - inteligentné komitovanie podľa viacerých kritérií
  - úspešné ukončenie listenera znamená komit ofsetu
  - výnimka znamená chybový stav
  - podpora pre retry: opakované doručenie



Pod ~~kapotou~~ perím

Konzument – poslucháč

```
kafka.subscribe(”zrno”)

while (true) {
      records = kafka.poll()
      for (r : records) {
            handle(r)
      }
}
```

„Kafka?! Chcem sledovať topic“

„Haló, Kafka?! Prosím si záznamy“

poll()

- požiada Kafku o *nejaké* záznamy
- získa dávku (**batch**) záznamov
- indikuje *heartbeat*: konzument je nažive

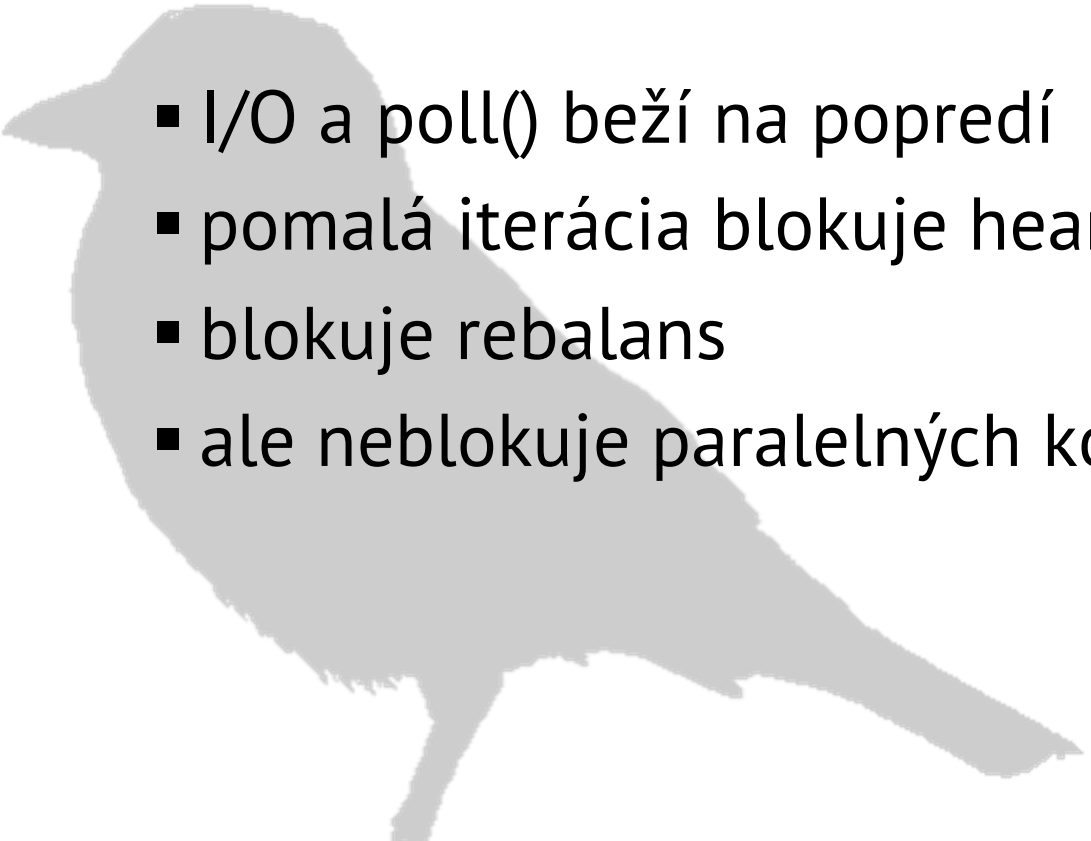
- obvykle zavolá biznisové metódy pre vybavenie záznamov

poll() a spracovanie

- nad dávkou sa volajú biznisové metódy pre jednotlivé záznamy
- napr. *listener* pre každý záznam

Java

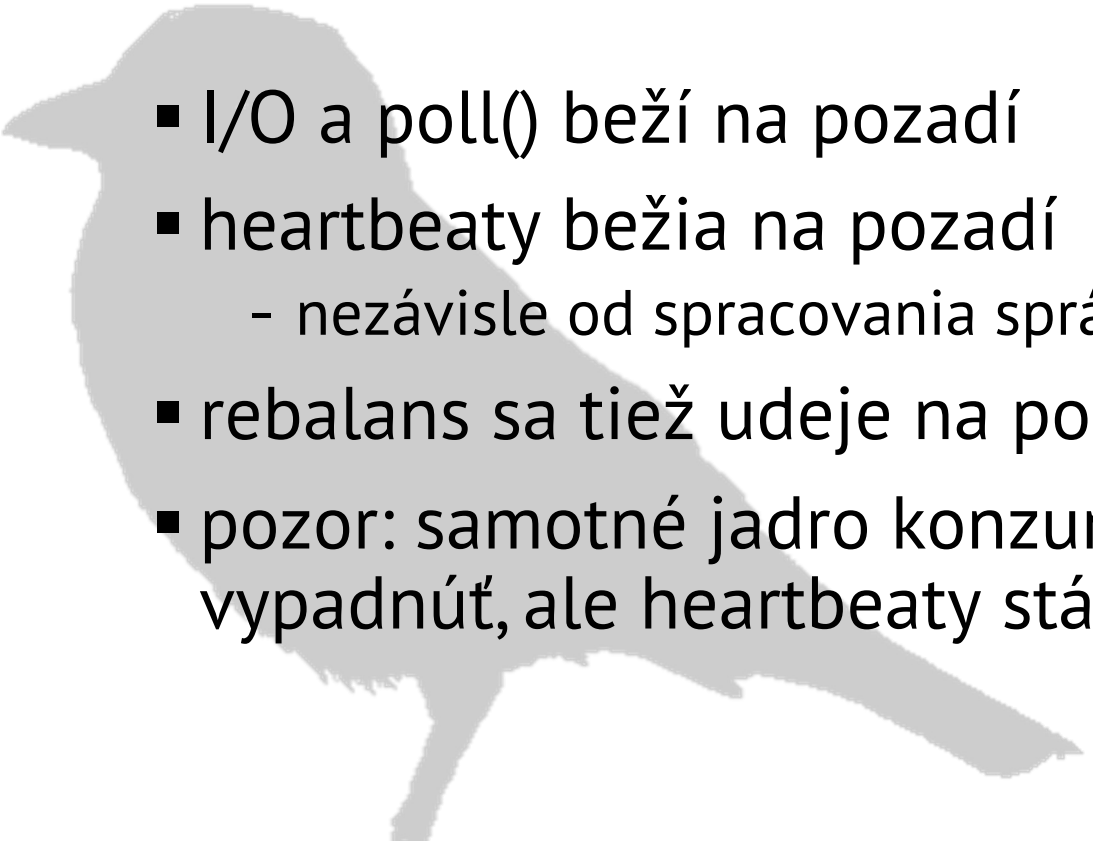
vybavenie záznamov musí zbehnúť rýchlo!



- I/O a poll() beží na popredí
- pomalá iterácia blokuje heartbeat
- blokuje rebalans
- ale neblokuje paralelných konzumentov

Go, C#, C/C++, Python

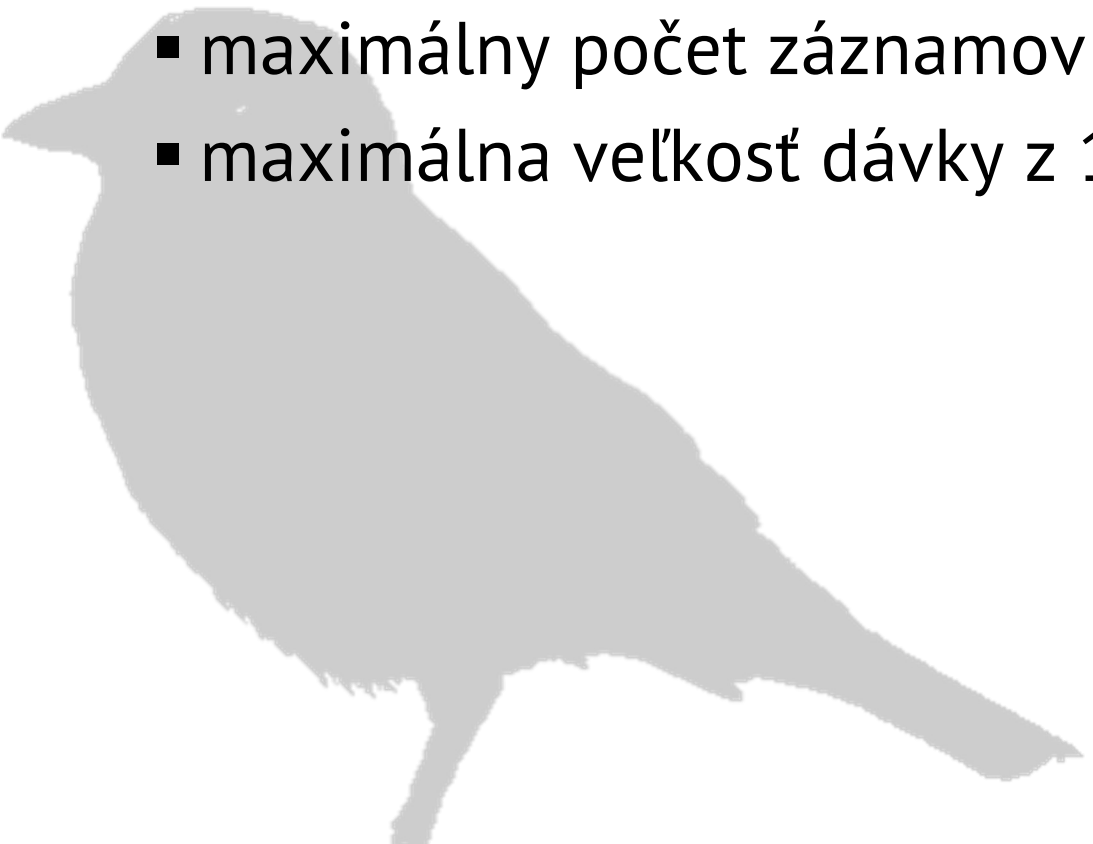
poll() beží na pozadí



- I/O a poll() beží na pozadí
- heartbeaty bežia na pozadí
  - nezávisle od spracovania správ
- rebalans sa tiež udeje na pozadí
- pozor: samotné jadro konzumenta môže vypadnúť, ale heartbeaty stále chodia

