

BGP

Byzantine generals/Byzantine fault tolerance/Byzantine agreement



Peter Borovanský, KAI, I-18,
borovan(a)ii.fmph.uniba.sk

Zdroje:

- klasika, 1982: L.Lamport, R.Shostak, M.Pease
 - <https://people.eecs.berkeley.edu/~luca/cs174/byzantine.pdf>
- pekne vysvetlené, ale nenaprogramuješ to z toho (asi):
 - <https://www.youtube.com/watch?v=e4wNoTV3Gw>
- programátorské vysvetlenie, ale ...
 - <https://marknelson.us/posts/2007/07/23/byzantine.html>
- Blockchain and BGP
 - https://www.youtube.com/watch?v=q3ja_07MFr8
 - <https://www.binance.vision/blockchain/byzantine-fault-tolerance-explained>



Tažba PARACoin-u

(Banícka)



Prémia prednášková 1: Zahrajte sa na ťažiča - minera.

Nájdite x, aby sha-256("PARA"+x) malo aspoň

- 4 úvodné nuly, [prvý má 0.125 bodu]
- 5 úvodných núl, [prvý má 0.25 bodu]
- 6 úvodných núl, [prvý má 0.5 bodu]
- 7 úvodných núl, [prvý má 1 bod] Matej, 18:02
- 8 úvodných núl, [prvý má 2 body] Filip 11:10, Dáša 14:16
- 9 úvodných núl, [prvý má 4 body] Lukáš, 20:08 (4, Roman 21:49)
- 10 úvodných núl, [prvý má 8 bodov]
- 11 úvodných núl, [prvý má 16 bodov]
- 12 úvodných núl, [prvý má 32 bodov ďalší nič]
- 13 úvodných núl, [prvý má 64 bodov]
- 14 úvodných núl, [prvý má instantne Ačko do indexu]



Potreba dohody

na čomkoľvek (nejakom rozhodnutí) ak komunikujeme

- ako separátne jednotky, distribuovane
- posielaním správ
- ktoré ale nemusia doraziť, alebo dorazia neskôr, alebo majú time-out
- môžu prísť modifikované
- s falošným odosielateľom, obsahom
- a okrem toho, niektoré uzly sú infikované, záškodníci
- občas nepošlú správu, ktorú by mali
- alebo zámerne klamú

Aplikácie:

- v systémoch, kde ide o peniaze, a nemajú centrálnu autoritu (blockchain)
- v systémoch, kde ide o život, riadiace systémy

Zahrajme sa

Host A **sends** a TCP **SYN**chronize packet to Host B

Host B receives A's **SYN**

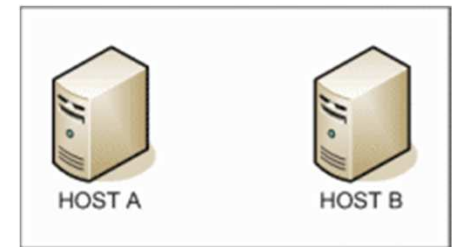
Host B **sends** a **SYN**chronize-**ACK**nowledgement

Host A receives B's **SYN-ACK**

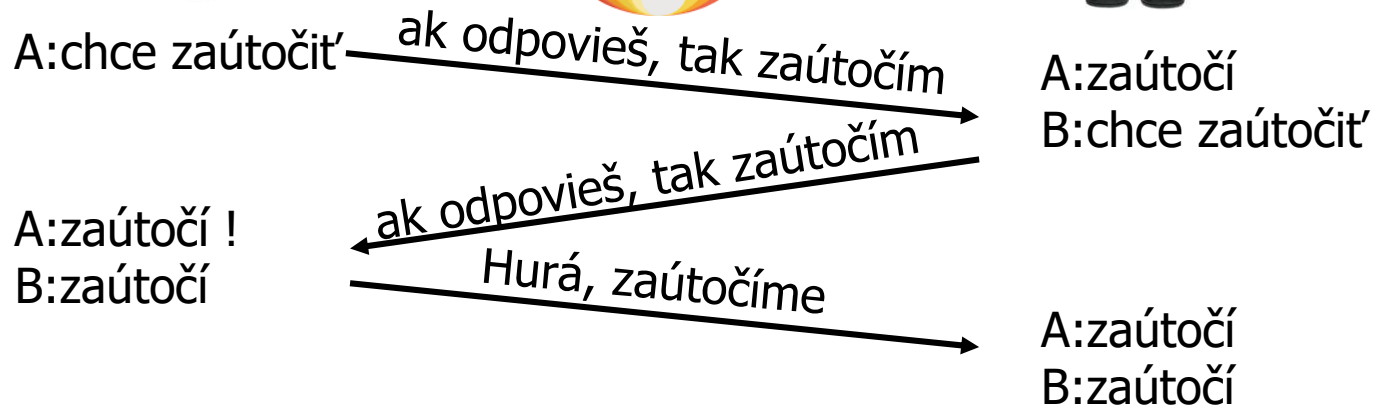
Host A **sends** **ACK**nowledge

Host B receives **ACK**.

TCP socket connection is ESTABLISHED.



Two generals



Dvaja generáli sa majú dohodnúť, či zaútočia alebo nie, ale musia sa zhodnúť
Komunikujú pomocou poslíčkov cez vojnové pole, ich správy nemusia byť doručené
Neexistuje protokol, ktorý by dosiahol a zaručil zhodu generálov



Byzantine General Problem

Komunikácia používa tzv. oral messages

- uzly komunikujú napriamo
- obsah a odosielateľ správy sú pod kontrolou odosielateľa
 - ak ste klamár, tak v správe klamete, ale je jasné, od koho správa prišla,
 - správa príde v konečnom čase, nestratí sa

Malá časť uzlov sú môže byť falošných a systém sa vie správne rozhodnúť

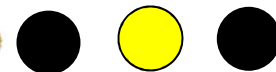
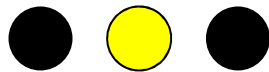
A to je:

- všetci čestní sa rozhodli rovnako
- falošných rozhodnutie nás nezaujímajú

1982: ak falošných je menej ako $1/3$ všetkých, tak to ide.

