ČVUT
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

PDV 11 2017/2018

Konsensus a Algoritmu Raft

Michal Jakob
michal.jakob@fel.cvut.cz
Centrum umělé inteligence, katedra počítačů, FEL ČVUT

Co mají tyto příklady společného?

Skupina procesů usilujících o :

- udržování jejich lokálních seznamů aktivních procesů aktuálních [detekce selhání]
- zvolení lídra a zajištění, že každý ví, kdo je lídrem [volba lídra]
- zajistit vzájemně exkluzivního přístupu ke sdílenému prostředku (např. souboru) [vyloučení procesů]
- usilujících o doručení stejných aktualizací ve stejném pořadí [uspořádaný multicast]

Souvisí s konsensem

Ve všech těchto případech se procesy snaží vzájemně koordinovat, aby se shodly na nějaké hodnotě

- na stavu každého procesu (aktivní/neaktivní)
- kdo je lídrem
- kdo má přístup k sdílenému prostředku
- pořadí zpráv

Všechny tyto problémy souvisejí s problémem **konsensu**

Problém konsensu

N procesů

Každý proces P má

- vstupní proměnou x_P (výchozí návrh): zpočátku buď 0 nebo 1
- výstupní proměnou y_p: může být změněna pouze jednou

Problém konsensu: navrhnout takový protokol, že buď:

- všechny procesy nastaví svou výstupní proměnou na 0
- všechny proces nastaví svou výstupní proměnnou na 1

Cílem je **shodnout na hodnotě** výstupní proměnné.

- Procesy nemohou mít hodnotu výstupní proměnné pevně předprogramovanou – výstupní proměnné musí záviset na vstupních proměnných

Proč je konsensus důležitý?

Mnoho problémů v DS je **ekvivalentních** konsensu

- perfektní detekce selhání
- volba lídra
- spolehlivý nebo totálně uspořádaný multicast
- …

Vyřešení konsensu by tedy bylo velmi užitečné.

Řešitelnost konsensu

V **synchronním** DS **je** konsensus řešitelný.

V **asynchronních** DS **není** konsensus řešitelný**.**

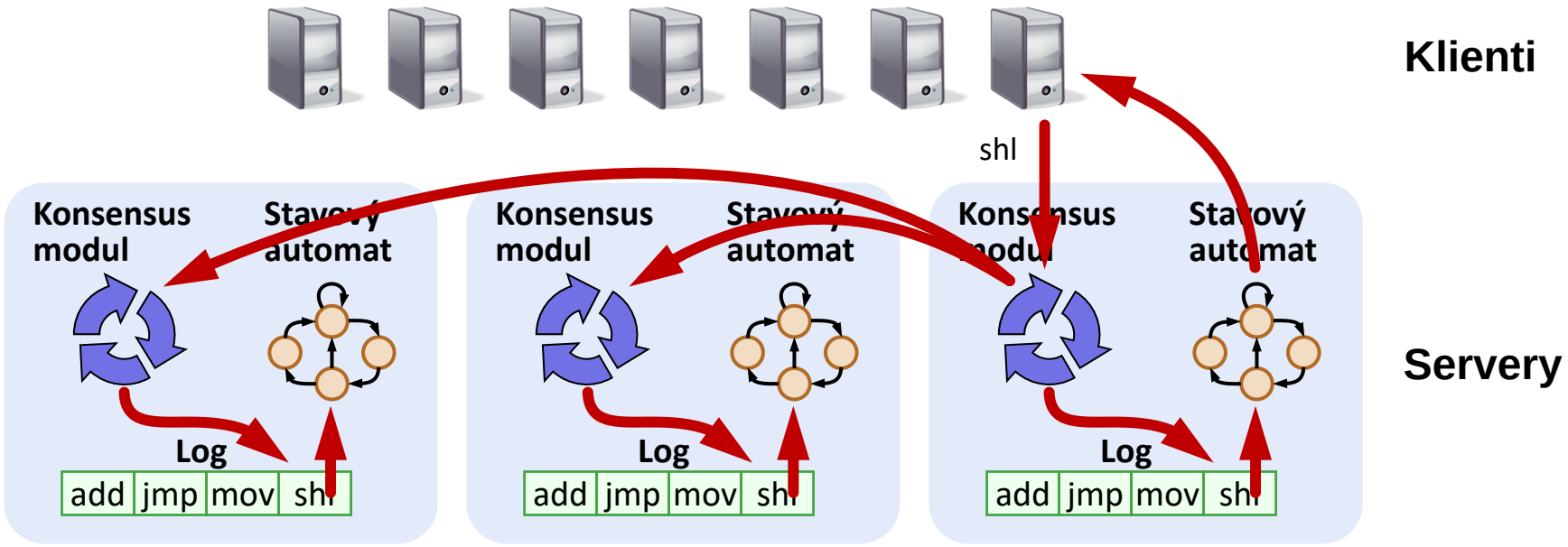
- pro jakýkoliv algoritmus existuje nejhorší možný průběh (se selháními procesů nebo kanálů), který zabrání dosažení konsensu
- důkaz v tzv. **FLP teorém:** V asynchronním systému nelze dosáhnout současně bezpečnosti a živosti, pokud v něm může docházet k selháním (byť i jediného procesu).

Ale: V praxi vždy vyžadujeme bezpečnost a díky částečné synchronicitě ve velkém množství běhů DS konsensu dosáhneme v **konečném čase**.

- existují i pravděpodobnostní algoritmy mající **konečnou středního** hodnotu běhu

Algoritmus RAFT

Cíl: Replikovaný log



Replikovaný log → **replikovaný stavový automat**

- Všechny procesy vykonávají příkazy ve stejném pořadí

Konsensus modul zajišťuje správnou **replikaci logu** a rozhoduje, kdy mohou být příkazy vykonány.

Zpracování požadavků klientů postupuje pokud je **nadpoloviční většina** serverů aktivních.

Model selhání: havárie serveru (fail-stop) + **nedokonalý FIFO** kanál

Přístupu k problému konsensu

Symetrický/bez lídra

- všechny servery mají stejnou roli
- klienti mohou kontaktovat kterýkoliv server

Asymetrický/s lídrem

- v každém okamžiku je jeden server lídrem a ostatní přijímají jeho rozhodnutí
- klienti komunikují s lídrem

Raft využívá lídra – výhody:

- **dekomponuje** problém na 1) **běžný chod** a 2) **změny lídra**
- **zjednodušuje** běžný chod (nedochází ke konfliktům)
- **efektivnější** než symetrické přístupy bez lídra

Přehled Raftu

1. Volba lídra
   - volba jednoho ze serveru jako lídra
   - detekce selhání a vyvolání volby nového lídra
2. Běžný chod (základní replikace logu)
3. Bezpečnost a konzistence po změně lídra
4. Neutralizace starých lídrů
5. Interakce s klienty
6. (Rekonfigurace)

Volba lídra

Stavy serveru[1]