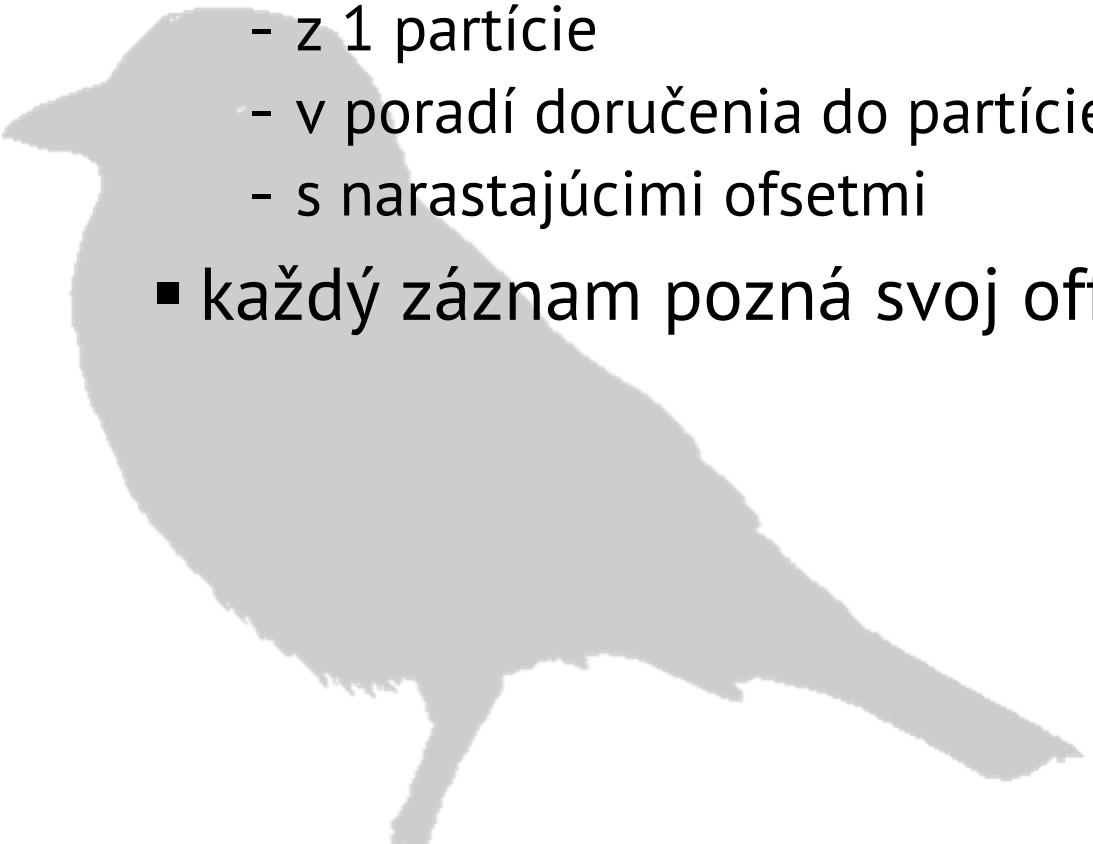
konzument a poll() je konfigurovateľný

- ako dlho čakať na dávku
- minimálna veľkosť dávky v bajtoch
- maximálny počet záznamov v dávke
- maximálna veľkosť dávky z 1 partície



poll() a dávka

- dávka je sada záznamov s kľúčmi a hodnotami
- obvykle
  - z 1 partície
  - v poradí doručenia do partície
  - s narastajúcimi ofsetmi
- každý záznam pozná svoj offset z partície topicu

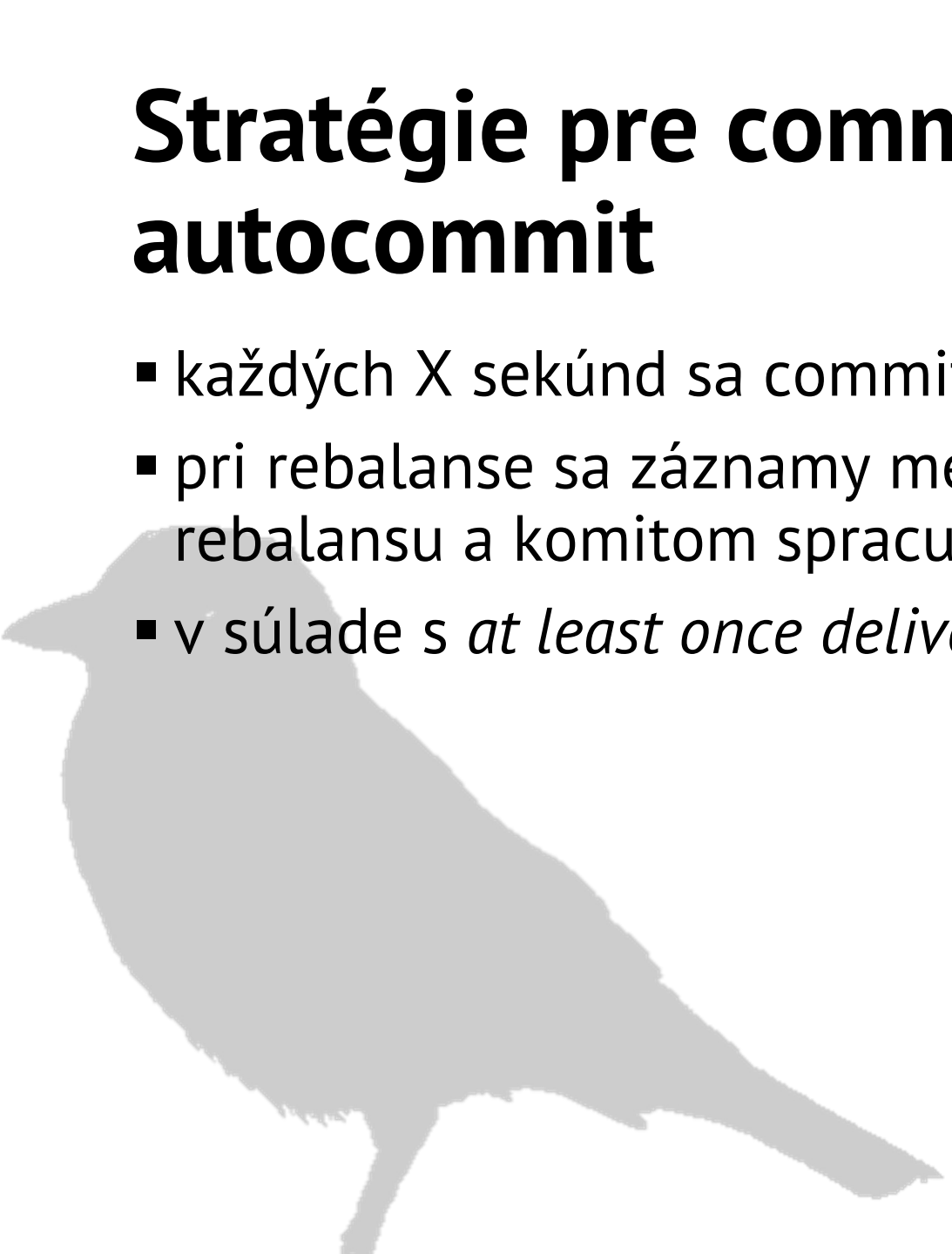


poll() a heartbeat

- konzument musí reagovať, inak ho Kafka vyradí z grupy
- konfigurácia:
  - ako často má konzument posielať *heartbeat* Kafke
  - ako dlho vydrží konzument bez kontaktu s Kafkou

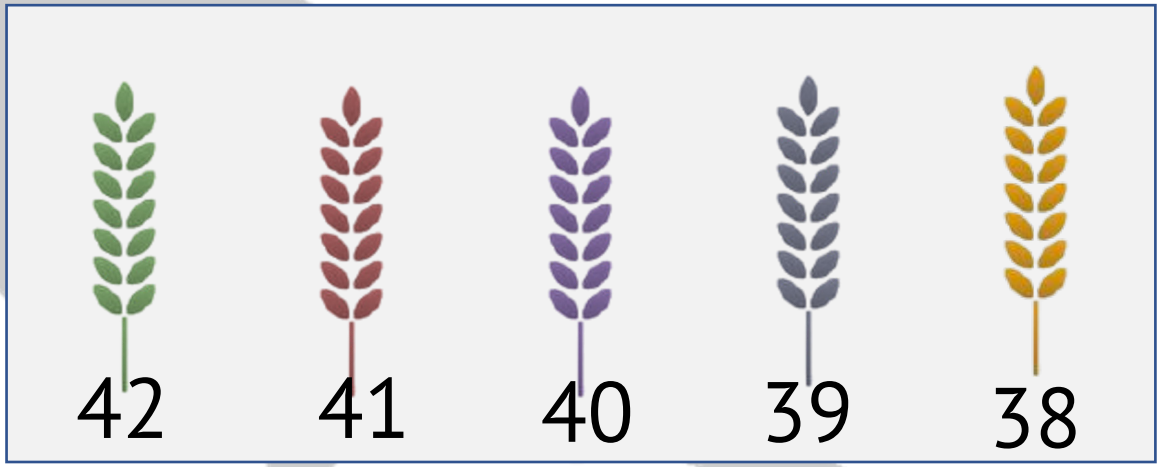
Stratégie pre commit: autocommit

- každých X sekúnd sa commitne
- pri rebalanse sa záznamy medzi momentom rebalansu a komitom spracujú viackrát
- v súlade s *at least once delivery*



Konzument

- nech je autocommit po 5 sekundách
- nech posledný komitnutý ofset v partícii je 37
- nech poll() vráti 5 záznamov: najväčší offset v dávke je 42



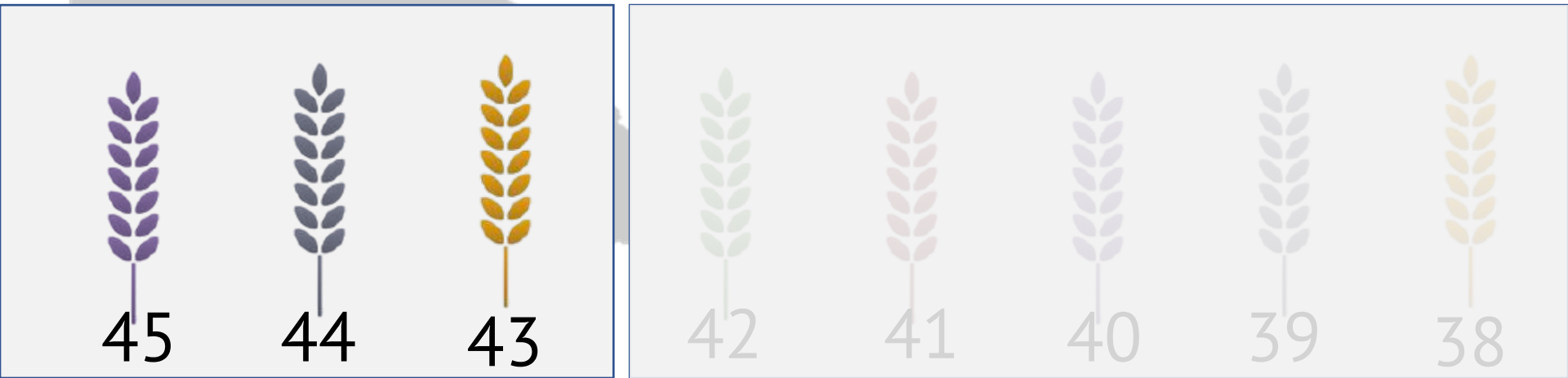
prvý poll()

Rýchly konzument

- spracujme dávku za 4 sekundy
- zavolajme ďalší **poll()**

druhý poll()

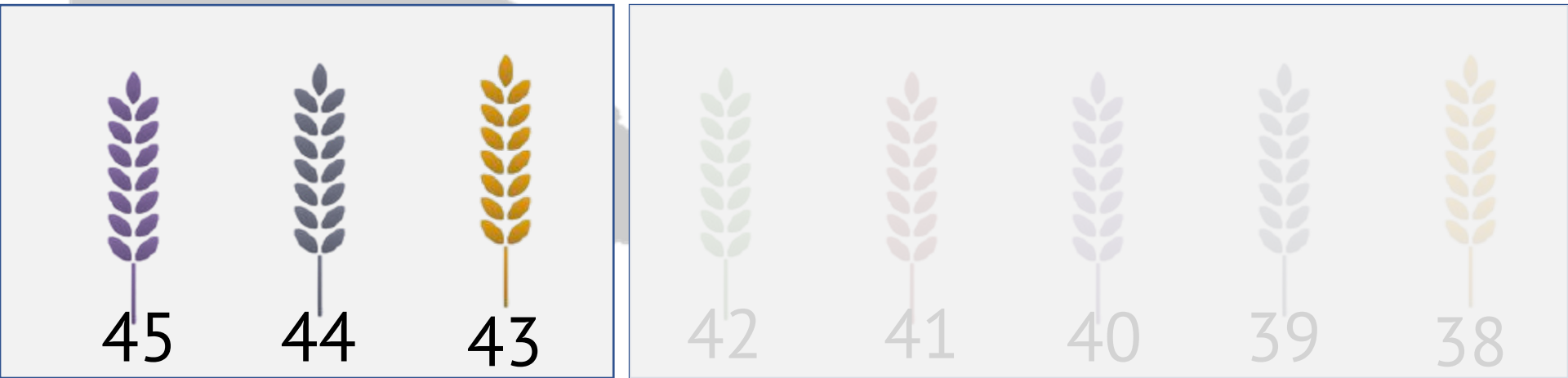
- keďže interval neuplynul, nekomitujeme
  - ofset je stále na 37
- dostaneme ďalšiu dávku: napr. 3 správy