Resp. Počet všetkých je N , falošných M , tak musí platiť, že $N > 3.M$

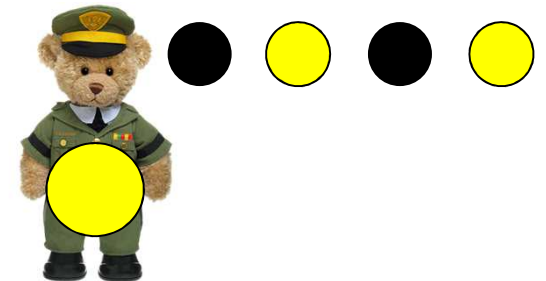
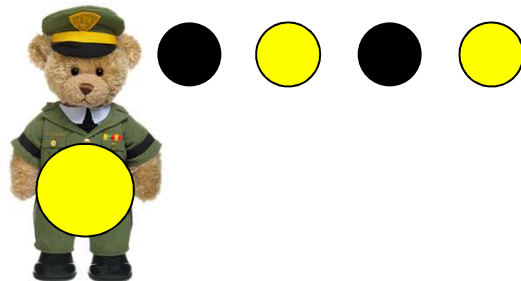
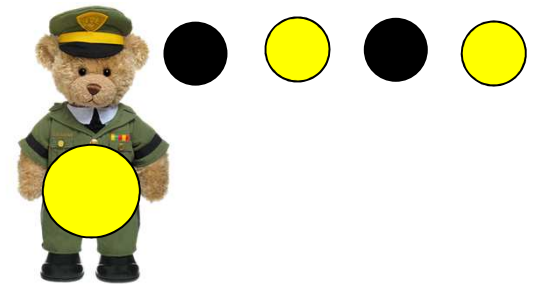
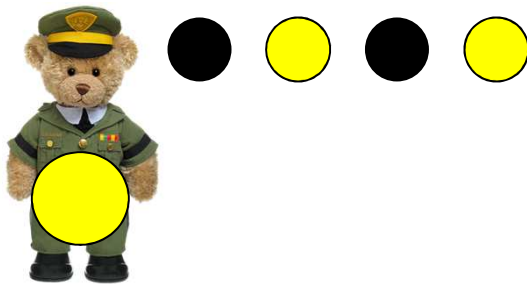
Three generals



Byzantínski generáli majú problém, že

- poslíčkovia/posielanie správ funguje spoľahlivo, dá sa overiť obsah aj odosielateľ
- ale niektorí generáli sú falošní
- všetci féroví generáli sa musia zhodnúť

Four generals

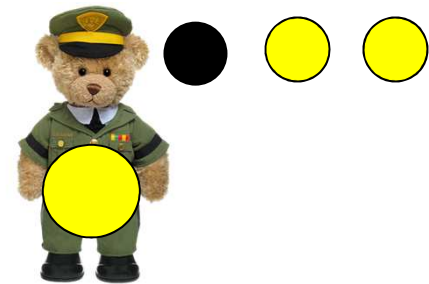
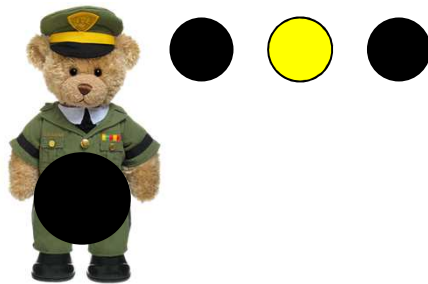


Byzantínski generáli pri remíze neútočia, lenivosť

Majorita – je väčšinový hlas, a v prípade remízy, napr. neútočiť

Podstatné, že pri sa remíze všetci musia rozhodnúť pre tú istú voľbu

Three generals

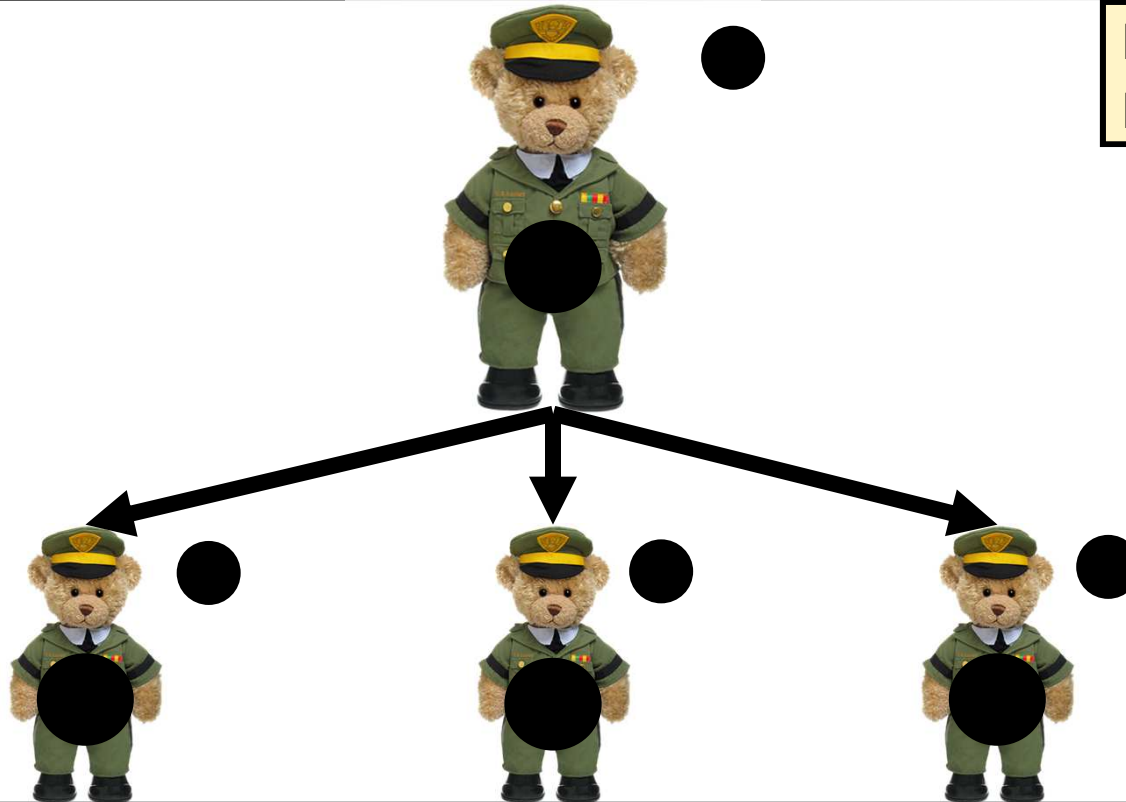


Zatiaľ nikto neklamal, čo ak je tam jeden klamár

Na konečnom názore klamára nezáleží, ale on ovplyvňuje ostatných rozhodnutia
Generál 3 síce chce útočiť (čo je irelevantné), tak klamal generálovi 2, že nechce