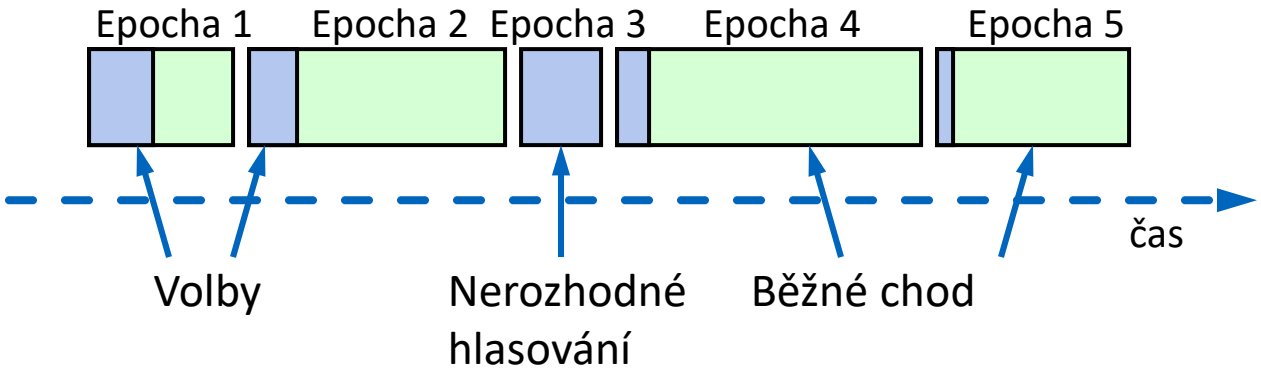
V každém okamžiku je každý server v právě **jednom stavu**:

Lídr	Následovník	Kandidát
obsluhuje požadavky klientů a replikuje log	pasivní – pouze reagují na zprávy od jiných serverů	přechodná role v průběhu volby lídra

Běžný chod: 1 lídr , $N - 1$ následovníků

[1]Skupinu procesů, které se účastní konsensu, označovat jako servery – pro lepší odlišení od procesů, které běží na klientských počítačích a které se konsensu neúčastní.

Epochy (volební období)



Čas je rozdělen do **epoch:** každá epocha má své **číslo**, čísla jsou **inkrementována** a nikdy nejsou znovu použitá. Každý server si (persistentně!) udržuje číslo **aktuální epochy**.

Epochy mohou mít dvě části

- **Volby** (buď nedopadnou nebo vyústí ve zvolení právě jednoho lídra)
- **Běžný chod** pod jedním zvoleným lídrem

Maximálně jeden lídr v každé epoše; některé epochy ale nemají lídra (neúspěšné volby).

Epochy slouží k identifikování **zastaralých informací**.

Raft Protocol Summary

Followers

- Respond to RPCs from candidates and leaders.
- Convert to candidate if election timeout elapses without either:
  - Receiving valid AppendEntries RPC, or
  - Granting vote to candidate

Candidates

- Increment currentTerm, vote for self
- Reset election timeout
- Send RequestVote RPCs to all other servers, wait for either:
  - Votes received from majority of servers: become leader
  - AppendEntries RPC received from new leader: step down
  - Election timeout elapses without election resolution: increment term, start new election
  - Discover higher term: step down

Leaders

- Initialize nextIndex for each to last log index + 1
- Send initial empty AppendEntries RPCs (heartbeat) to each follower; repeat during idle periods to prevent election timeouts
- Accept commands from clients, append new entries to local log
- Whenever last log index ≥ nextIndex for a follower, send AppendEntries RPC with log entries starting at nextIndex, update nextIndex if successful
- If AppendEntries fails because of log inconsistency, decrement nextIndex and retry
- Mark log entries committed if stored on a majority of servers and at least one entry from current term is stored on a majority of servers
- Step down if currentTerm changes

Persistent State

Each server persists the following to stable storage synchronously before responding to RPCs:

currentTerm	latest term server has seen (initialized to 0 on first boot)
votedFor	candidateId that received vote in current term (or null if none)
log[]	log entries

Log Entry

term	term when entry was received by leader
index	position of entry in the log
command	command for state machine

RequestVote RPC

Invoked by candidates to gather votes.

Arguments:

candidateId	candidate requesting vote
term	candidate's term
lastLogIndex	index of candidate's last log entry
lastLogTerm	term of candidate's last log entry

Results:

term	currentTerm, for candidate to update itself
voteGranted	true means candidate received vote

Implementation:

1. If term > currentTerm, currentTerm ← term (step down if leader or candidate)
2. If term == currentTerm, votedFor is null or candidateId, and candidate's log is at least as complete as local log, grant vote and reset election timeout

AppendEntries RPC

Invoked by leader to replicate log entries and discover inconsistencies; also used as heartbeat .

Arguments:

term	leader's term
leaderId	so follower can redirect clients
prevLogIndex	index of log entry immediately preceding new ones
prevLogTerm	term of prevLogIndex entry
entries[]	log entries to store (empty for heartbeat)
commitIndex	last entry known to be committed

Results:

term	currentTerm, for leader to update itself
success	true if follower contained entry matching prevLogIndex and prevLogTerm

Implementation:

1. Return if term < currentTerm
2. If term > currentTerm, currentTerm ← term
3. If candidate or leader, step down
4. Reset election timeout
5. Return failure if log doesn't contain an entry at prevLogIndex whose term matches prevLogTerm
6. If existing entries conflict with new entries, delete all existing entries starting with first conflicting entry
7. Append any new entries not already in the log
8. Advance state machine with newly committed entries

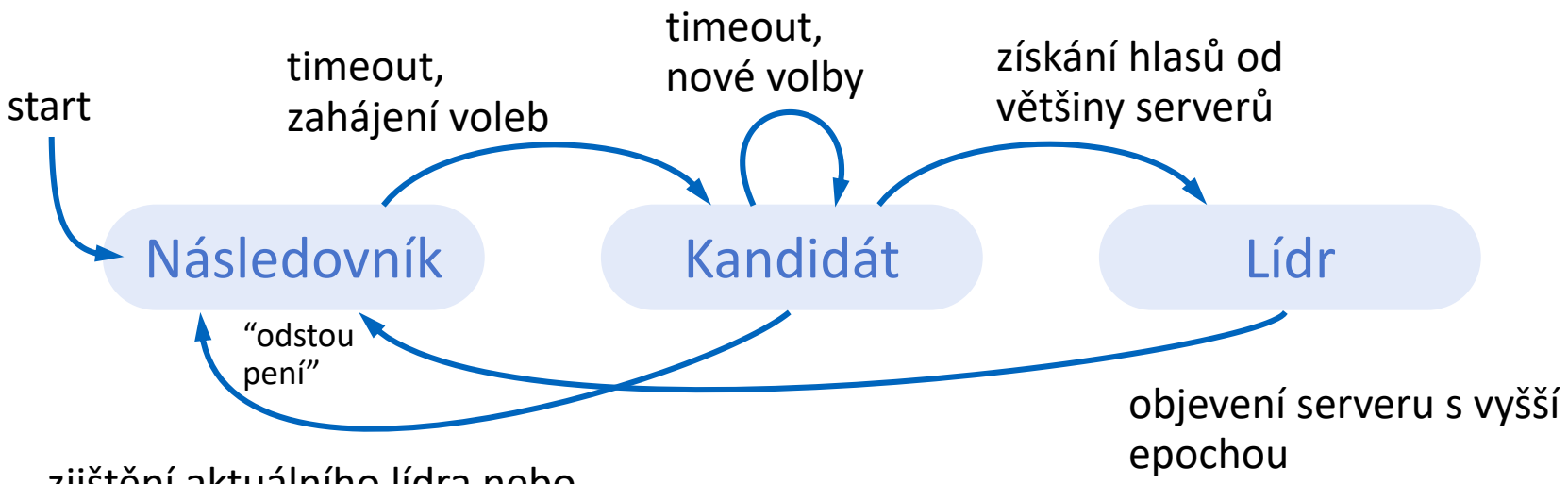
Stavy serveru

Servery začínají jako **Následovníci**.