Rýchly konzument

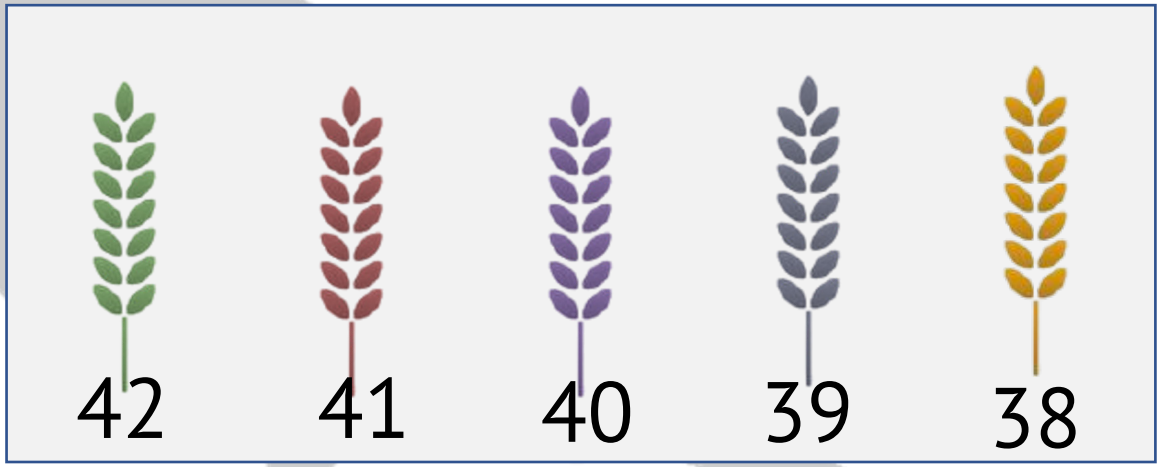
- spracujme dávku za 4 sekundy
- zavolajme ďalší **poll()**
- keďže interval už uplynul (4 + 4 sekundy > 5), komitujeme offset 45

tretí poll()



Pomalý konzument

- nech je autocommit po 5 sekundách
- nech posledný ofset v partícii je 37
- nech poll() vráti 5 správ: najväčší offset v dávke je 42

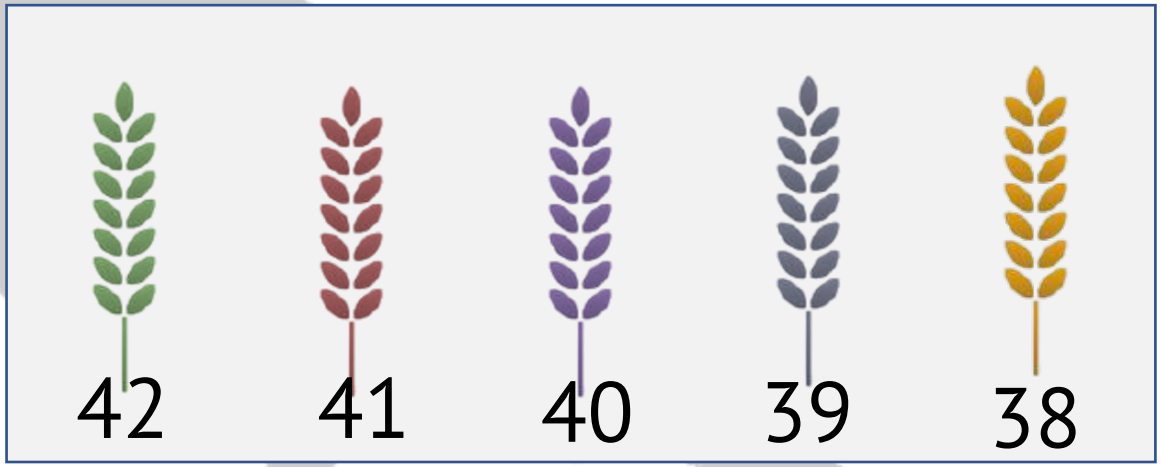


prvý poll()

Pomalý konzument

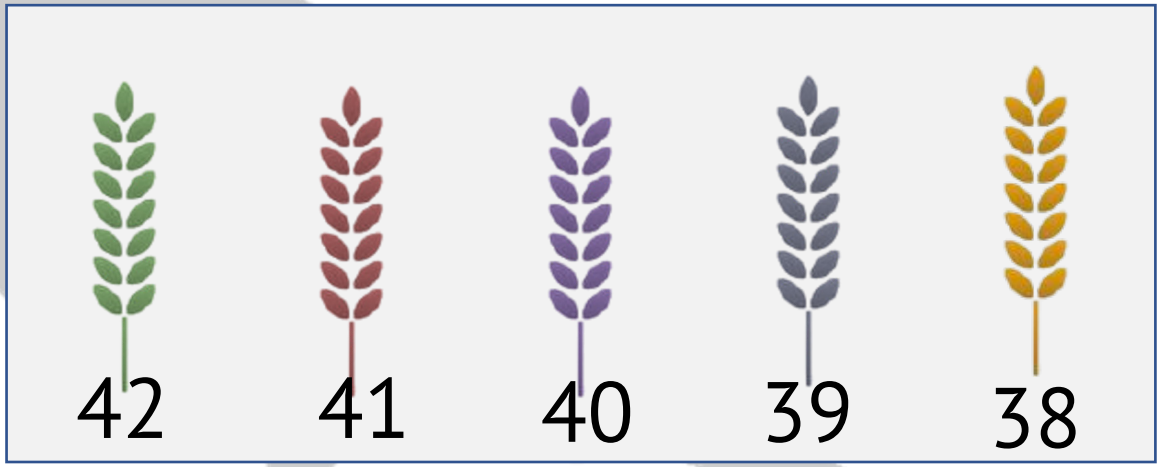
Pozor však na heartbeat!

- spracujme dávku za 6 sekúnd
- zavolajme ďalší **poll()**
- keďže interval uplynul, komitujeme offset 42
  - 6 s > limit 5s



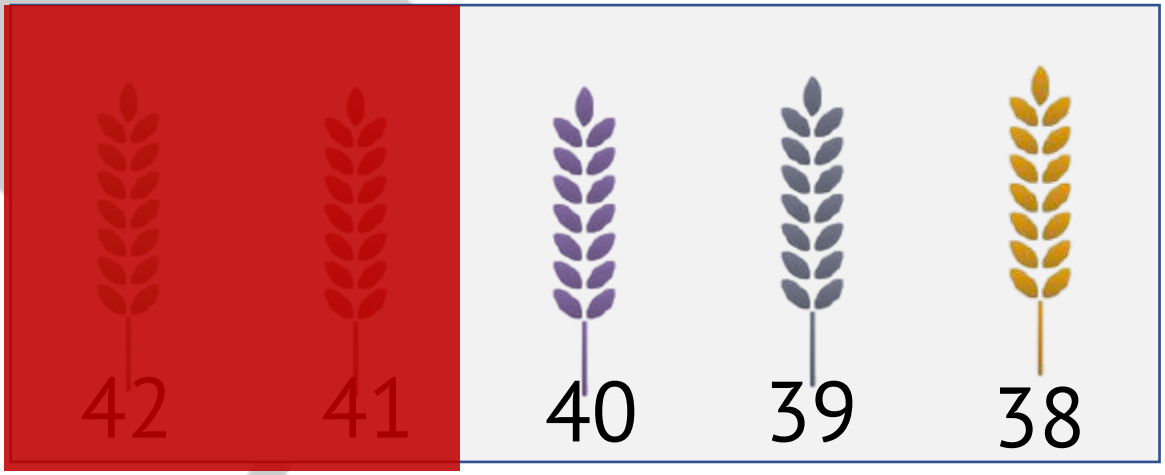
Konzument s chybou

- nech je autocommit po 5 sekundách
- nech posledný ofset v partícii je 37
- nech poll() vráti 5 správ: najväčší offset v dávke je 42



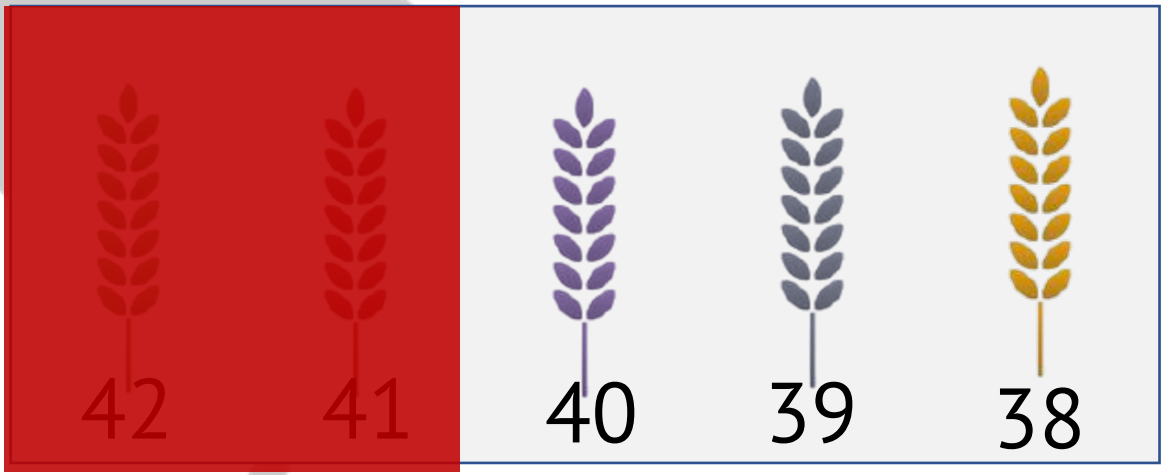
Konzument s chybou

- spracujeme prvé 3 záznamy (38, 39, 40)
- počas záznamu 40 nastane výnimka
- pozor na záznamy 41 a 42 – musíme ich spracovať, inak sa stratia!
- autokomit pri ďalšom poll() komitne ofset 42



Konzument s chybou

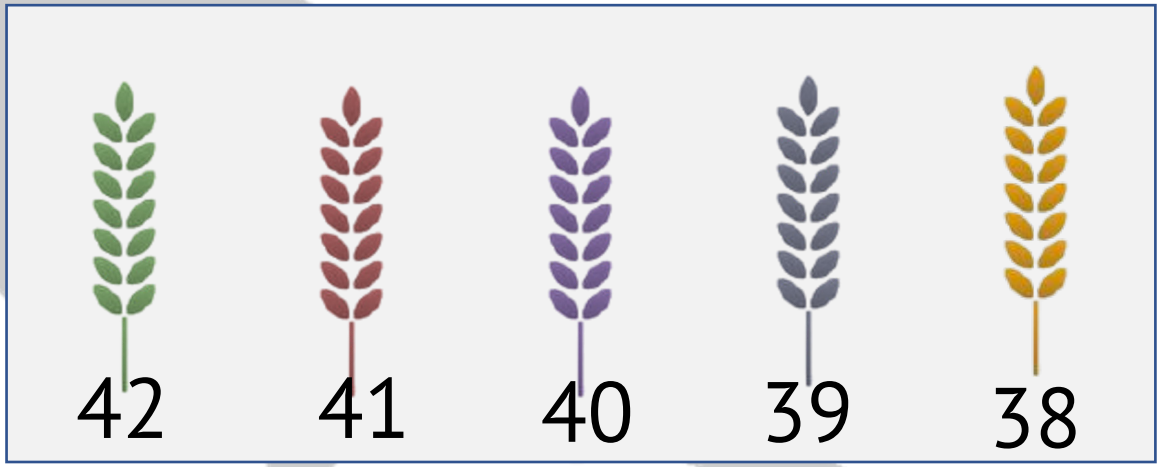
- spracujeme prvé 3 záznamy (38, 39, 40)
- počas záznamu 40 nastane výnimka
- pozor na záznamy 41 a 42 – musíme ich spracovať, inak sa stratia!
- autokomit pri ďalšom poll() komitne ofset 43