Four generals

$N = 4$ – všetci
 $M = 0$ klamári



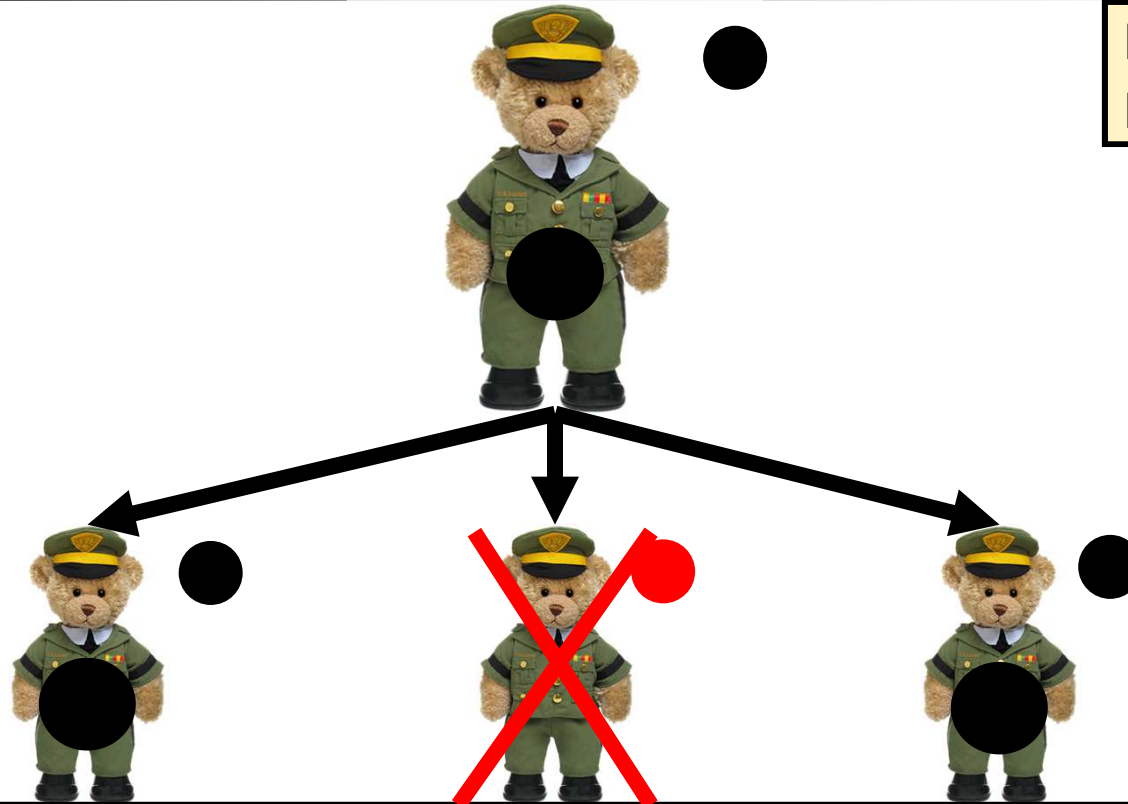
BGP – jeden veľký generál, a zvyšní nižší dôstojníci

Ak je generál čestný, všetci čestní sa majú zhodnúť na rozhodnutí generála

Ak je generál klamár, všetci čestní sa majú zhodnúť na nejakom rovnakom rozhodnutí

Four generals

$N = 4$ – všetci
 $M = 1$ klamár

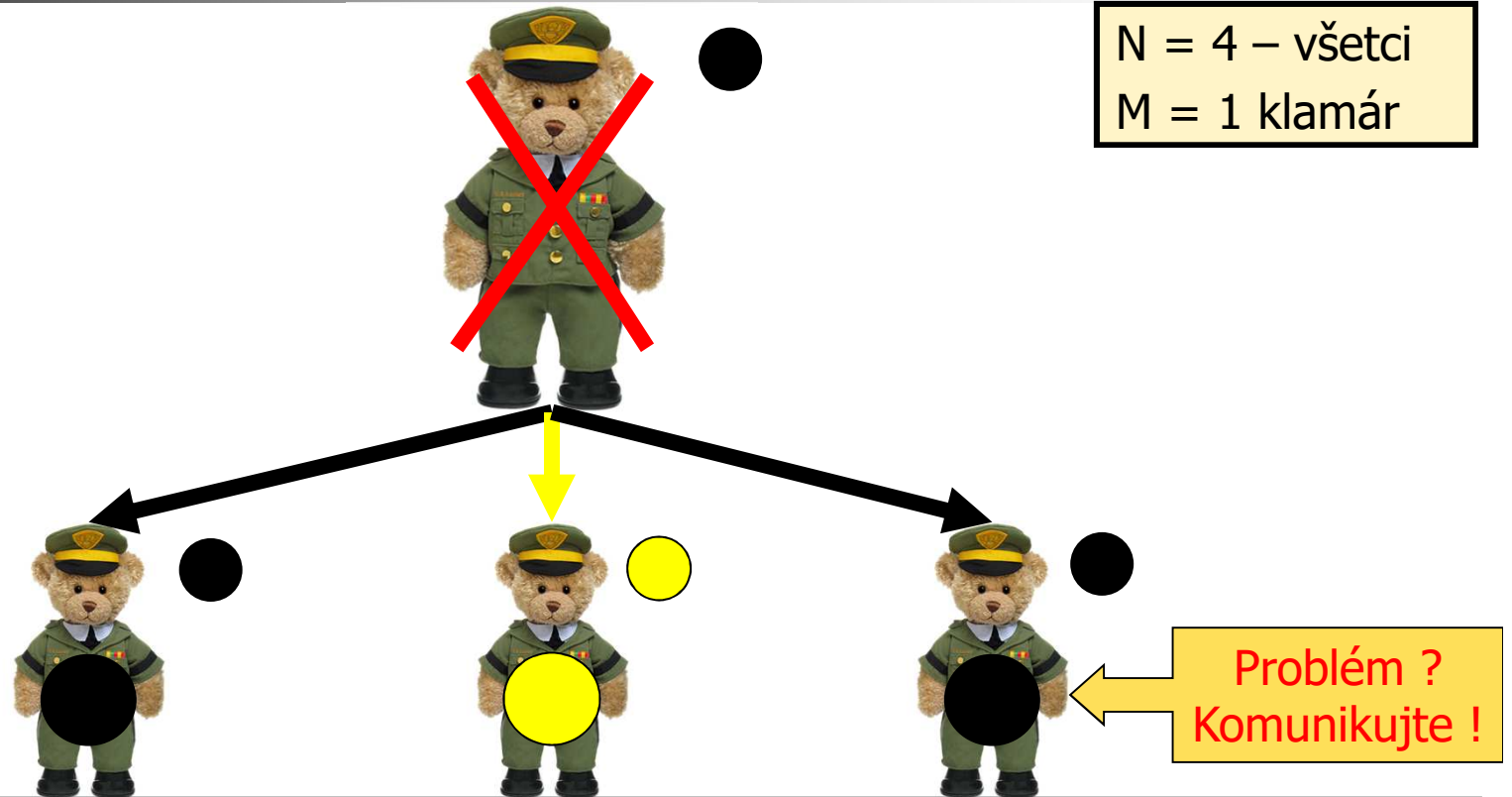


BGP – jeden veľký generál, a zvyšní nižší dôstojníci

Ak je generál čestný, všetci čestní sa majú zhodnúť na rozhodnutí generála

Ak je generál klamár, všetci čestní sa majú zhodnúť na nejakom rovnakom rozhodnutí