- Následovníci očekávají zprávy od Lídra nebo Kandidátů

Lídři posílají **heartbeats** (prázdné zprávy **`AppendEntries`**), aby si udrželi autoritu.

Jakmile Následovník neobdrží zprávu do **volebního timeoutu** (typicky 100-500ms), předpokládá, že Lídr havaroval a **iniciuje volbu** nového lídra.



Spuštění voleb

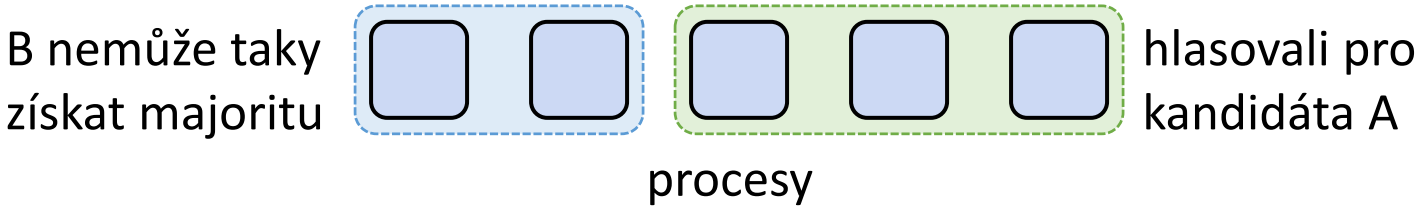
Server, který vyvolá volby, provede následující:

1. Zvýší číslo epochy.
2. Změní svůj stav na KANDIDÁT
3. Zahlasuje pro sebe
4. Pošle **`RequestVote`** všem ostatním serverům a čeká dokud nenastane jedno z následujících:
   1. Obdrží hlasy od většiny serverů:
      - Změní stav na LÍDR
      - Pošle **`AppendEntries`** heartbeats všem ostatním procesům
   2. Přijme zprávu od validního LÍDRA:
      - Vrátí se do stavu NÁSLEDOVNÍK
   3. Nikdo nevyhraje volby (vyprší volební timeout):
      - Zvýší epochu a začné nové volby

Klíčové vlastnosti voleb

Bezpečnost: maximálně jeden vítěz v každé epoše

- každý proces hlasuje pouze jednou v jedné epoše (a hlas persistuje)
- dva kandidáti nemohou získat většinu v jedné epoše



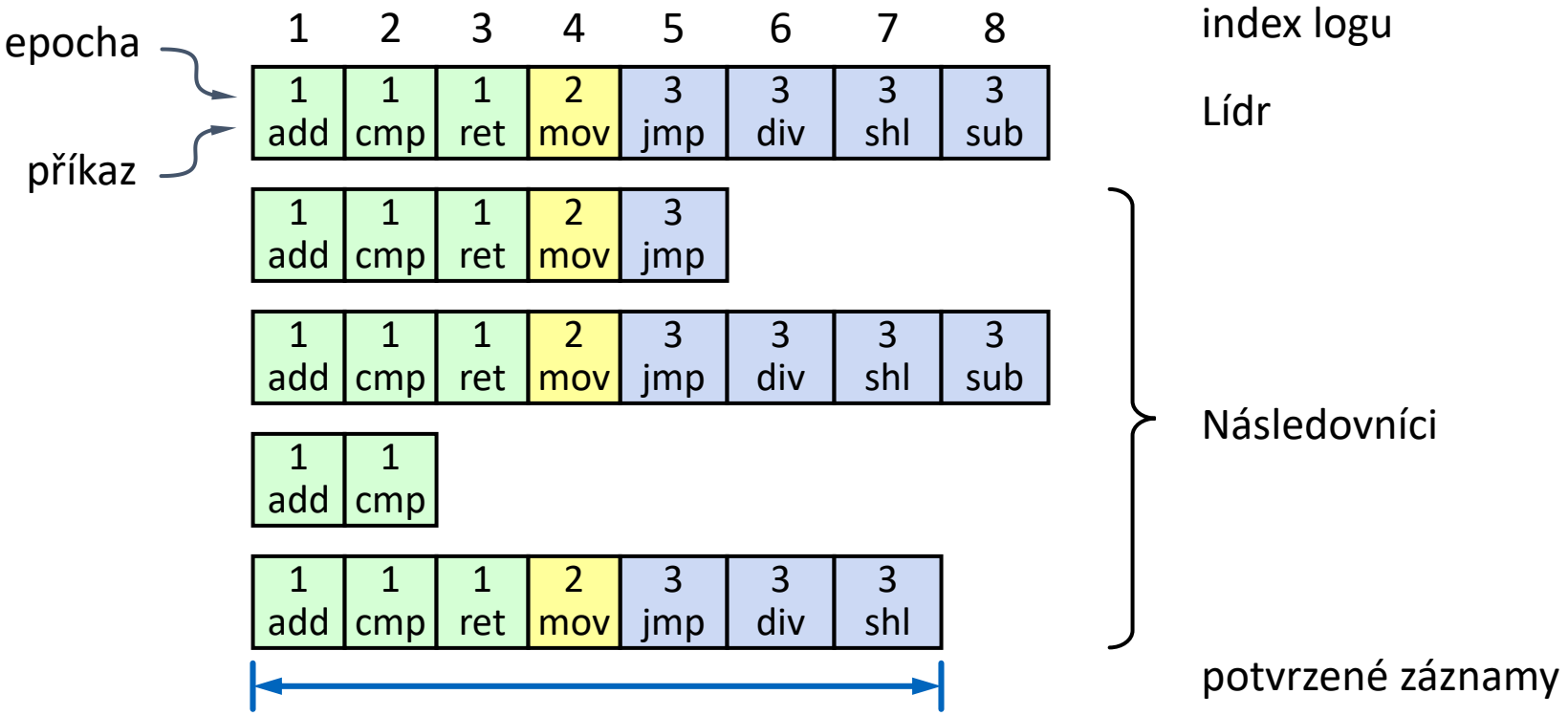
Živost: jeden z kandidátů musím časem vyhrát