At most once delivery

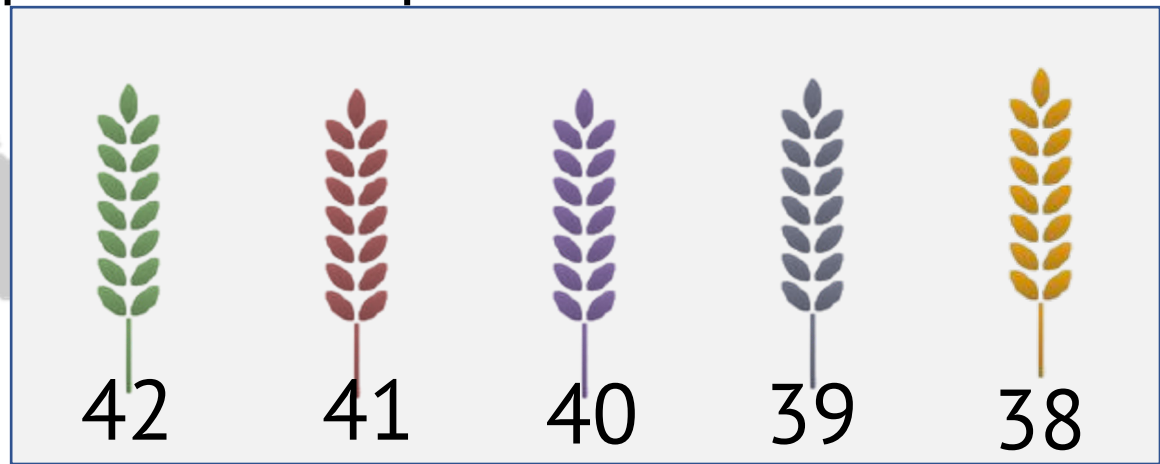
Konzument a rebalans

- nech je autocommit po 5 sekundách
- nech posledný komitnutý ofset v partícii je 37
- nech poll() vráti 5 správ: najväčší offset v dávke je 42



Rýchly konzument a rebalans

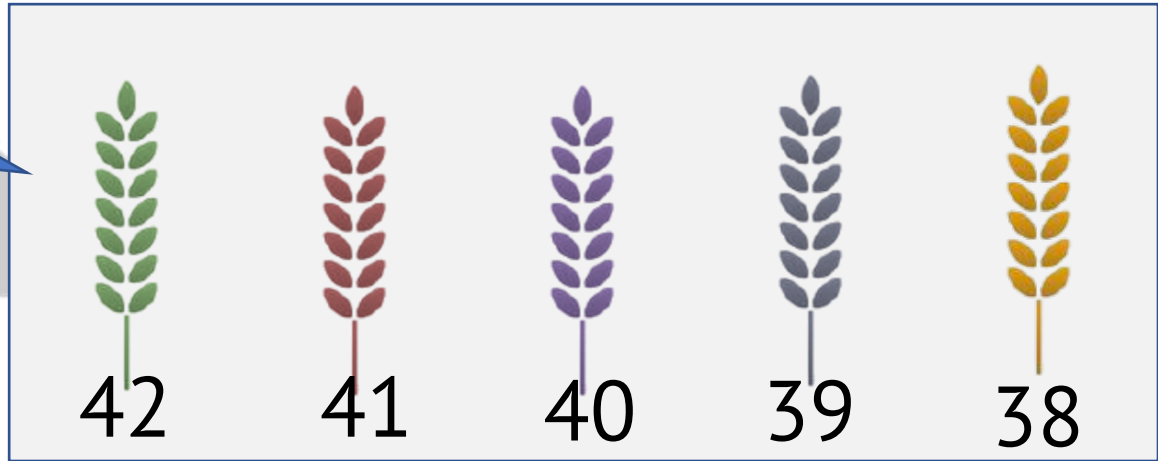
- spracujme dávku 5 záznamov za 4 sekundy
- zavolajme ďalší **poll()**
- keďže interval neuplynul, nekomitujeme
  - ofset je stále na 37
- predstavme si, že nastáva rebalans!
- konzumentovi sa pridelí nová partícia!



Rýchly konzument a rebalans

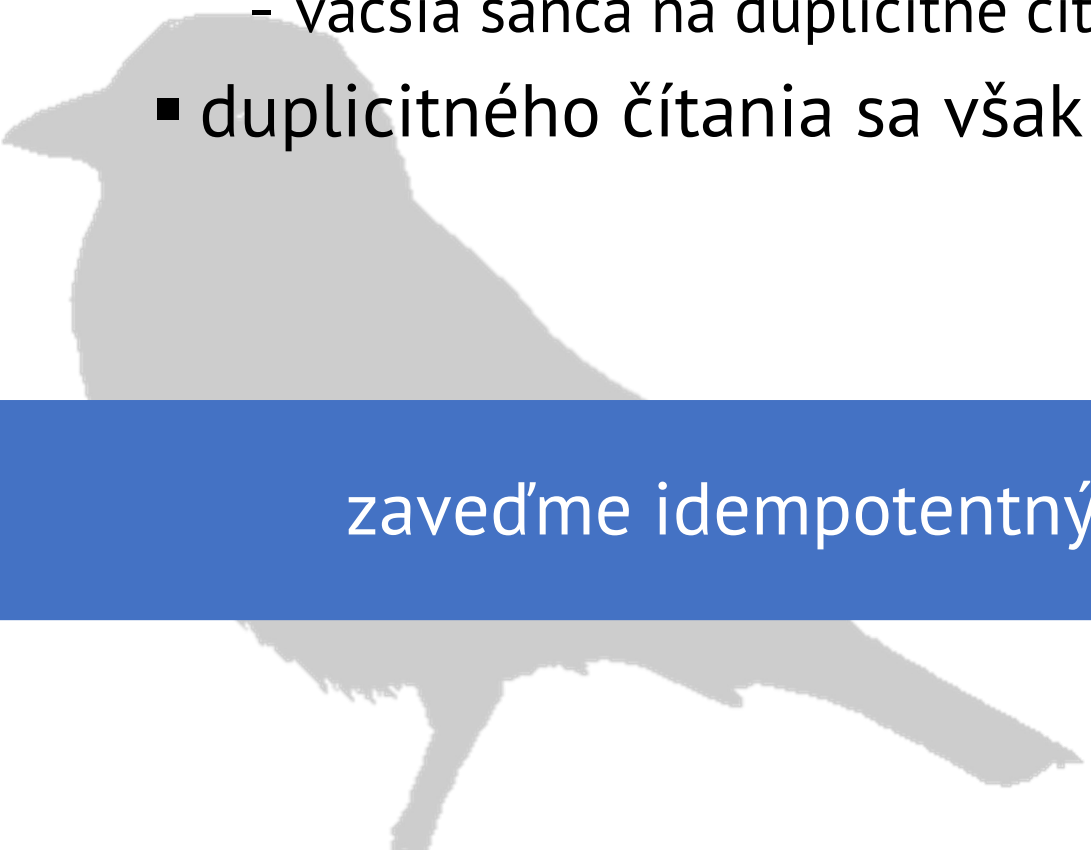
- po rebalanse dostane nejaký konzument partíciu
- posledný komitnutý ofset je **37**
- ale záznamy 38 až 42 sme už spracovali!
- nastáva **duplicitné čítanie**

At-least-once delivery



At least once delivery

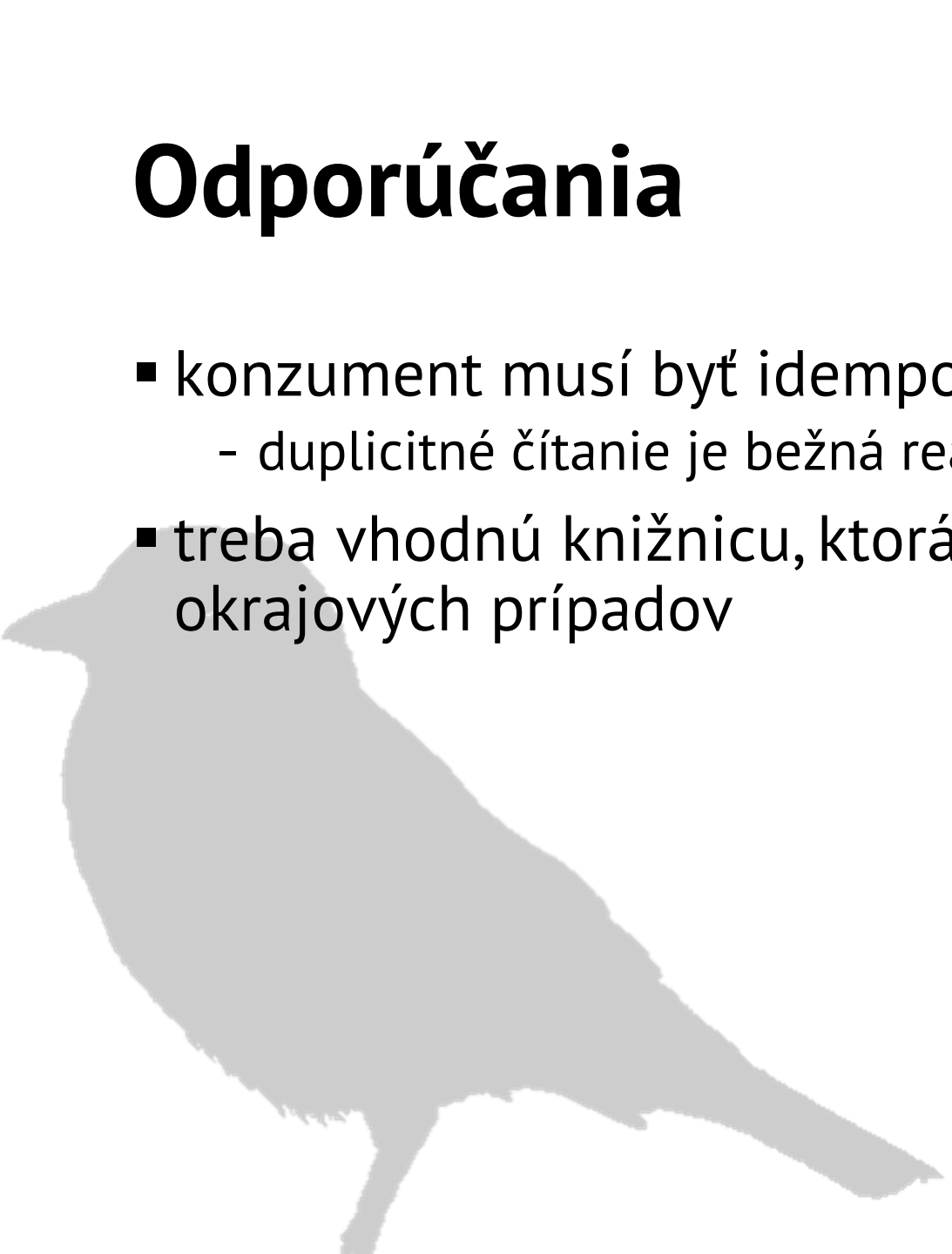
- dlhší interval na autokomit
  - väčší prietok
  - väčšia šanca na duplicitné čítanie
- duplicitného čítania sa však **nikdy** nezbavíme!



zaveďme idempotentných konzumentov

Odporúčania

- konzument musí byť idempotentný
  - duplicitné čítanie je bežná realita
- treba vhodnú knižnicu, ktorá nás odtieni od okrajových prípadov



Ručný komit

- prečítame dávku záznamov
- spracujeme *všetky* záznamy
- explicitne komitneme ofset z **poll()**