Four generals

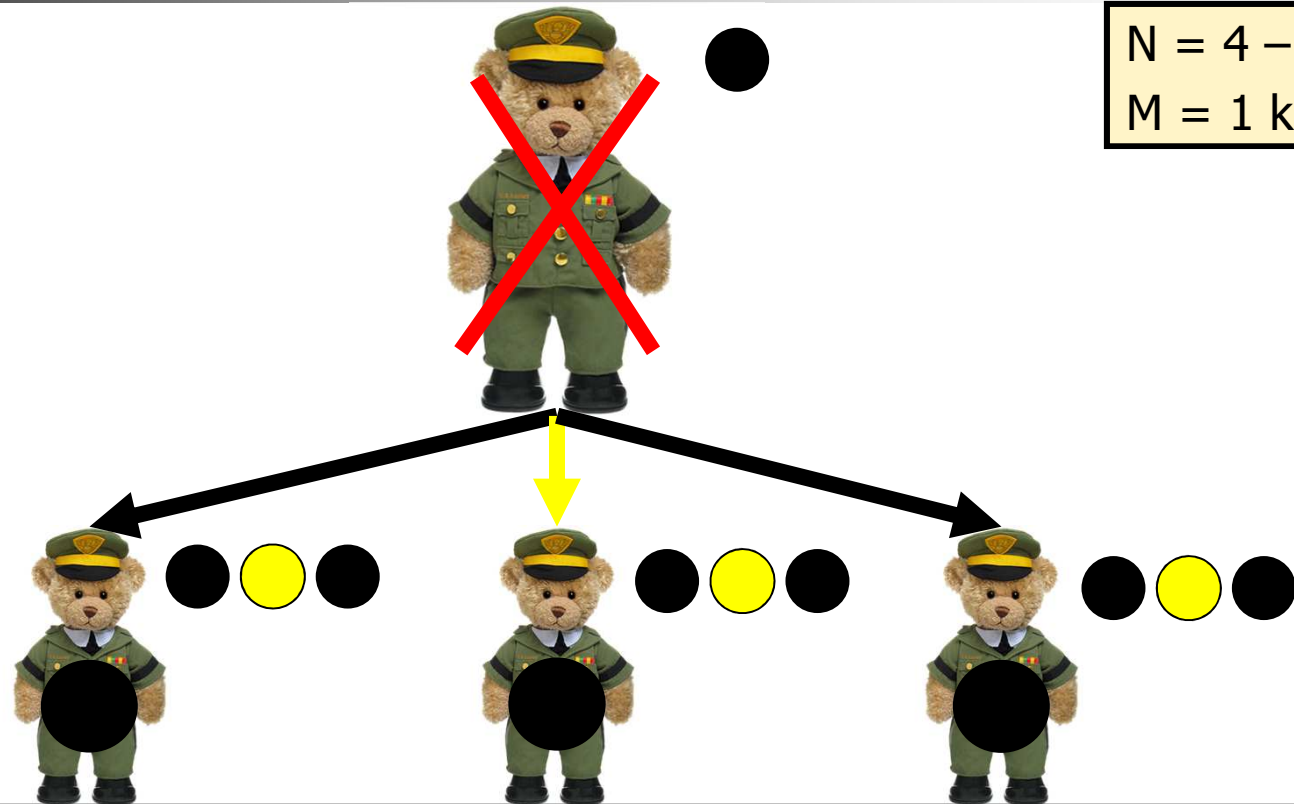


BGP – jeden veľký generál, a zvyšní nižší dôstojníci

Ak je generál čestný, všetci čestní sa majú zhodnúť na rozhodnutí generála

Ak je generál klamár, všetci čestní sa majú zhodnúť na nejakom rovnakom rozhodnutí

Four generals



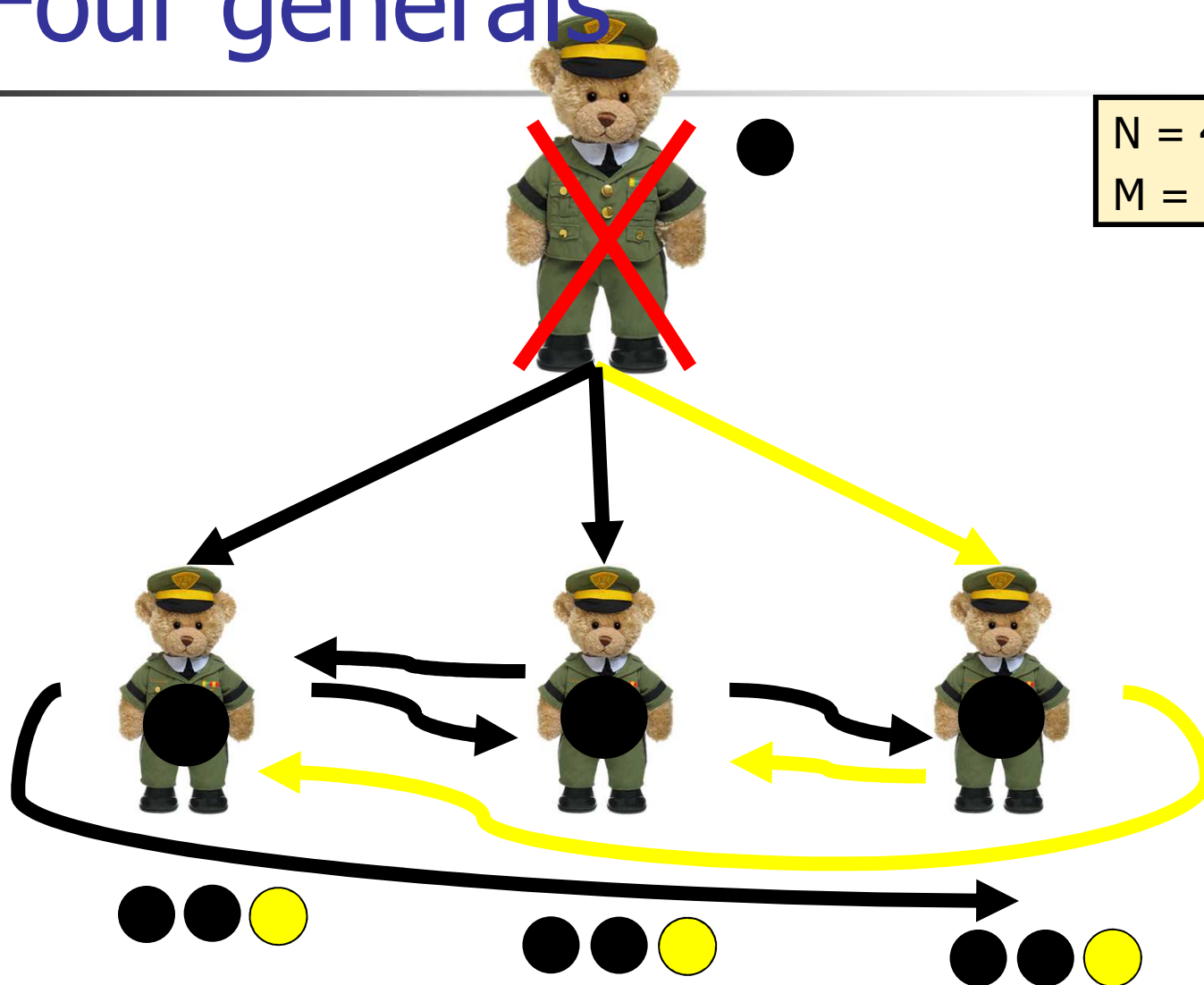
$N = 4$ – všetci
 $M = 1$ klamár

BGP – jeden veľký generál, a zvyšní nižší dôstojníci

Ak je generál čestný, všetci čestní sa majú zhodnúť na rozhodnutí generála

Ak je generál klamár, všetci čestní sa majú zhodnúť na nejakom rovnakom rozhodnutí