- Každý proces volí volební timeout **náhodně** v intervalu $[T, 2T]$
- Jeden proces typicky iniciuje volby a zvítězí dříve, než ostatní začnou
- funguje dobře pokud $T \gg RTT$ (čas oběhu zpráv)

https://raft.github.io/

Běžný chod

Struktura logů



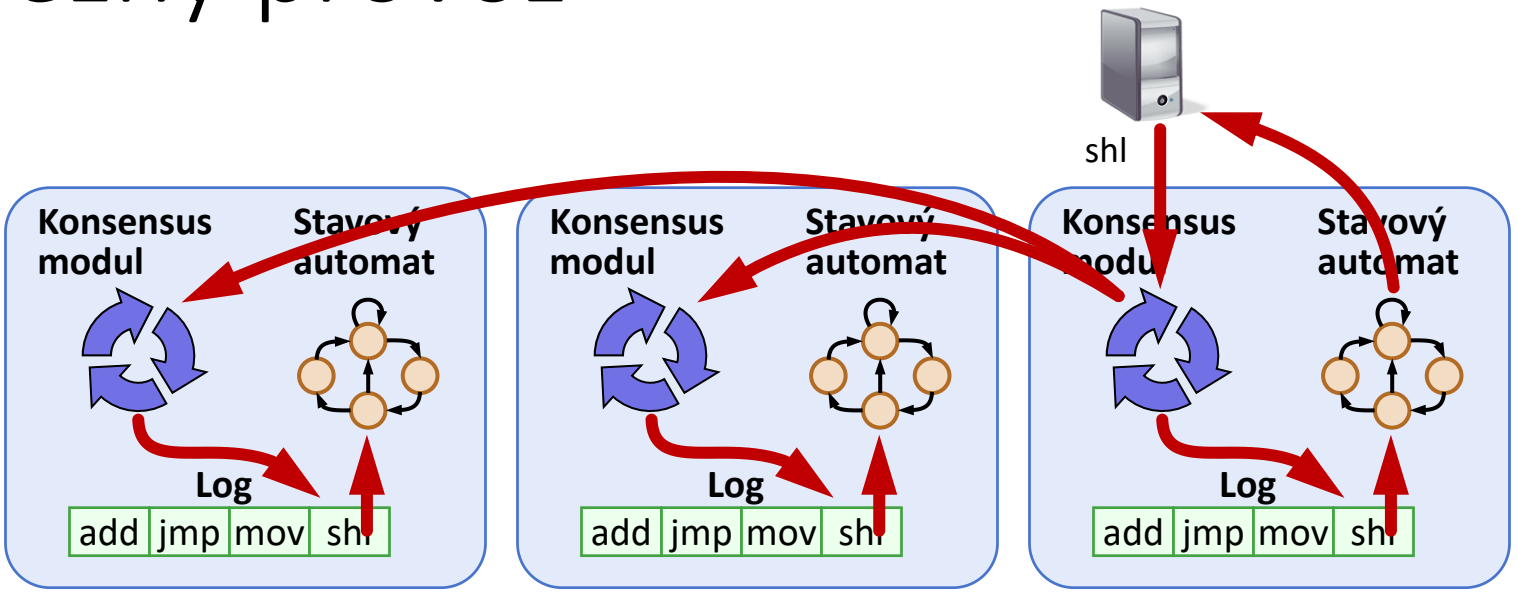
Záznam v logu = < index, epocha, příkaz >

Logy jsou uloženy v **perzistentním uložišti** (disk); tj. přežijí havárie

Záznam je **potvrzený (commited)**, je-li známo, že je uložen ve většině procesů

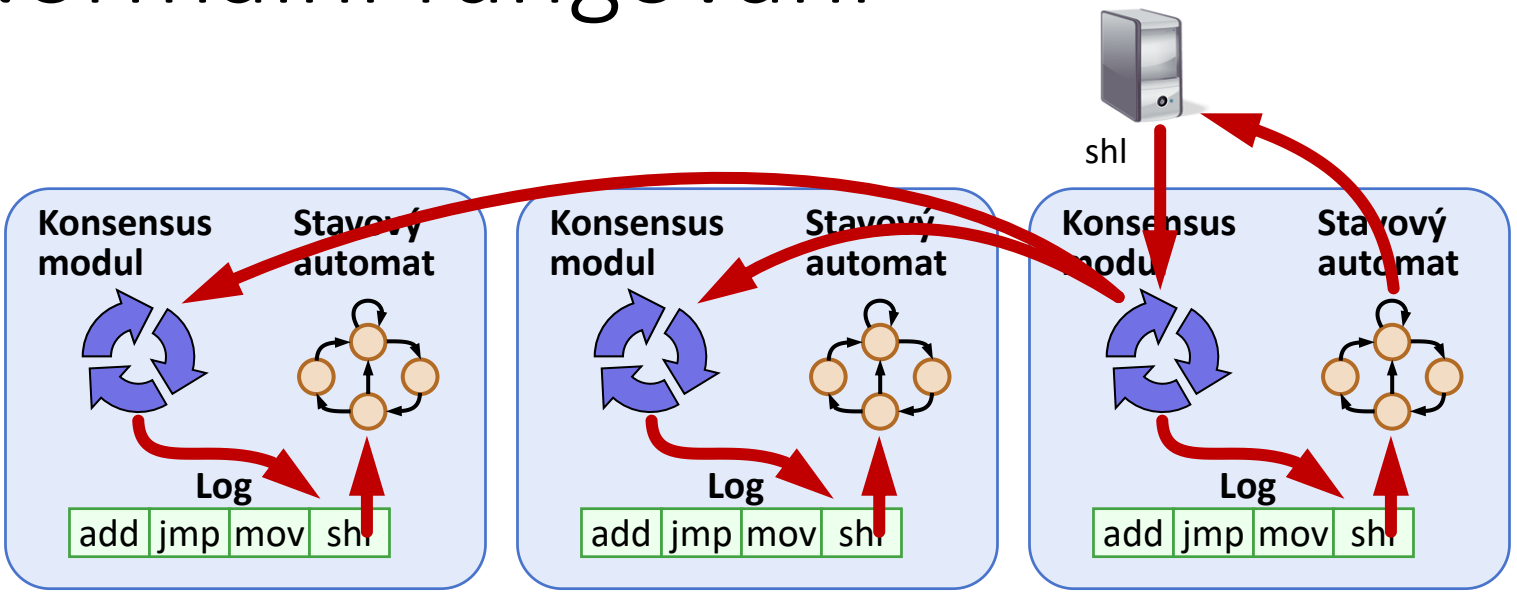
- trvalý: bude nakonec vykonán stavovým automatem

Běžný provoz



1. Klient pošle příkaz lídrovi.
2. Lídr přidá příkaz na konec svého logu.
3. Lídr pošle zprávu **`AppendEntries`** následovníkům, typicky paralelně, a čeká na odpovědi.
4. Jakmile je nový záznam potvrzený (committed)
   - Lídr předá příkaz k vykonání svému stavovému automatu a výsledek pošle kilentovi.
   - Lídr přidá informaci o potvrzení (commit) do následující zprávy **`AppendEntries`** pro následovníky
   - Následovnící předají příkaz svým stavovým automatům.

Normální fungování



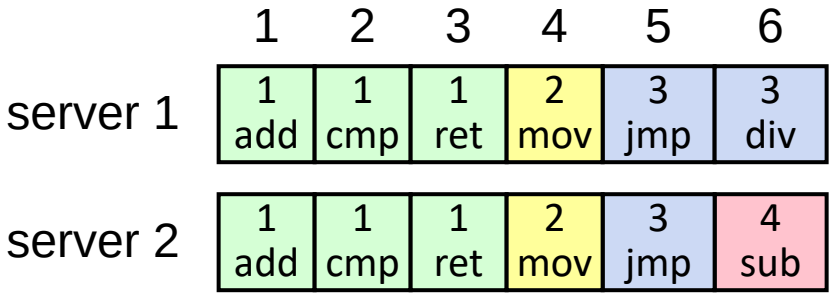
Havarovaní / pomalí následovnící?