- ak nastane počas dávky rebalans, celá dávka bude at least once

Synchrónny ručný komit ofsetu

- konzument čaká, kým Kafka neodpovie s výsledkom komitu
  - pri nedostupnosti opakuje komit, kým neuspeje
- znižuje sa prietok
  - ak komitujeme zriedkavejšie, zvyšujeme šancu duplicitného čítania

```
consumer.commitSync()
```

Asynchrónny ručný komit ofsetu

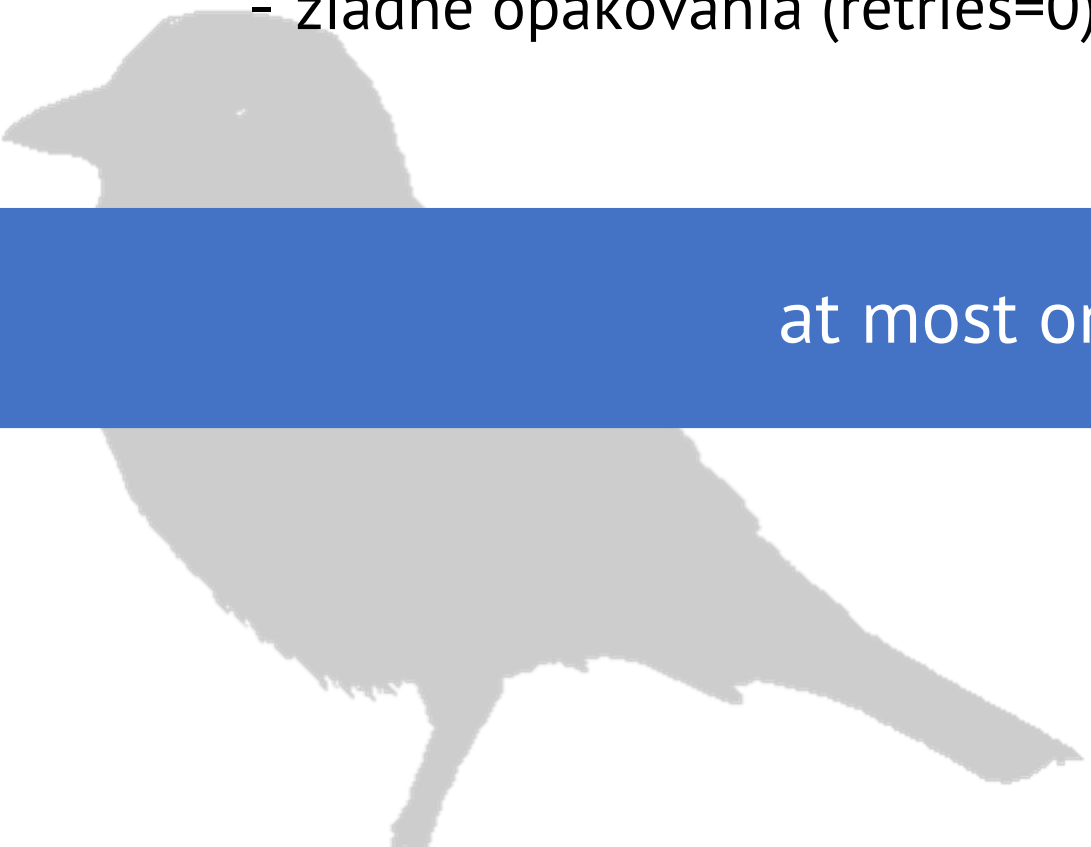
- konzument zahlási komit ofsetu a nečaká
- môžeme nechtiac komitnúť nižší ofset než je realita
  - prvá iterácia asynchrónne komitne 43, ktorý **nedorazí**
  - druhá iterácia komitne 46, ktorý **dorazí**
  - pri opakovaní komitu 43 si môžeme prepísať vyšší ofset
  - a spôsobíme duplicitné čítanie!

consumer.commitAsync()

Stratégie pre producentov

Prostý producent

- pošle správu cez **send()**, dúfa v najlepšie
  - žiadne ACKs (acks=0)
  - žiadne opakovania (retries=0)



at most once

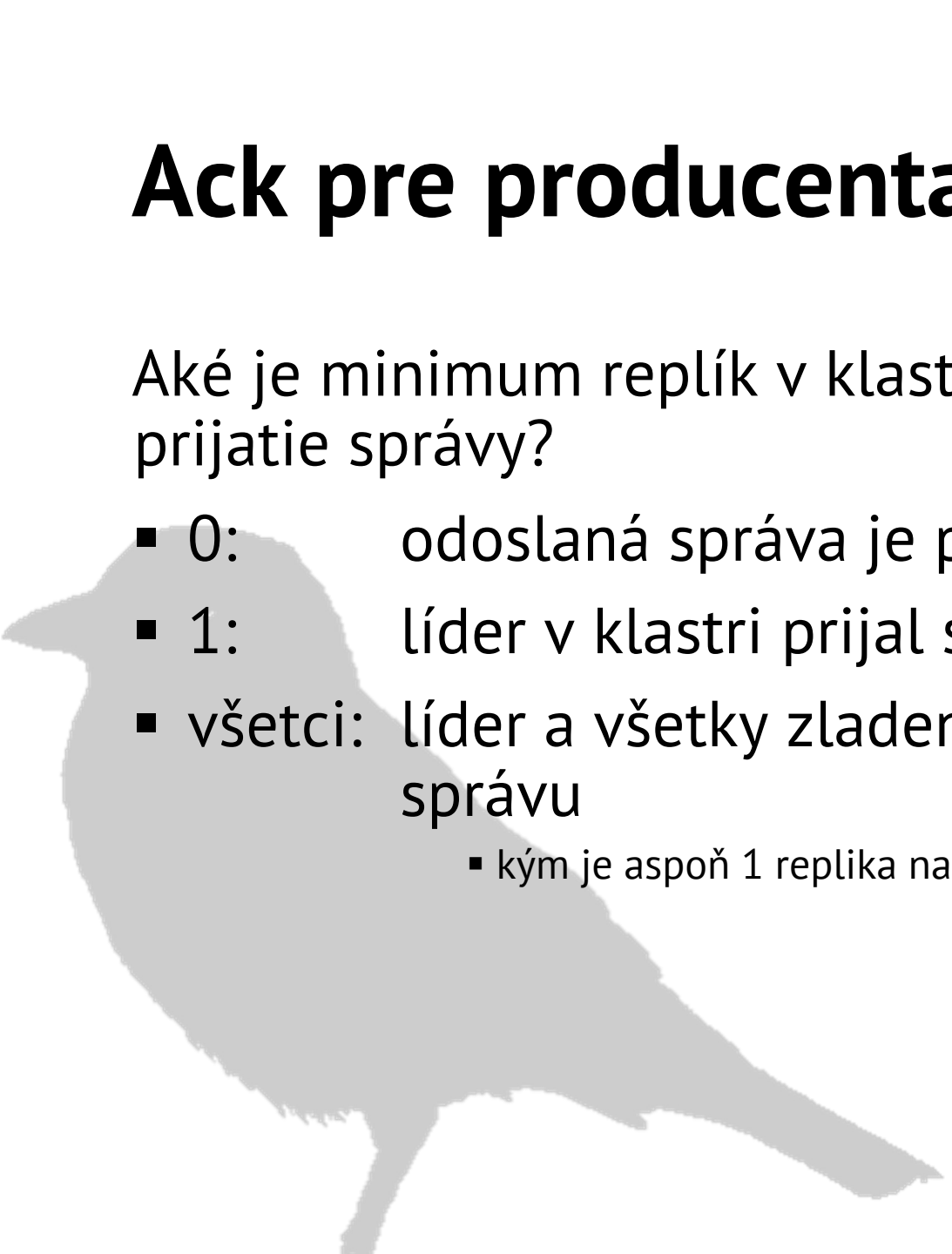
Prostý producent s ACK

- Kafka môže zaslať producentovi potvrdenku o doručení záznamu: **acknowledgement**
- Tri úrovne kompromisu **výkon** vs **spoľahlivosť doručenia**

Ack pre producenta

Aké je minimum replík v klastrí, ktoré potvrdia prijatie správy?

- 0: odoslaná správa je prijatá správa
- 1: líder v klastri prijal správu
- všetci: líder a všetky zladené repliky prijali správu
  - kým je aspoň 1 replika nažive, záznam sa nestratí



Prostý producent s ACK a retry

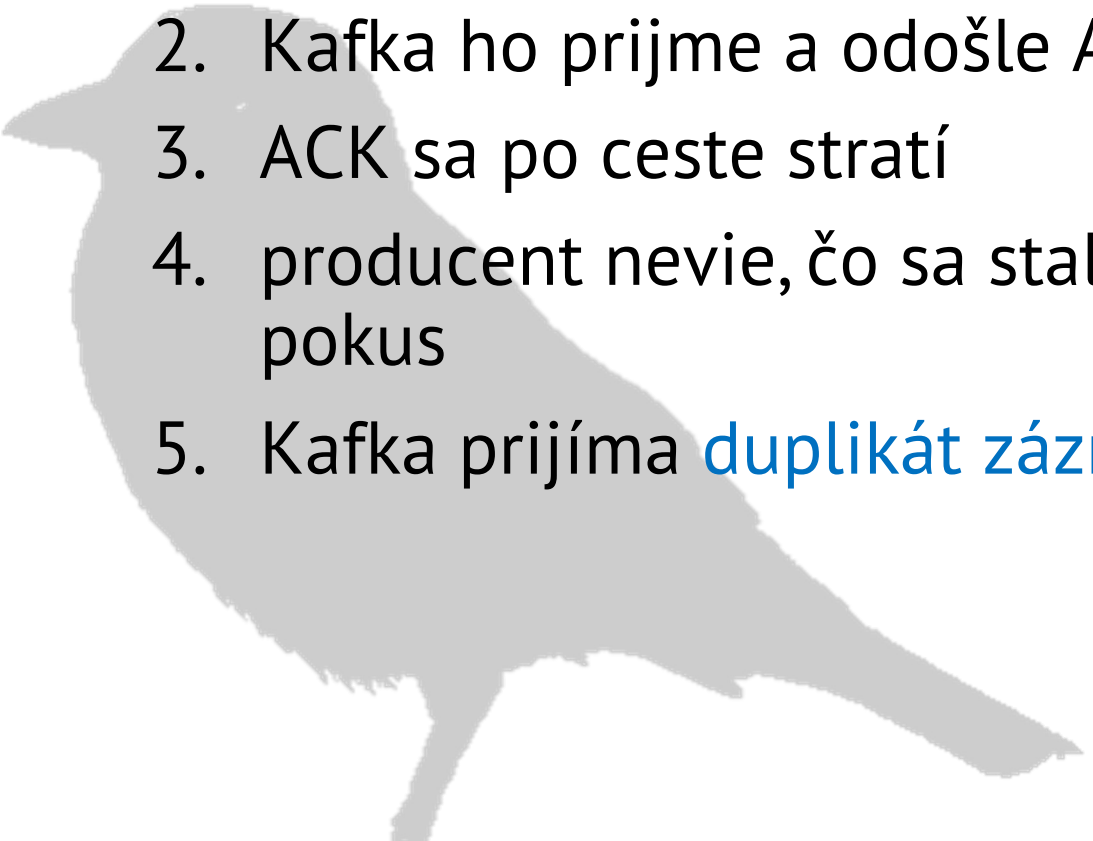
- pošle správu cez **send()**
- bude čakať na ACKs
- kde to môže zlyhať?
  - ak sa správu nepodarí odoslať cez send()
  - ak sa správa odošle, ale ACK zo servera nedorazí

opakujme cez **retry**

Prostý producent s ACK a retry: Duplikáty

Scenár:

1. producent odošle záznam
2. Kafka ho prijme a odošle ACK
3. ACK sa po ceste stratí
4. producent nevie, čo sa stalo s ACK, opakuje pokus
5. Kafka prijíma duplikát záznamu!