Four generals

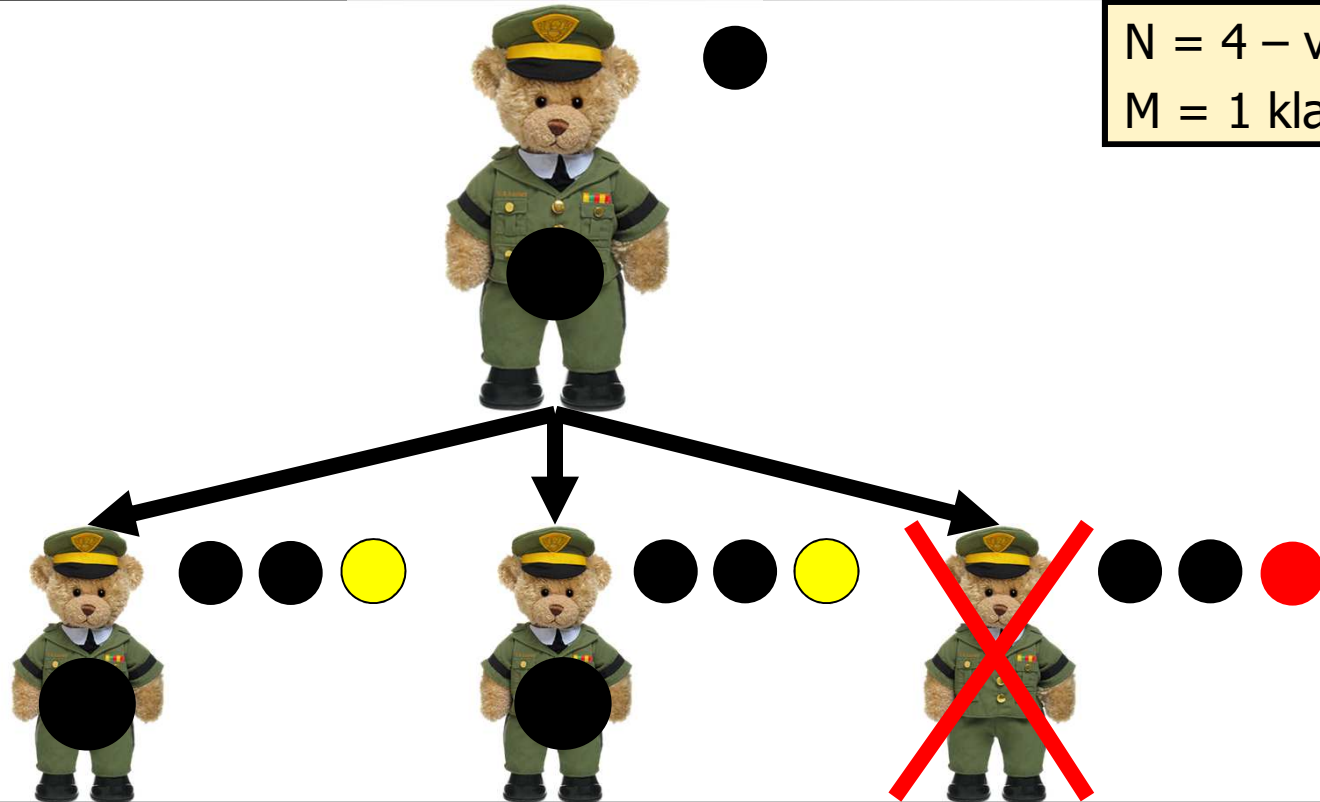


$N = 4$ – všetci

$M = 1$ klamár

Four generals

$N = 4$ – všetci
 $M = 1$ klamár

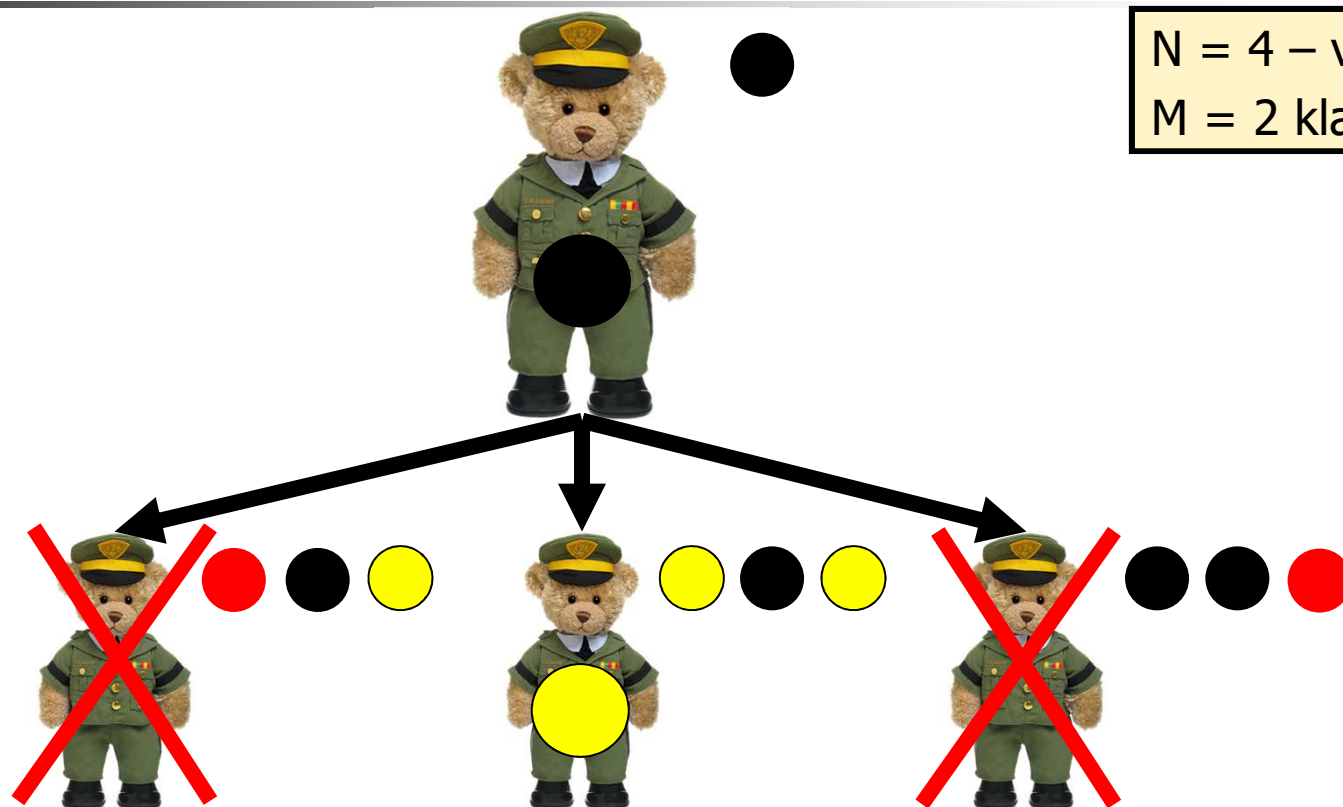


BGP – jeden veľký generál, a zvyšní nižší dôstojníci

Ak je generál čestný, všetci čestní sa majú zhodnúť na rozhodnutí generála

Ak je generál klamár, všetci čestní sa majú zhodnúť na nejakom rovnakom rozhodnutí

Four generals



$N = 4$ – všetci
 $M = 2$ klamári

BGP – jeden veľký generál, a zvyšní nižší dôstojníci

Ak je generál čestný, všetci čestní sa majú zhodnúť na rozhodnutí generála

Ak je generál klamár, všetci čestní sa majú zhodnúť na nejakom rovnakom rozhodnutí



Formát kolujúcich správ

Každá správa obsahuje

- rozhodnutie-informáciu, či zaútočíme alebo nie (input : int)
- cestu, ktorou správa prešla (path : []int – IDčka uzlov, cez ktoré UŽ prešla)
- IDčka uzlov/dôstojníkov, ktorých ešte nenavštívila (others : []int)
- hĺbkou, pokiaľ treba správu - M+1 kôl, začíname level=M+1 a klesáme k =0
- keďže ide o rekurziu, tak je jasné, že končíme pri level = 0

```
type Message struct {  
    level    int    // starting with level M+1 down-to 0  
    input    int    // propagating value int  
    path     []int  // path of the message, e.g. {0,1,3}  
    others   []int  // list of nodes, message to be forwarded to  
}
```



Fáza šírenia správ

Osoby a obsadenie: 0 je generál, $1..N-1$ sú dôstojníci

Generál sa nejako rozhodne a pošle svojim $N-1$ dôstojníkom po jednej správe: dôstojníkovi $i \in \{1..N-1\}$ pošle správu:

{level: M, input: *generálove rozhodnutie*, path: {0}, others: $\{1..N-1\} \setminus \{i\}$ }

Ak **je generál klamár**, tak jeho správy dôstojníkom sú takéto:

{level: M, input: úplný generálsky random, path: {0}, others: $\{1..N-1\} \setminus \{i\}$ }

Dôstojník D, ak kedykoľvek dostane správu {level, input, path, others }

- zapamätá si správu, čo dostal v strome podľa path
- ak level = 0, správa končí, neposiela sa ďalej
- ak level > 0, tak sa všetkým next \in others pošle nová správa