- Lídr opakovaně posílá zprávu **`AppendEntries`**, dokud doručení neuspěje

V běžném provozu velmi efektivní:

- Stačí úspěšné doručení **`AppendEntries`** *většině* procesů

Konsistence logů

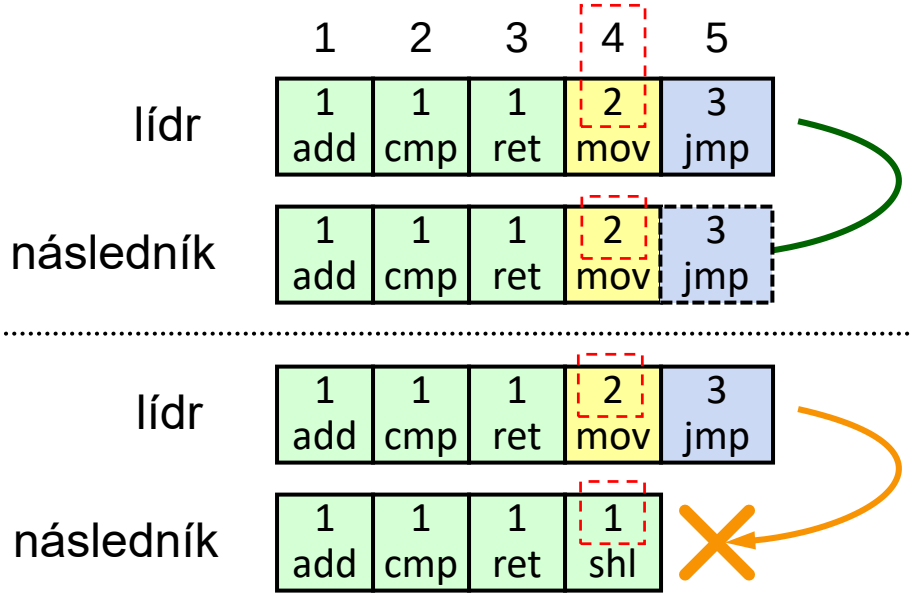


Pro zajištění konsistence Raft vynucuje následující invarianty[1]:

1. Mají-li záznamy logů uložené na různých serverech stejný index a epochu, pak
   - obsahují **stejný příkaz**
   - logy jsou **identické** ve všech **předcházejících** záznamech
2. Je-li daný záznam potvrzený, jsou **potvrzené** i všechny **předcházející** záznamy

[1]Invariant = vlastnost splněna po celou dobu běhu algoritmu

Kontrola konzistence



`AppendEntries` obsahuje `<index,term>` záznamu předcházejícího nově přidávané záznamy.

Následovník musí obsahovat **shodný záznam**; jinak je zápis odmítnut.

Kontrola shodnosti předcházejícího záznamu implementuje **indukční krok** a zajišťuje konzistenci logu.

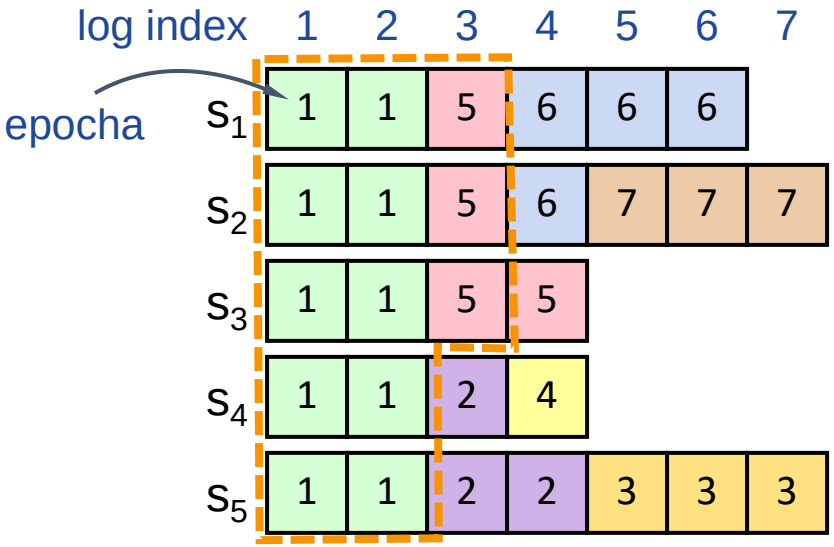
Změna lídra

Změny lídra

Log nového lídra vždy reprezentuje „pravdu“ (**správné záznamy)** – po jeho zvolení není potřeba žadné speciální kroky, vykoná logiku běžného chodu.

- logika běžného chodu *časem* udělá logy následovníků identické s logem nového lídra
- záznamy logu předchozího lídra mohou být částečně replikovány (ale nepotvrzeny) – budou postupně eliminovány

Dojde-li k několika haváriím lídrů po sobě, může být v lozích jednotlivých procesů řada přebytečných záznamů, které budou postupně eliminovány.



Bezpečnost

Obecně nutná bezpečnostní garance pro replikaci

Jakmile je příkaz ze záznamu logu vykonán některým stavovým automatem, nesmí žádný jiný stavový automat vykonat *jiný* příkaz pro stejný záznam.

Bezpečnostní invariant Raftu: Jakmile lídr prohlásí záznam v logu za potvrzený, jakýkoliv budoucí lídr bude mít tento záznam ve svém logu.

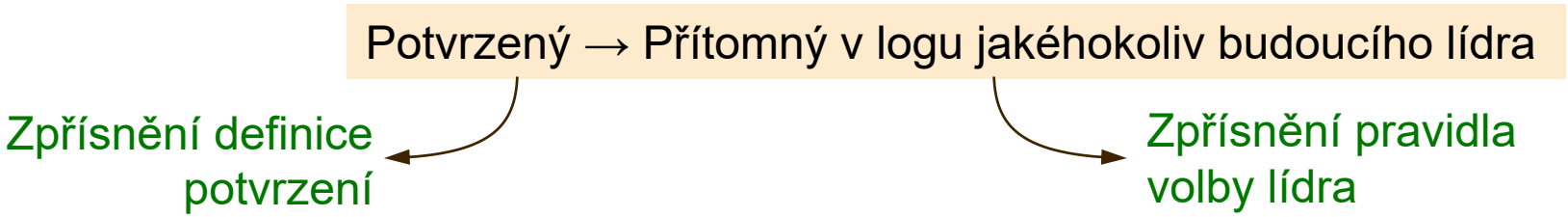
Invariant Raftu implikuje bezpečnostní garanci:

- lídři **nikdy nepřepisují** záznamy ve svých lozích (pouze přidává)
- pouze záznamy **v logu lídra** mohou být **potvrzeny**
- záznamy (příkazy) musí být v logu **potvrzeny předtím**, než jsou **vykonány** stavovým automatem

Zpřísnění Raftu

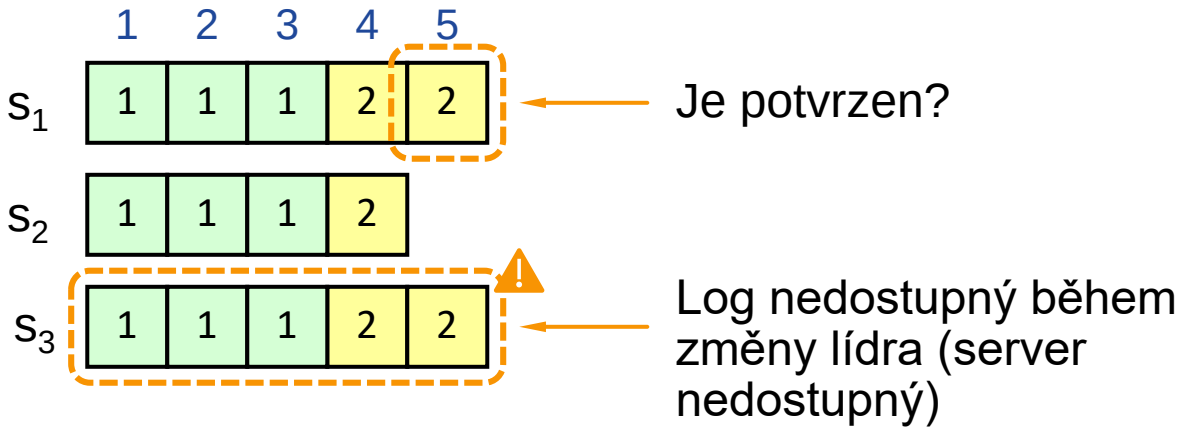
Dosavadní logika fungování Raftu bezpečnostní invariant **negarantuje**.