Idempotentný producent

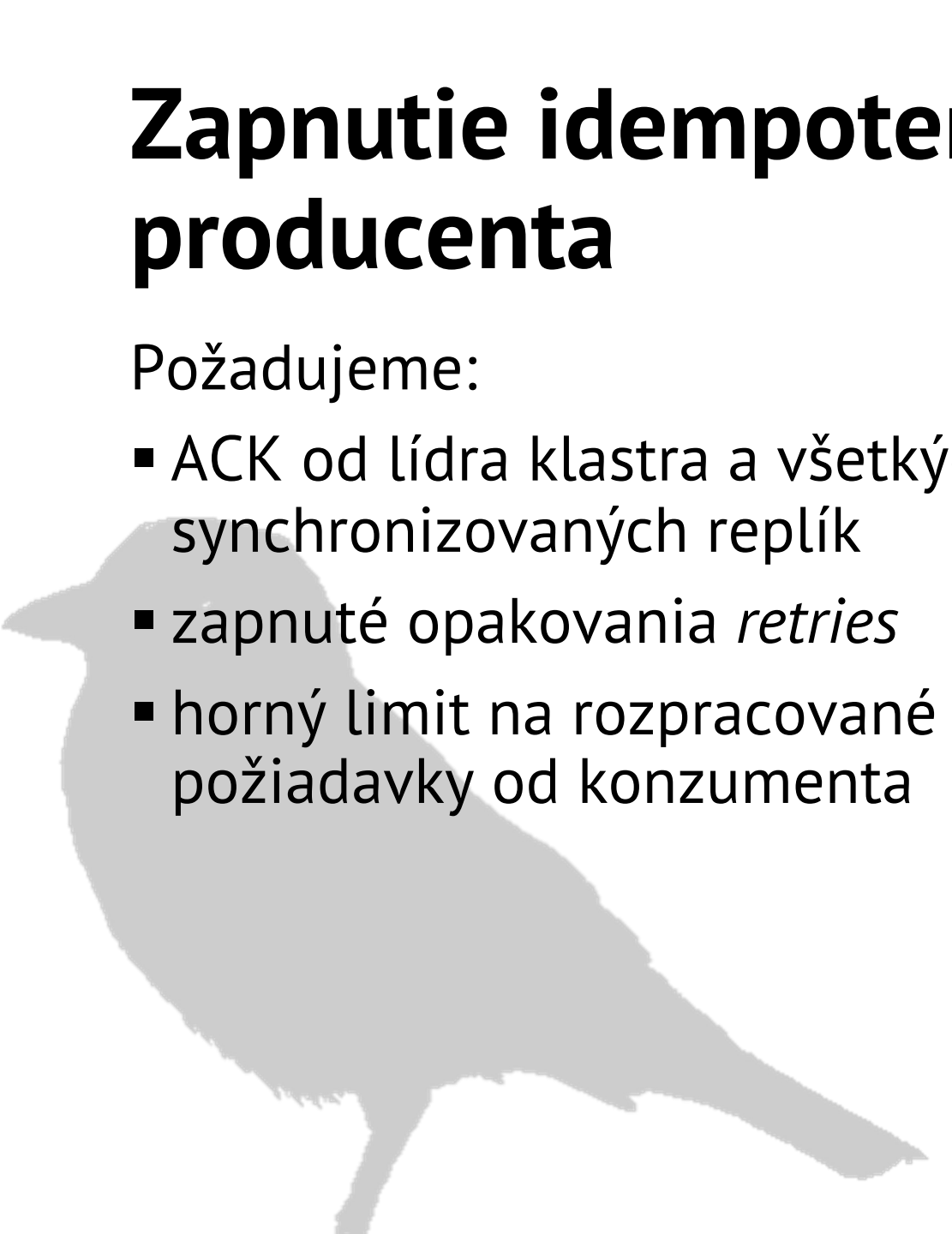
Garancia, že do prúdu sa zapíše presne 1 kópia správy

žiadne duplikáty, ani pri zlyhaniach a opakovaniach

Zapnutie idempotentného producenta

Požadujeme:

- ACK od lídra klastra a všetkých synchronizovaných replík
- zapnuté opakovania *retries*
- horný limit na rozpracované nepotvrdené požiadavky od konzumenta



Konfigurácia
enable.idempotence=true

Vyžadujú sa:

acks: all

retries: >0

max.in.flight.requests.per.connection <=5

Kafka 3.x (sept. 2021) automaticky zapína idempotentných klientov

Garancie idempotentných producentov

Správy odoslané producentom

- v danom poradí,
- doručené do spoločnej partície,

budú čítané konzumentom v danom poradí

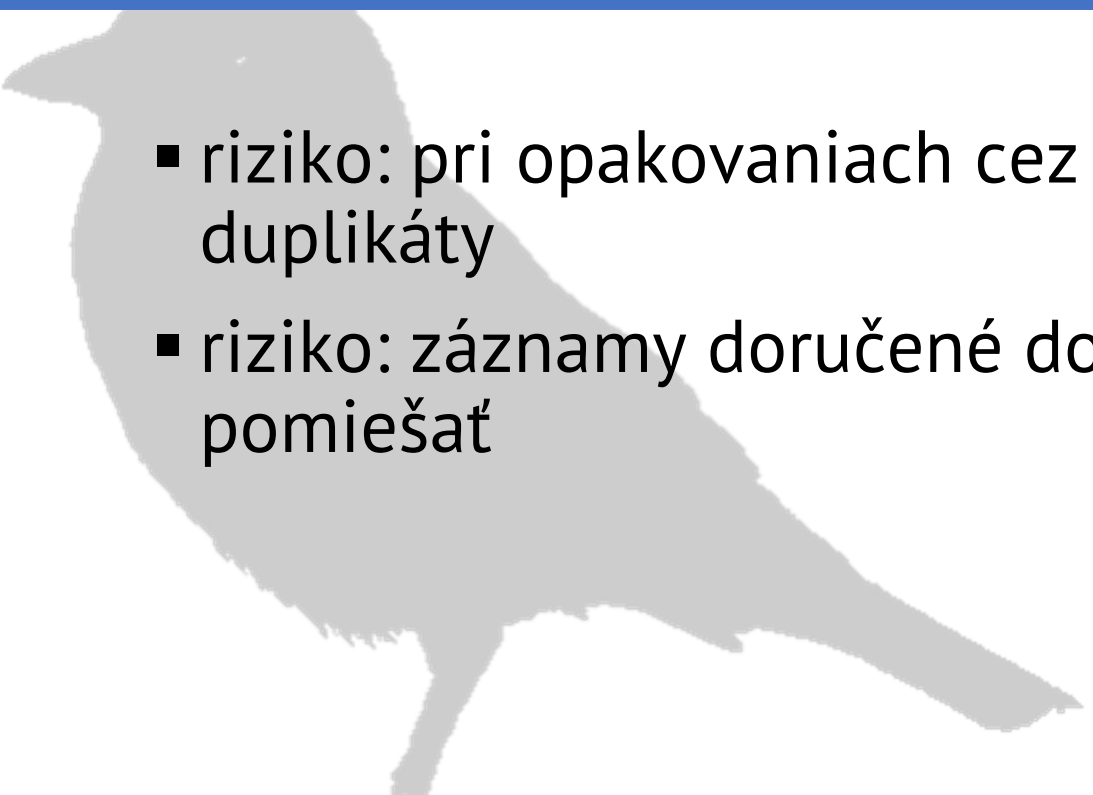
- v partícii nevzniknú duplikáty

poradie záznamov v partícii sa zaručuje

poradie záznamov v topicu nie je garantované

Zvyšovanie výkonu pre producentov

Vypneme idempotentnosť na producentovi



- riziko: pri opakovaniach cez retry vzniknú duplikáty
- riziko: záznamy doručené do partície sa môžu pomiešať

Producenti bez idempotencie a miešanie správ v partícii

- ak je **enable.idempotence** vypnuté
- a **retries** zapnuté [default]
- a **max.in.flight.requests.per.connection** >1 [default]

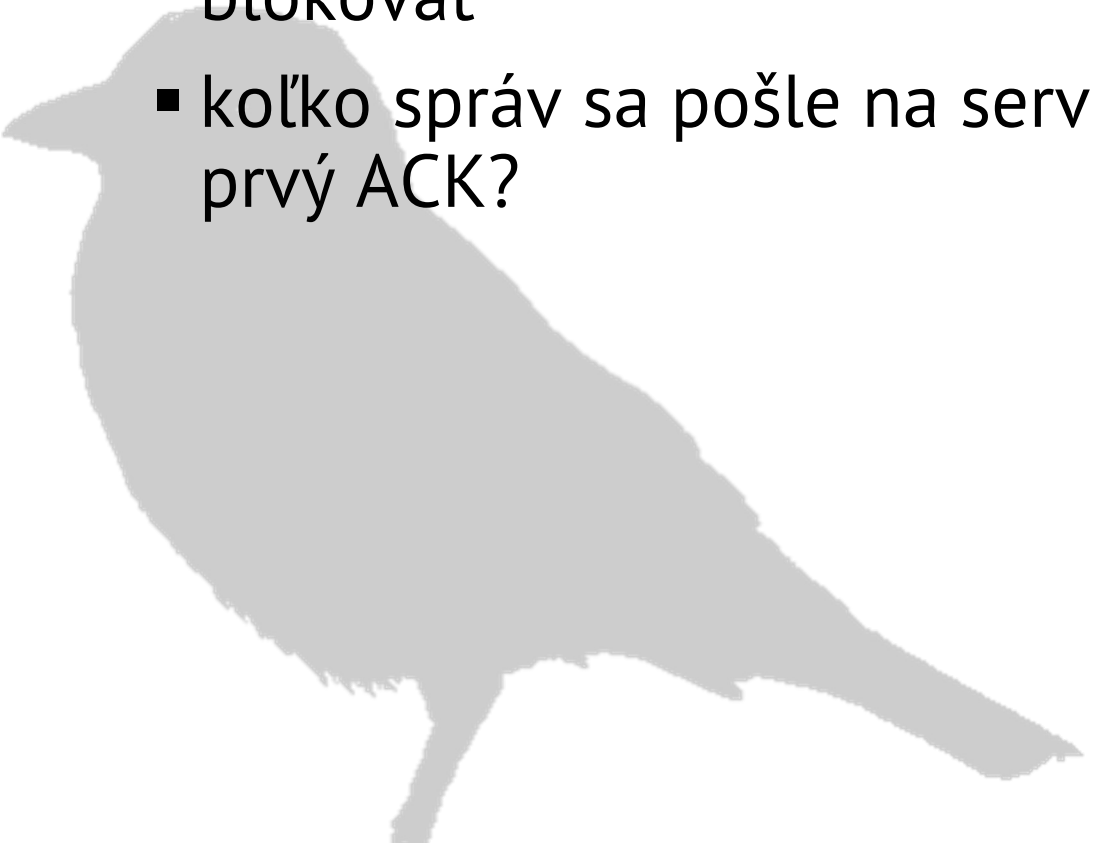
Ak pošleme 2 dávky do spoločnej partície a producent spustí 2 súbežné požiadavky a