 - pre nexta {level-1, input, path+{D}, others\{next} }, **ak je D čestný**,
 - pre nexta {level-1, random, path+{D}, others\{next} }, **ak je D klamár**

Dump správ, N=4, M=1

(zapni si DEBUG=true)

```
0::<- (level=2, input=0: path [], others: [1 2 3]), forward to: [1 2 3]
■ 3::<- (level=1, input=0: path [0], others: [1 2]), forward to: [1 2]
  ■ 3::<- (level=0, input=1: path [0 2], others: [1]), final
  ■ 3::<- (level=0, input=0: path [0 1], others: [2]), final
■ 1::<- (level=1, input=0: path [0], others: [2 3]), forward to: [2 3]
  ■ 1::<- (level=0, input=0: path [0 3], others: [2]), final
  ■ 1::<- (level=0, input=1: path [0 2], others: [3]), final
■ 2::<- (level=1, input=1: path [0], others: [1 3]), forward to: [1 3]
  ■ 2::<- (level=0, input=0: path [0 3], others: [1]), final
  ■ 2::<- (level=0, input=0: path [0 1], others: [3]), final
```

general mal povodny rozkaz 0 ale je to klamar

agent 1 sa rozhodol 0

agent 2 sa rozhodol 0

agent 3 sa rozhodol 0

Disclaimer:

v skutočnosti správy môžu, aj prídu, v inom poradí, utriedené len kvôli názornosti

Dump správ, N=4, M=1

(zapni si DEBUG=true)

```
0::<- (level=2, input=1: path [], others: [1 2 3]), forward to: [1 2 3]
■ 3::<- (level=1, input=1: path [0], others: [1 2]), forward to: [1 2]
  ■ 3::<- (level=0, input=0: path [0 1], others: [2]), final
  ■ 3::<- (level=0, input=1: path [0 2], others: [1]), final
■ 2::<- (level=1, input=1: path [0], others: [1 3]), forward to: [1 3]
  ■ 2::<- (level=0, input=1: path [0 3], others: [1]), final
  ■ 2::<- (level=0, input=0: path [0 1], others: [3]), final
■ 1::<- (level=1, input=1: path [0], others: [2 3]), forward to: [2 3]
  ■ 1::<- (level=0, input=1: path [0 3], others: [2]), final
  ■ 1::<- (level=0, input=1: path [0 2], others: [3]), final
```

general mal povodny rozkaz 1

agent 1 je klamar

agent 2 sa rozhodol 1

agent 3 sa rozhodol 1

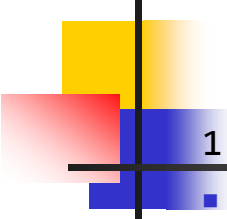
Disclaimer:

v skutočnosti správy môžu, aj prídu, v inom poradí, utriedené len kvôli názornosti