➔ nutno zpřísnit pravidla:



Výběr nejlepšího lídra

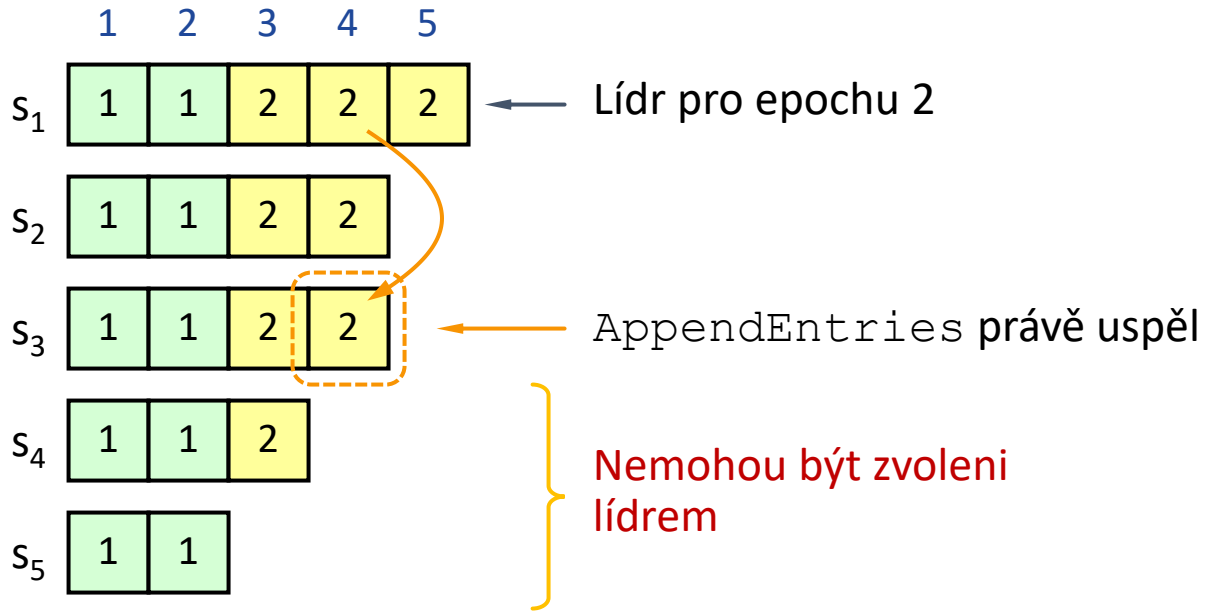
Nelze říct, které záznamy byly potvrzeny



Raft volí kandidáta, u kterého je nejvyšší pravděpodobnost že obsahuje všechny potvrzené záznamy

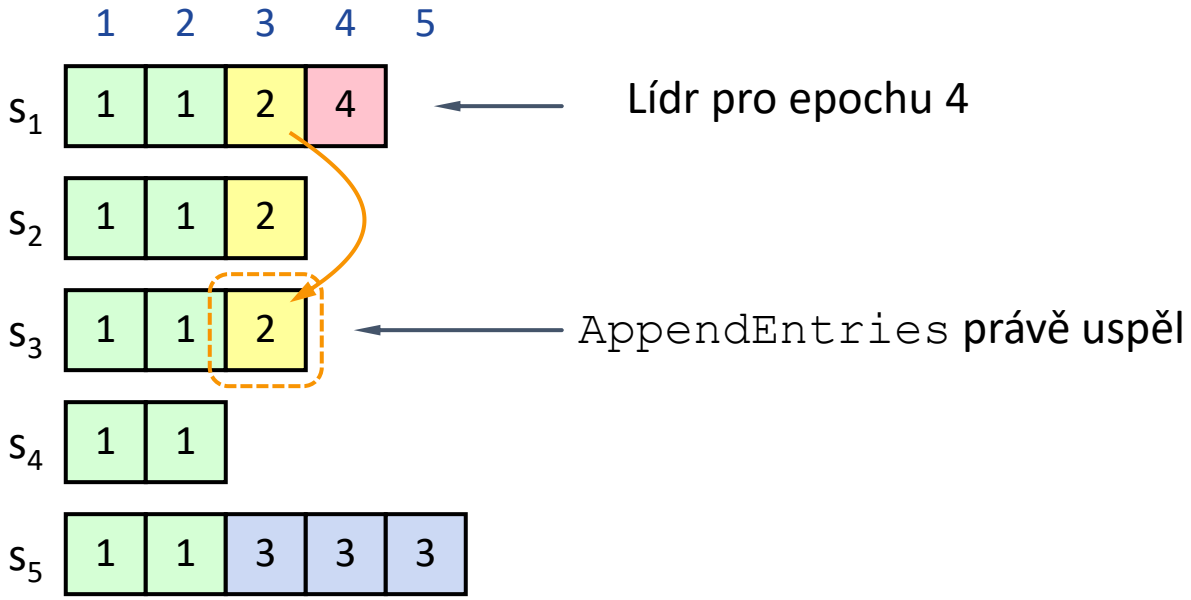
- Kanidáti do zprávy **`RequestVote`** vloží index a epochu posledního záznamu svého logu
- Volící server hlas pro kandidáta odmítne, pokud jeho vlastní log je **úplnější**, tj. pokud má na konci záznam s vyšší epochou nebo stejnou epochou, ale vyšším indexem.
- Lídr tedy bude mít *nejúplnější* log mezi většinou procesů, kterou byl zvolen.

Potvrzování záznamu z aktuální epochy



Záznam 4 je bezpečně potvrzen: jakýkoliv lídr pro epochu tři musí obsahovat v logu záznam 4.