- dávka #1 zlyhá,
- dávka #2 uspeje
- a zopakuje sa #1

Záznamy z #2 sa možno zapíšu pred záznamami z #1

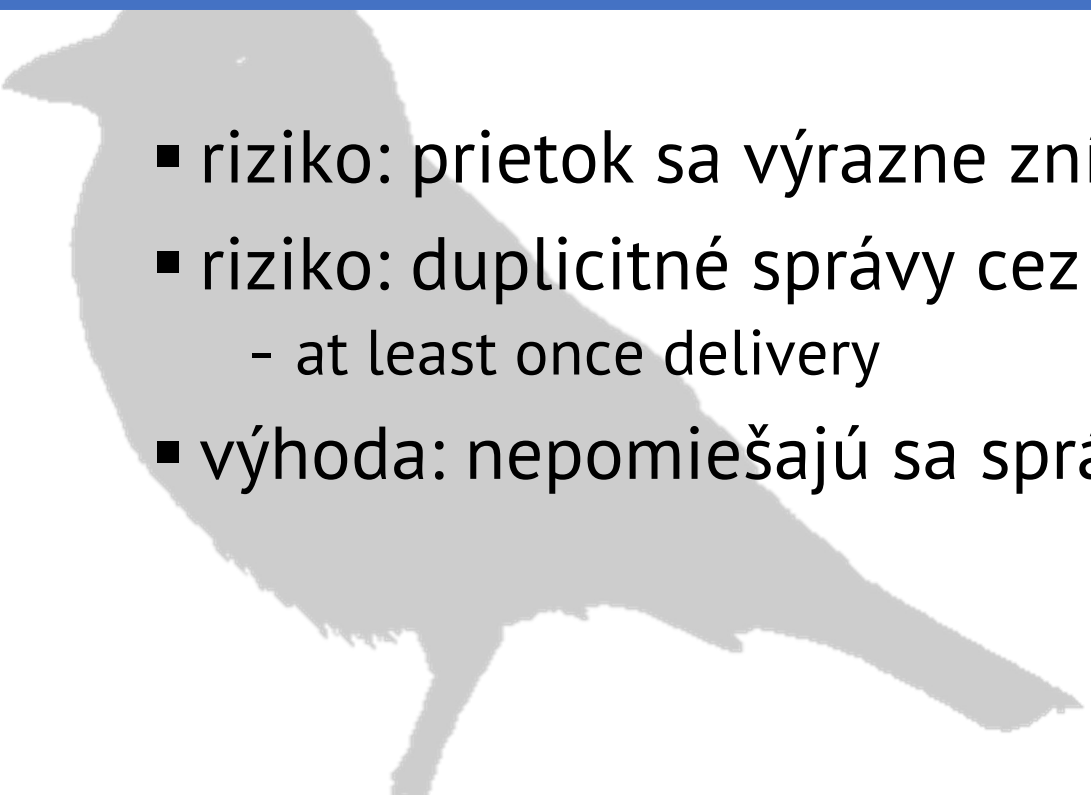
max.in.flight.requests.per.connection (5)

- maximálny počet neACKnutých požiadaviek, ktoré klient pošle v 1 pripojení, kým nezačne blokovať
- koľko správ sa pošle na server bez čakania na prvý ACK?



Zvyšovanie výkonu pre producentov

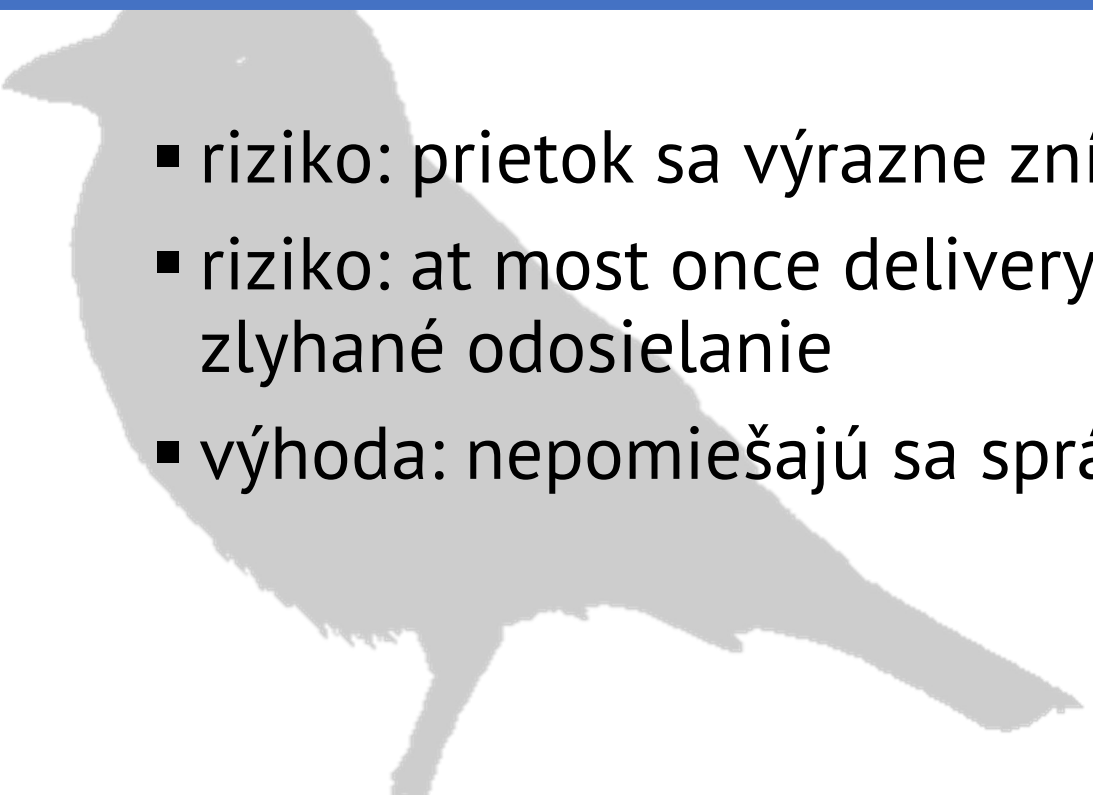
Vypneme idempotentnosť na producentovi
Znížime in-flight na 1



- riziko: prietok sa výrazne zníži
- riziko: duplicitné správy cez opakované retries
  - at least once delivery
- výhoda: nepomiešajú sa správy

Zvyšovanie výkonu pre producentov

Vypneme idempotentnosť na producentovi
Znížime in-flight na 1
Vypneme opakovania retries



- riziko: prietok sa výrazne zníži
- riziko: at most once delivery, lebo neopakujeme zlyhané odosielanie
- výhoda: nepomiešajú sa správy

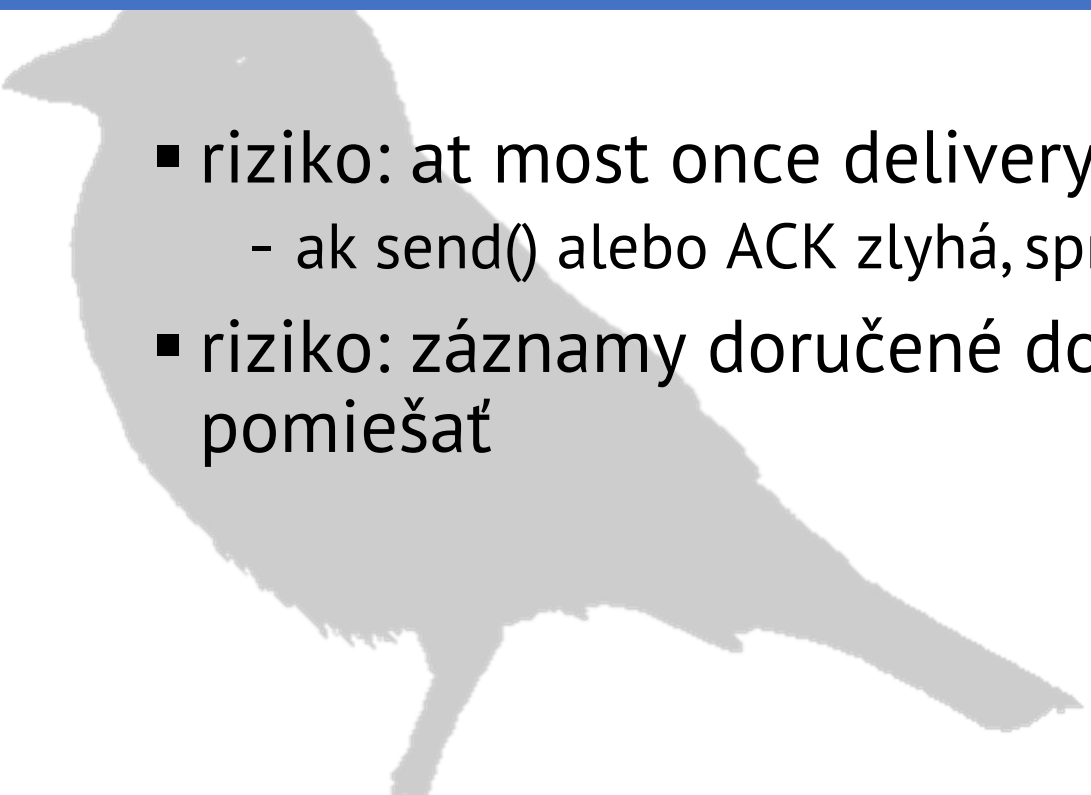
Zvyšovanie výkonu pre producentov

Vypneme idempotentnosť na producentovi
a znížime ACK na 1

- riziko: budú duplikáty pri opakovaniach cez retries
- riziko: ak líder klastra ACKne záznam, a náhodou vypadne, repliky záznam už neuvidia
  - at most once delivery
- riziko: záznamy doručené do partície sa môžu pomiešať

Zvyšovanie výkonu pre producentov

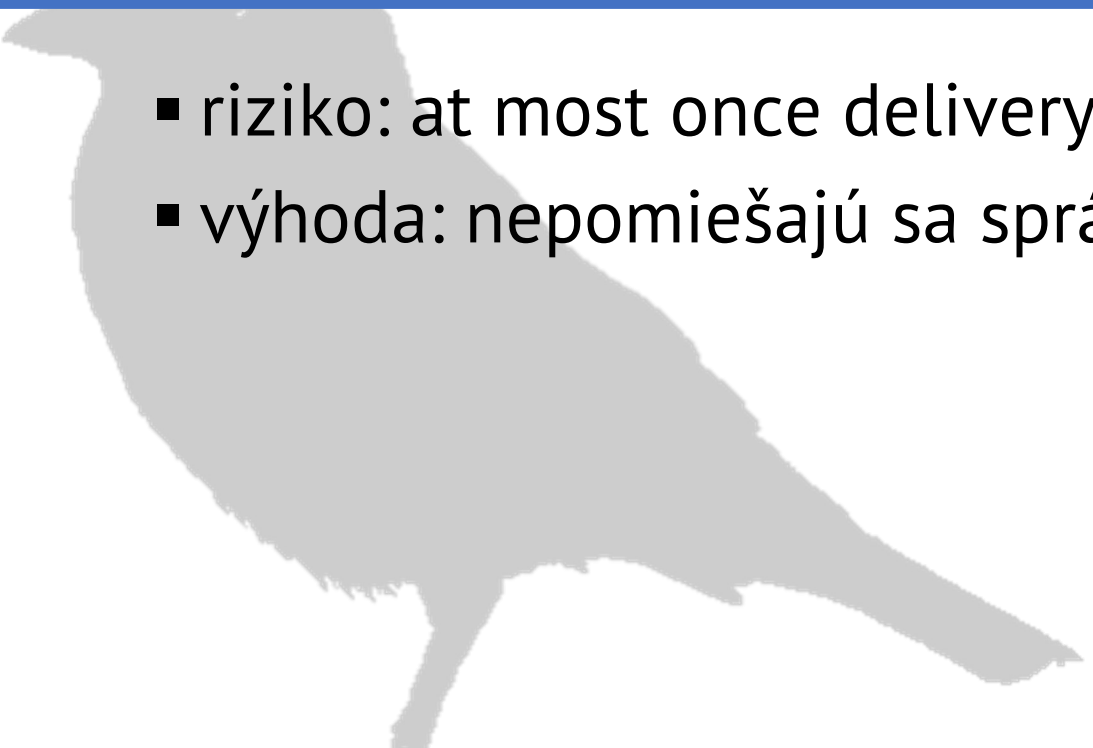
Vypneme idempotentnosť na producentovi
a zakážeme opakovania cez retry



- riziko: at most once delivery
  - ak send() alebo ACK zlyhá, správa sa stráca
- riziko: záznamy doručené do partície sa môžu pomiešať

Zvyšovanie výkonu pre producentov

Vypneme idempotentnosť na producentovi
in-flight necháme na >1
vypneme opakovania retries