Správy pre N=7, M=2

!len správy, ktoré dostane len dôstojník 1!
aj tak je ich $26 = 5 + 5*4$

general mal rozkaz 0
agent 1 sa rozhodol 0
agent 2 sa rozhodol 0
agent 3 je klamar
agent 4 je klamar
agent 5 sa rozhodol 0
agent 6 sa rozhodol 0



```
1::<- (level=2, input=0: path [0], others: [2 3 4 5 6]), forward to: [2 3 4 5 6]
■ 1::<- (level=1, input=0: path [0 6], others: [2 3 4 5]), forward to: [2 3 4 5]
  ■ 1::<- (level=0, input=0: path [0 6 2], others: [3 4 5]), final
  ■ 1::<- (level=0, input=1: path [0 6 3], others: [2 4 5]), final
  ■ 1::<- (level=0, input=0: path [0 6 5], others: [2 3 4]), final
  ■ 1::<- (level=0, input=0: path [0 6 4], others: [2 3 5]), final
■ 1::<- (level=1, input=0: path [0 2], others: [3 4 5 6]), forward to: [3 4 5 6]
  ■ 1::<- (level=0, input=0: path [0 2 6], others: [3 4 5]), final
  ■ 1::<- (level=0, input=0: path [0 2 5], others: [3 4 6]), final
  ■ 1::<- (level=0, input=1: path [0 2 4], others: [3 5 6]), final
  ■ 1::<- (level=0, input=1: path [0 2 3], others: [4 5 6]), final
■ 1::<- (level=1, input=0: path [0 3], others: [2 4 5 6]), forward to: [2 4 5 6]
  ■ 1::<- (level=0, input=0: path [0 3 5], others: [2 4 6]), final
  ■ 1::<- (level=0, input=0: path [0 3 2], others: [4 5 6]), final
  ■ 1::<- (level=0, input=1: path [0 3 6], others: [2 4 5]), final
  ■ 1::<- (level=0, input=0: path [0 3 4], others: [2 5 6]), final
■ 1::<- (level=1, input=0: path [0 5], others: [2 3 4 6]), forward to: [2 3 4 6]
  ■ 1::<- (level=0, input=0: path [0 5 4], others: [2 3 6]), final
  ■ 1::<- (level=0, input=1: path [0 5 3], others: [2 4 6]), final
  ■ 1::<- (level=0, input=0: path [0 5 2], others: [3 4 6]), final
  ■ 1::<- (level=0, input=0: path [0 5 6], others: [2 3 4]), final
■ 1::<- (level=1, input=1: path [0 4], others: [2 3 5 6]), forward to: [2 3 5 6]
  ■ 1::<- (level=0, input=1: path [0 4 5], others: [2 3 6]), final
  ■ 1::<- (level=0, input=0: path [0 4 6], others: [2 3 5]), final
  ■ 1::<- (level=0, input=0: path [0 4 2], others: [3 5 6]), final
  ■ 1::<- (level=0, input=0: path [0 4 3], others: [2 5 6]), final
```



Agent

(Generál či dôstojník)

```
type Agent struct {  
    id int  
    cheating bool // true ak je klamár  
    channel chan Message // kanál, na ktorom počúva  
    children map[string] []Message // strom prijatých správ  
    received int // celkový počet prijatých správ  
}
```

children["0"] – 5 správ
children["06"] – 4 správy
children["02"] – 4 správy
...
received:26

```
1::<- (level=2, input=0: path [0], others: [2 3 4 5 6]), forward to: [2 3 4 5 6]  
■ 1::<- (level=1, input=0: path [0 6], others: [2 3 4 5]), forward to: [2 3 4 5]  
  ■ 1::<- (level=0, input=0: path [0 6 2], others: [3 4 5]), final  
  ■ 1::<- (level=0, input=1: path [0 6 3], others: [2 4 5]), final  
  ■ 1::<- (level=0, input=0: path [0 6 5], others: [2 3 4]), final  
  ■ 1::<- (level=0, input=0: path [0 6 4], others: [2 3 5]), final  
■ 1::<- (level=1, input=0: path [0 2], others: [3 4 5 6]), forward to: [3 4 5 6]  
  ■ 1::<- (level=0, input=0: path [0 2 6], others: [3 4 5]), final  
  ■ 1::<- (level=0, input=0: path [0 2 5], others: [3 4 6]), final  
  ■ 1::<- (level=0, input=1: path [0 2 4], others: [3 5 6]), final  
  ■ 1::<- (level=0, input=1: path [0 2 3], others: [4 5 6]), final  
■ 1::<- (level=1, input=0: path [0 3], others: [2 4 5 6]), forward to: [2 4 5 6]  
  ■ 1::<- (level=0, input=0: path [0 3 5], others: [2 4 6]), final  
  ■ 1::<- (level=0, input=0: path [0 3 2], others: [4 5 6]), final  
  ■ 1::<- (level=0, input=1: path [0 3 6], others: [2 4 5]), final  
  ■ 1::<- (level=0, input=0: path [0 3 4], others: [2 5 6]), final  
■ 1::<- (level=1, input=0: path [0 5], others: [2 3 4 6]), forward to: [2 3 4 6]  
  ■ 1::<- (level=0, input=0: path [0 5 4], others: [2 3 6]), final  
  ■ 1::<- (level=0, input=1: path [0 5 3], others: [2 4 6]), final  
  ■ 1::<- (level=0, input=0: path [0 5 2], others: [3 4 6]), final  
  ■ 1::<- (level=0, input=0: path [0 5 6], others: [2 3 4]), final  
■ 1::<- (level=1, input=1: path [0 4], others: [2 3 5 6]), forward to: [2 3 5 6]  
  ■ 1::<- (level=0, input=1: path [0 4 5], others: [2 3 6]), final  
  ■ 1::<- (level=0, input=0: path [0 4 6], others: [2 3 5]), final  
  ■ 1::<- (level=0, input=0: path [0 4 2], others: [3 5 6]), final  
  ■ 1::<- (level=0, input=0: path [0 4 3], others: [2 5 6]), final
```