Potvrzování záznamu z dřívější epochy



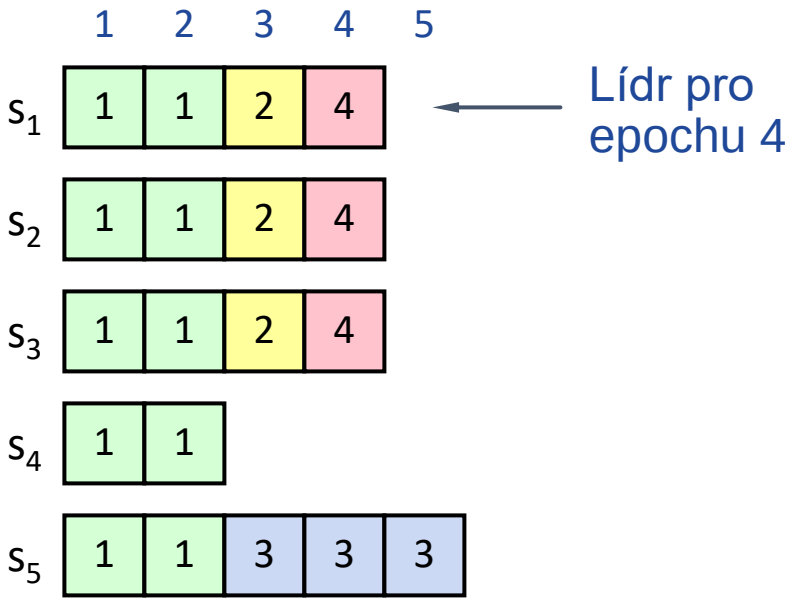
Záznam 3 **není bezpečně** potvrzený

- S_5 může být zvolen jako lídr pro epochu 5
- Byl by-li zvolen, přepíše záznam 3 v S_1 ,S_2 ,S_3

Nová pravidla pro potvrzování

Aby lídr považoval záznam za potvrzený:

1. záznam musí být uložený na většině serverů
2. **aspoň jeden nový záznam** z lídrovy aktuální epochy musí být taky na většině serverů

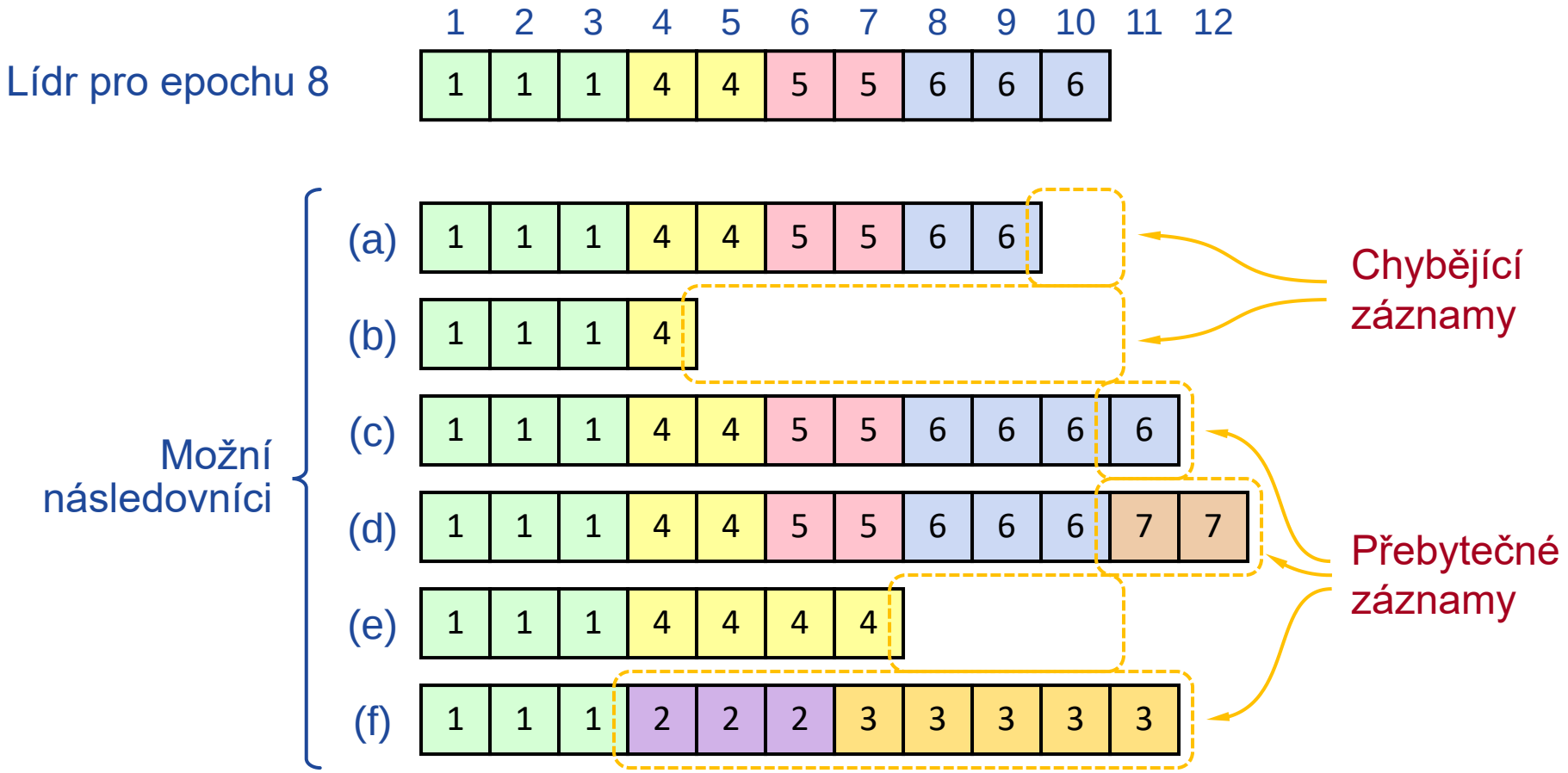


Příklad: Jakmile je záznam 4 potvrzen, S_5 nemůže být zvolen lídrem pro epochu 5 a záznamy 3 a 4 jsou bezpečně potvrzeny.

Kombinace nové pravidla pro výběr lídra a zpřísněné definice potvrzování garantuje bezpečnostní invariant Raftu