- riziko: at most once delivery
- výhoda: nepomiešajú sa správy

Producenti a konfiguračné nastavenia

enable.idempotence

- garancia, že producent zapíše do prúdu presne 1 záznam
- v opačnom prípade sa pri opakovaniach (*retries*) môžu zapísať duplikáty

Kafka 3.0, sept. 2021

max.in.flight.requests.per.connection (5)

- maximálny počet neACKnutých požiadaviek, ktoré klient pošle v 1 pripojení, kým nezačne blokovať
- koľko správ sa pošle na server bez čakania na prvý ACK?
- **<= 5** a **enable.idempotence** garantuje poradie zasielania záznamov
  - inak sa pomieša poradie doručenia
  - opakované záznamy môžu prísť neskôr, hoci boli pôvodne odosielané skôr

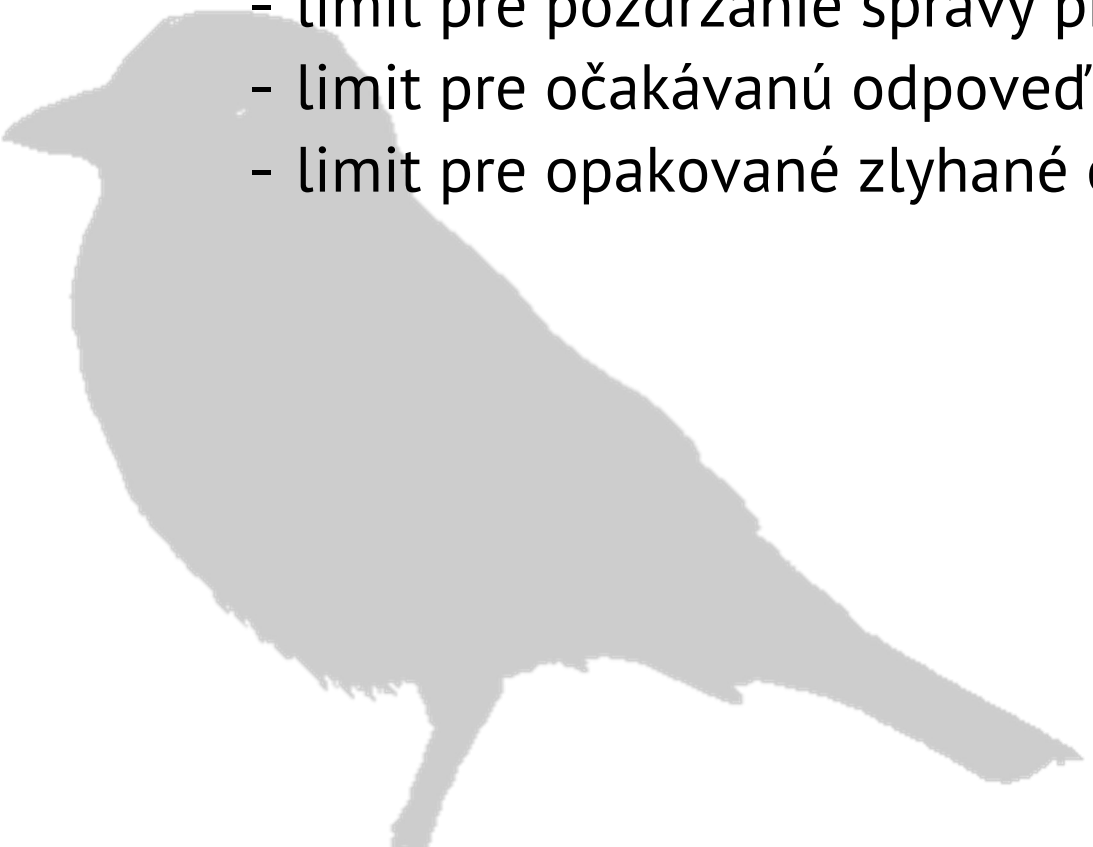
Kafka 3.0, sept. 2021

retries (∞)

- klient zopakuje posielanie záznamov, ak nastala chyba
  - buď **retries**-krát
  - alebo vypršal **delivery.timeout.ms**

delivery.timeout.ms (2 minúty)

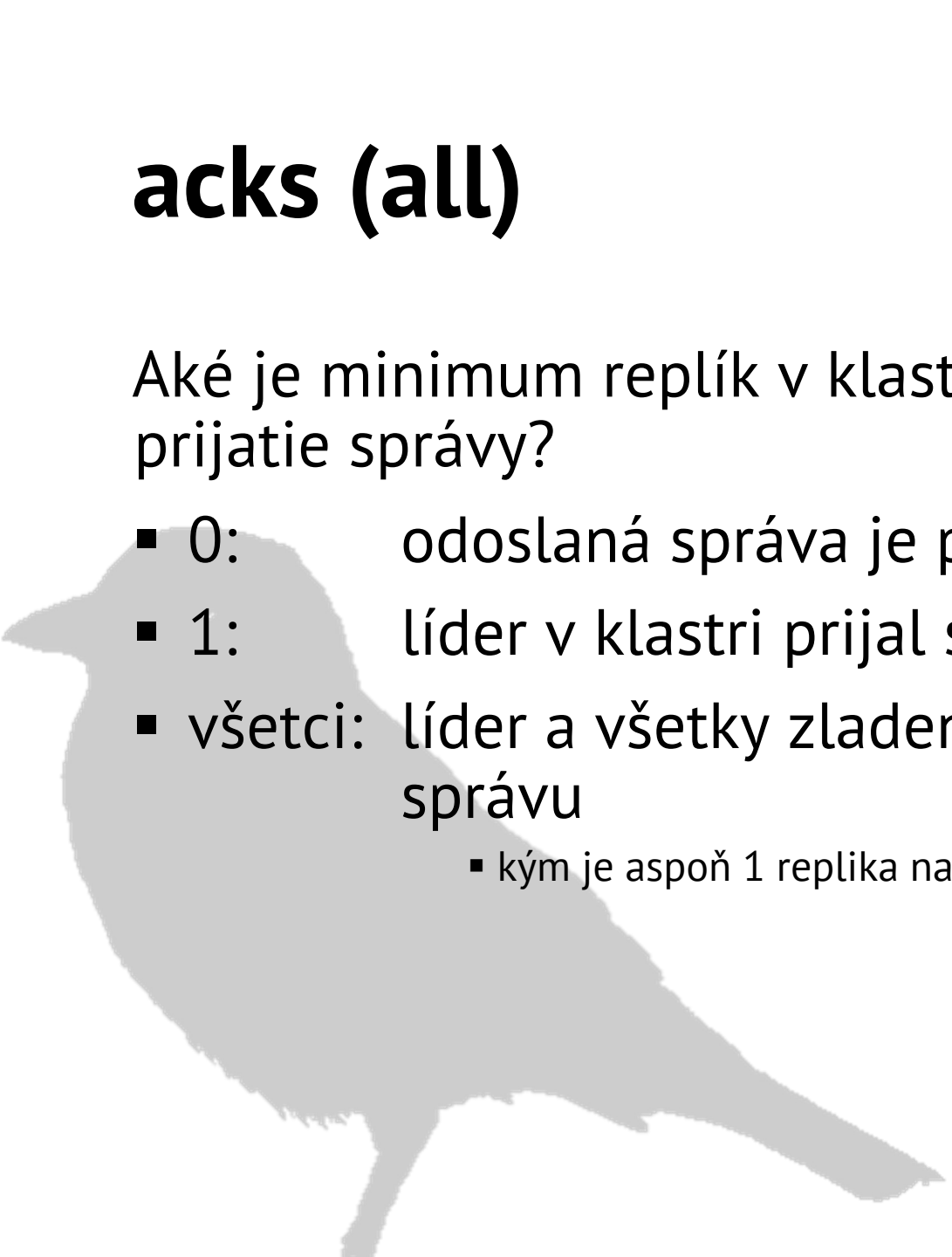
- časový limit, keď **send()** pre producenta zlyhá alebo uspeje
  - limit pre pozdržanie správy pred odoslaním
  - limit pre očakávanú odpoveď od brokera
  - limit pre opakované zlyhané odosielania



acks (all)

Aké je minimum replík v klastrí, ktoré potvrdia prijatie správy?

- 0: odoslaná správa je prijatá správa
- 1: líder v klastri prijal správu
- všetci: líder a všetky zladené repliky prijali správu
  - kým je aspoň 1 replika nažive, záznam sa nestratí



Ďalšie benefity Kafky

- exactly once-delivery
  - preskúmané a funkčné
- podpora pre schému správ
  - JSON / Avro / Protobuf

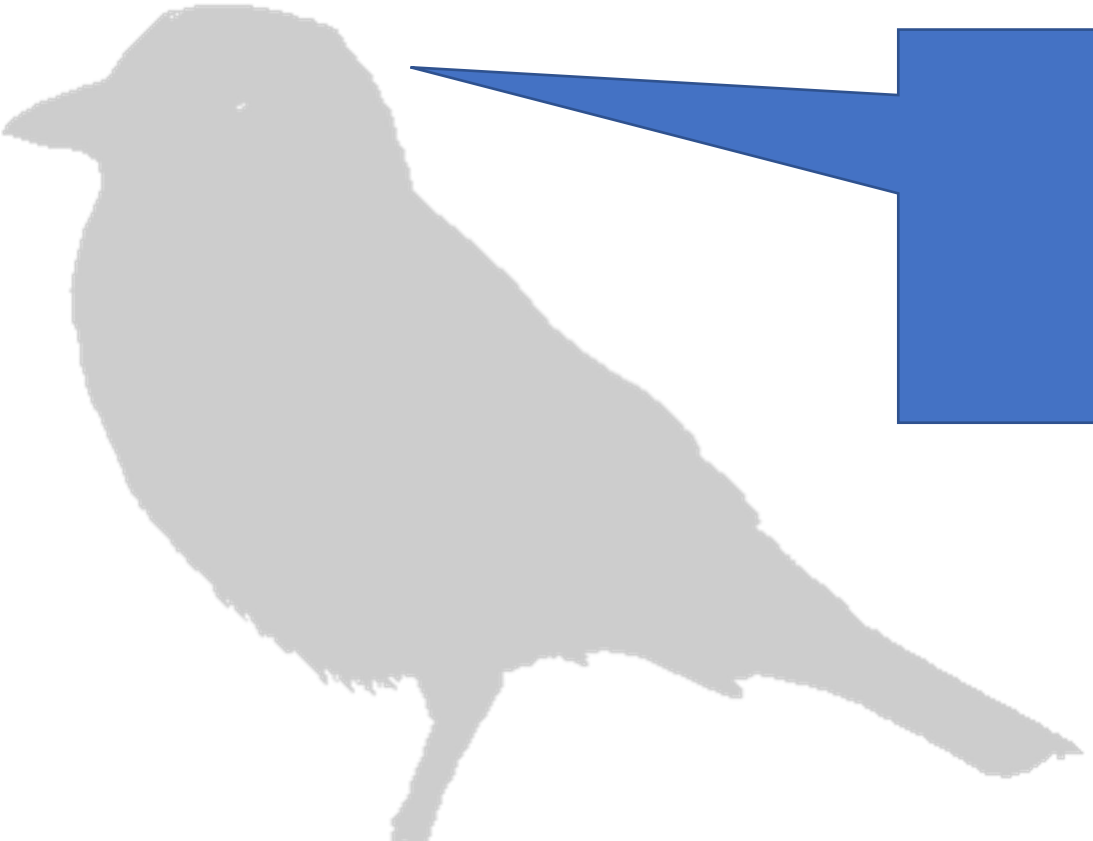
Kafka vs RabbitMQ

Kafka: "dumb broker, smart consumers"

- menej out of box možností, viac výkonu
- výkonnosť je ovplyvnená architektúrou a implementáciou projektu
- implementácia nad JVM

RabbitMQ: "smart pipes, dumb consumers"

- typické topológie sú out of box
- stačí pochopiť architektúru a správne ju použiť
- implementácia je potom jednoduchšia



Otázky?