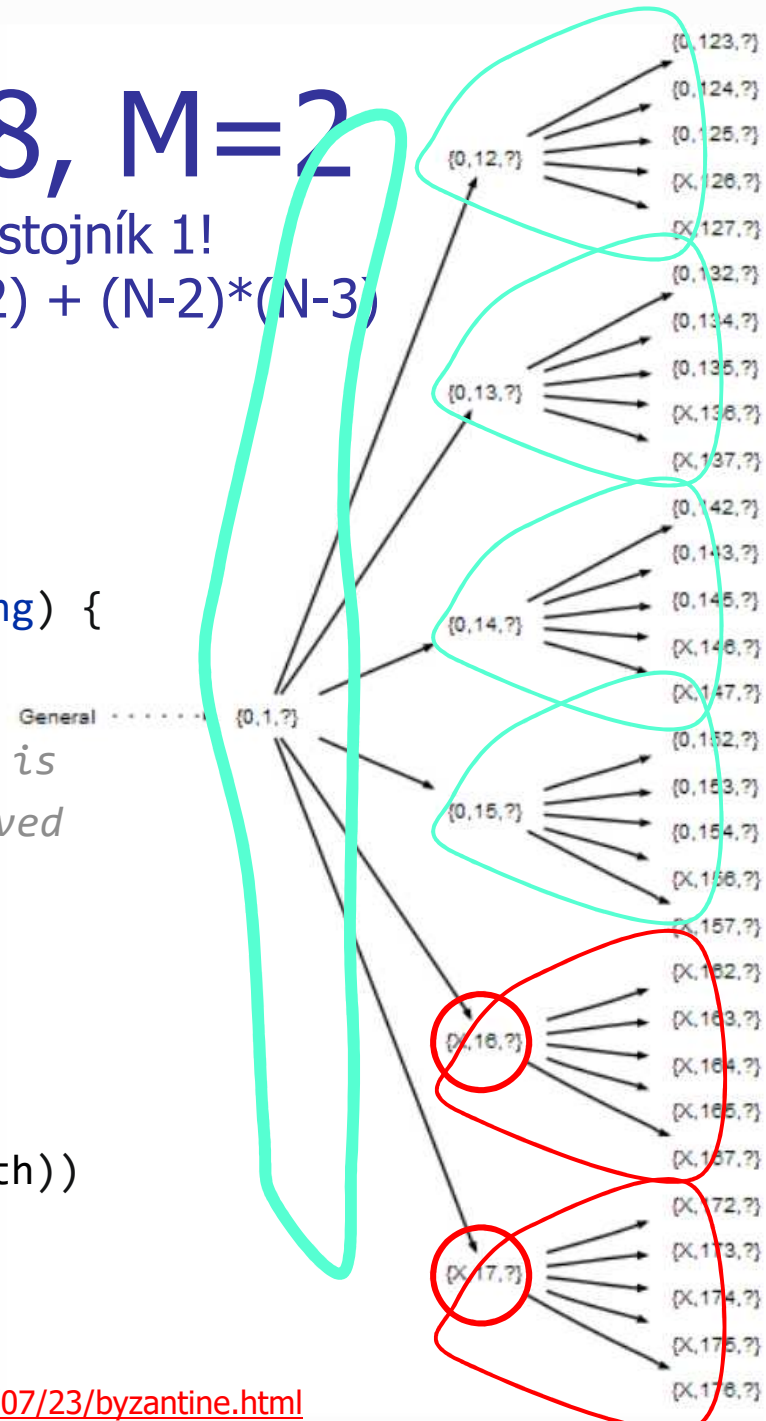
Šírenie správ

```
func (agent *Agent)run() {           // agent lifecycle
    go func() {
        for {
            msg := <-agent.channel // prisla message
            if msg.level == 0 {
                fmt.Printf("%v:<- %v, final\n", agent.id, msg.toString())
            } else {
                fmt.Printf("%v:<- %v, forward to: %v \n", agent.id, msg.toString(), msg.others)
                for indx, next := range msg.others {
                    others1 := append(append([]int{},msg.others[:indx]...), msg.others[(indx+1):]...)
                    newInput := msg.input
                    if agent.cheating {
                        newInput = int(rand.Intn(2))
                    }
                    agents[next].channel <- Message{msg.level-1, newInput, append(msg.path, agent.id), others1}
                }
            }
            if len(msg.path) > 0 {
                msgkey := getKey(msg.path[:len(msg.path)-1])
                agent.children[msgkey] = append(agent.children[msgkey], msg)
            }
        }
    }()
}
```

aj tak je ich $36 = 6 + 6*5 = (N-2) + (N-2)*(N-3)$

```
var TABS = "\t\t\t\t\t\t\t\t\t\t\t\t"
```

```
func (agent *Agent) traverse(path string) {  
    messages := agent.children[path]  
    for index, msg := range messages { General  
        if index > 0 { // first message is  
            // the original message received  
            // by agent, skip it  
            fmt.Printf("%v %v:%v\n",  
                TABS[:len(path)],  
                getKey(msg.path),  
                msg)  
            agent.traverse(getKey(msg.path))  
        }  
    }  
}
```





Resume

Algorithm OM(0) The general sends his value to every lieutenant. Each lieutenant uses the value he receives from the general.

Algorithm OM(m), $m > 0$ The general sends his value to each lieutenant.

- For each i , let v_i be the value lieutenant i receives from the general. Lieutenant i acts as the general in Algorithm OM($m-1$) to send the value v_i to each of the $n-2$ other lieutenants.
- For each i , and each $j \neq i$, let v_i be the value lieutenant i received from lieutenant j in step 2 (using Algorithm ($m-1$)). Lieutenant i uses the value majority (v_1, v_2, \dots, v_n). -- Lamport's Algo

- Od 1982 existuje pôvodný článok <https://people.eecs.berkeley.edu/~luca/cs174/byzantine.pdf>
- (nie len) ja som mu nerozumel
- jediné, čo je jasné, že $N > 3 \cdot M$, resp. čestných je viac ako $2 \cdot \text{kľamárov}$
- ilustrácie a youtuby častia končia pri $M=1$, ergo $N=4$
- ak máme už dvoch kľamárov, Fault Tolerant Systém musí mať aspoň $N=7$
- a tam je už 156 správ, čo sa zle kreslí, vysvetľuje, simuluje aj chápe...
- Našiel som článok, ktorý to vysvetľuje programátorovi <https://marknelson.us/posts/2007/07/23/byzantine.html>
- obsahuje <500 riadkov C++ funkčného bohato-komentovaného kódu
- kód je sekvenčná simulácia v C++, žiaden náznak distribuovanosti, vlákna
- po vyše týždni som to furt nechápal
- konzultácia mi nepomohla
- zobral som to ako personal challenge,
- môj kód ma 200r, používa gorutiny, je distribuovaný
- a okrem toho..., súvisí to s Blockchainom <https://www.binance.vision/blockchain/byzantine-fault-tolerance-explained>

Our generals

(epilog)



Niektorí

- ste to zažili, a dáte/nedáte (mi) za pravdu ?
 - štátnice odhalia...
- to ešte len zažijete,
 - CSP (Communicating Sequential Processes, T.Hoare, 1978) je background za GO
 - programujte si to, len tak začnete mať pocit, že možno trochu rozumiete...
- ešte nevieme, kam pôjdete na magistra, ...

Motto: Čo nenaprogramuješ,
tomu nerozumieš.
... a niekedy ani po tom ...

Keď už programujete, tak programujte

- aby ste elegantne/efektívne vyriešili nejaký problém
- nie aby ste vedeli o nejaký jazyk naviac

Konkurentné programovanie je ťažké, lebo:

- nemáme prax, učia nás rozmýšľať sekvenčne...
- zle sa to ladí, dvakrát spustím, a dopadne to inak...
- chýba šikovná vizualizácia toho, čo sa deje...
- ak aj niečo vyzerá distribuovane, ešte to možno zdieľa spoločný priestor



ak všetky go rutiny prístupujú do globálnej premennej, tak to zle dopadne

`MUTEX.Lock()` `Counter++` `MUTEX.Unlock()`



Go mantra

“
Do not communicate by
sharing memory; instead,
share memory by communicating.

— Effective Go

”