Komplikace: nekonzistence logu

Změna lídra mohou vést k nekonzistencím logu



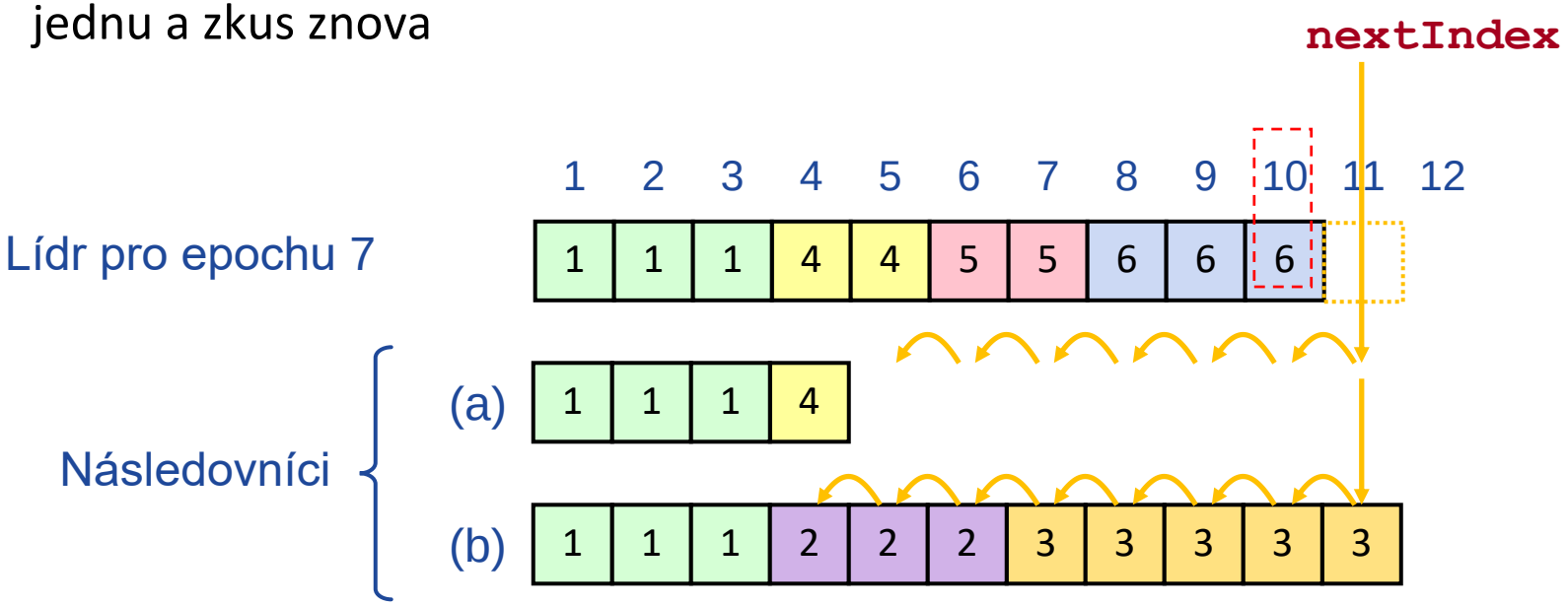
Oprava logů následovníků

Nový lídr musí udělat logy následovníků konzistentní se svým logem, tj. smazat přebytečné záznamy a doplnit chybějící záznam.

Lídr udržuje proměnou **`nextIndex`** pro každého následovníka:

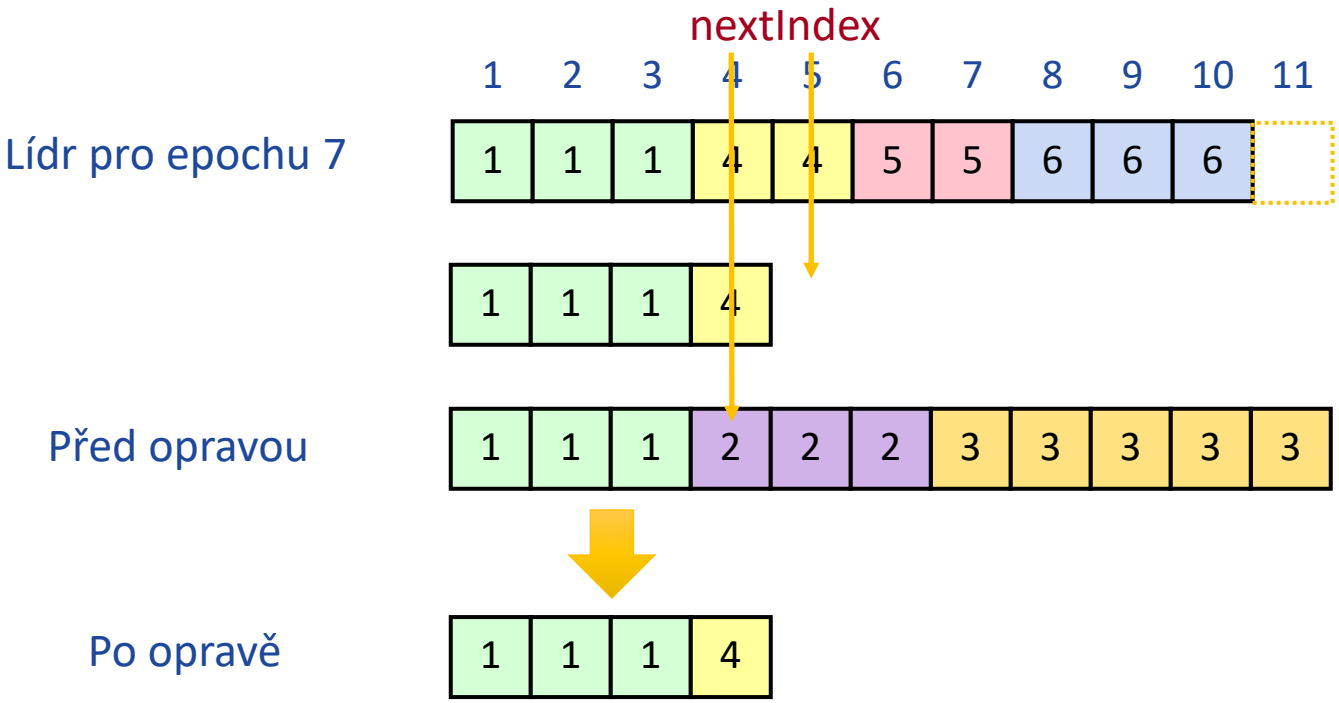
- index další záznamu logu, který by měl být odeslán následnikovi
- inicializován na 1 + index poslední záznamu lídra

Pokud kontrola konzistence **`AppendEntries`** selže, sniž nextIndex o jednu a zkus znova



Oprava logů následovníků

Pokud následovník přepíše nekonzistentní záznam, odstraní i **všechny následující** záznamy.



Neutralizace starého lídra

Neutralizace starých lídrů

Sesazený lídr **nemusí** být **trvale** havarovaný

- přechodné odpojení od sítě
- jiné procesy zvolí nového lídra
- starý lídr se znovu připojí a pokusí se potvrzovat svoje záznamy

Epochy slouží k **detekci neaktuálního** lídra

- každá zpráva obsahuje epochu odesílatele
- je-li epocha odesílatele starší, zpráva je odmítnuta, odesílatel se změní na Následovníka a aktualizuje si epochu
- je-li epocha příjemce starší, tak příjemce se změní na Následovníka, aktualizuje si epochu a následně zprávu normálně zpracuje

Volby aktualizují epochy většiny serverů ➔ sesazení lídři nemohou potvrdit nové záznamy

Klientský protokol

Protokol klienta

Klienti posílají příkazy lídrovi

- Není-li lídr známý, kontaktují libovolný server a ten je případně přesměruje na lídra

Lídr pouze vrací odezvu na příkaz poté, co je příkaz **zalogován**, **potvrzen** a následně vykonán **lídrem**.

Pokud **nepřijde** v časovém limitu **odezva** na požadavek (např. lídr havaroval):

- klient vybere (náhodně) jiný server
- a po případném přesměrování nakonec odešle příkaz novému lídrovi

Protokol klienta: jediné vykonání

Lídr může havarovat poté, co vykonal příkaz, ale před odesláním odpovědi

➔ Riziko opakovaného vykonání příkazu.

Řešení: Pro zajištění právě **jednoho vykonání** příkazu klient vloží unikátní ID příkazu do každého požadavku

- Toto ID je uložené v záznamech v logu

Před přijetím požadavku lídr zkontroluje, zda-li už nemá záznam se stejným ID ve svém logu

- Pokud ne → příkaz vykoná;
- Pokud ano → příkaz odmítne.

Souhrn

Problém konsensu je v jádru mnoha problémů v DS.

V asynchronním DS nelze při přítomností selhání konsensus vyřešit ve smyslu bezpečnosti a živosti.

Praktická řešení garantují bezpečnost.

Raft je moderní algoritmus pro replikaci logů / výpočtů.

Je využíván v řadě reálných DS.