recursive algorithm

dir= 1 or -1
dir=-dir

```
algorithm new_name(x, first, dir):
        % x (n ≥ x ≥ 1) is the recursion parameter %
(01)  SM[x, first, dir][i] ← old_name_i;
(02)  competing_i ← SM[x, first, dir].collect();
(03)  if |competing_i| = x
(04)     then last ← first + dir(2x − 2);
(05)          if old_name_i = max(competing_i)
(06)             then res_i ← last
(07)             else  res_i ← new_name(x − 1, last + dir̄, dir̄)
(08)          end if
(09)     else  res_i ← new_name(x − 1, first, dir)
(10)  end if;
(11)  return(res_i).
```

new_name(n,1,1)

- For a pair (first, dir); the Algo ensures at most x processes invoke new_name(x, first, dir)
- These processes compete for new names in space name of size 2x-1
- Initial space New_name (n,1,1)—> 1, 2n-1

- if k processes participate in the renaming, their main call new_name(n, 1, 1) will systematically entail the call new_name(n – 1, 1, 1), etc., until the call new_name(k, 1, 1).

- If k processes participate to new_name(k, 1, 1) at least one (the last process that writes) find k processes in competing

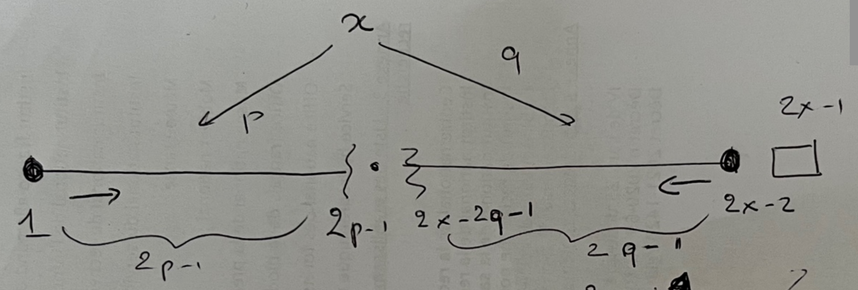
- In space 1..X, (dir=1) if k processes compete they get a name in 1..2k-1

- In space 1..X, (dir=-1) if k processes compete they get a name in X-(2k-2)..X

- **Splitter behavior** associated with the array of atomic registers SM [x, first, dir] is defined by the following properties: (let $x' = x - 1$):
  - At most $x' = x-1$ processes invoke new_name(x–1, first, dir) (line 9). Hence, these processes will obtain new names in an interval of size $(2x' - 1)$ as follows:
    - If dir = 1, the new names will be in the "going up" interval [first..first + (2x'– 2)],
    - if dir = -1, the new names will be in the "going down" interval [first – (2x'– 2)..first].
  - At most $x' = x - 1$ processes invoke new_name(x – 1, last + $\overline{dir}$, $\overline{dir}$) (line 7), where last = first + dir(2x – 2) (line 7). Hence, these $x' = x - 1$ processes will obtain their new names in a renaming space of size $(2x' - 1)$ starting at last + 1 and going from left to right if $\overline{dir} = 1$, or starting at last – 1 and going from right to left if $\overline{dir} = -1$.

- At most one process “stops”, i.e., defines its new name as last = first + dir(2x – 2) (lines 4,5,6). Let us observe that the only process that can stop is the one such that id has the greatest value in the array SM [x, first, dir] [1..n] which contains then exactly x identities.

$p \neq 0 \quad q \neq 0$



$x = p + q + 1$

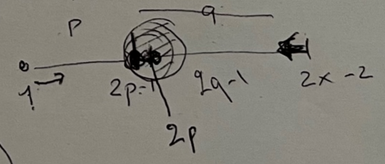
$2x - 2q - 1 = 2p + 1$

$x = p + q$

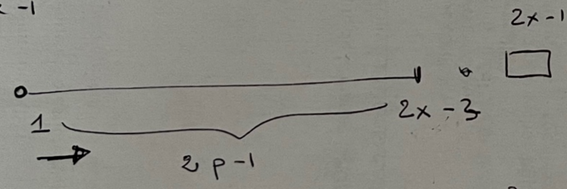
$1 \ldots 2p-1$

$2x-2q = 2p$

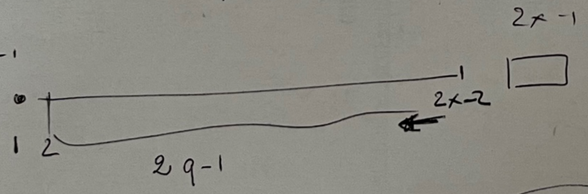
$2x-2$

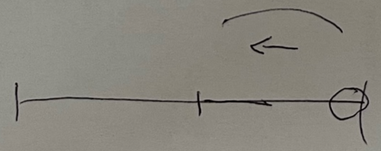


$p = x - 1$



$q = x - 1$





- With objects cons for 2 processes we can achieve strong adaptive renaming
- (strong = space of new name are in 1..n)
- Adaptive : if k processes compete they gets a name in 1..k

- With strong adaptive renaming we can achieve consensus for 2 processes

Byzantine failures

- A correct process is a process that follow its code
- A byzantine process might fail by exhibiting arbitrary behavior (e.g send bogus message)
- (but it can’t pretend to be another process)(or no message)

- With at most t crash failures , in synchronous system we can solve consensus
- This is not true with Byzantine failure
- First define « byzantine agreement » : the previous definition of validity « the decided value is a value proposed by some process » doesn’t work

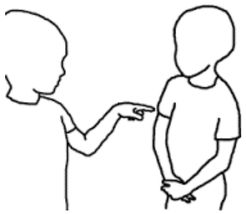
Byzantine agreement

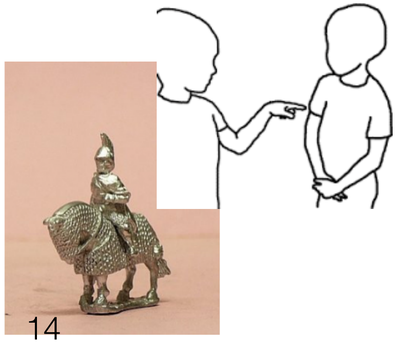
- **Agreement**: No two correct processes decide on different values
- **Validity** : if all correct processes propose the same value v then v is the only possible decision for correct processes
- **Termination** : All correct processes eventually decide

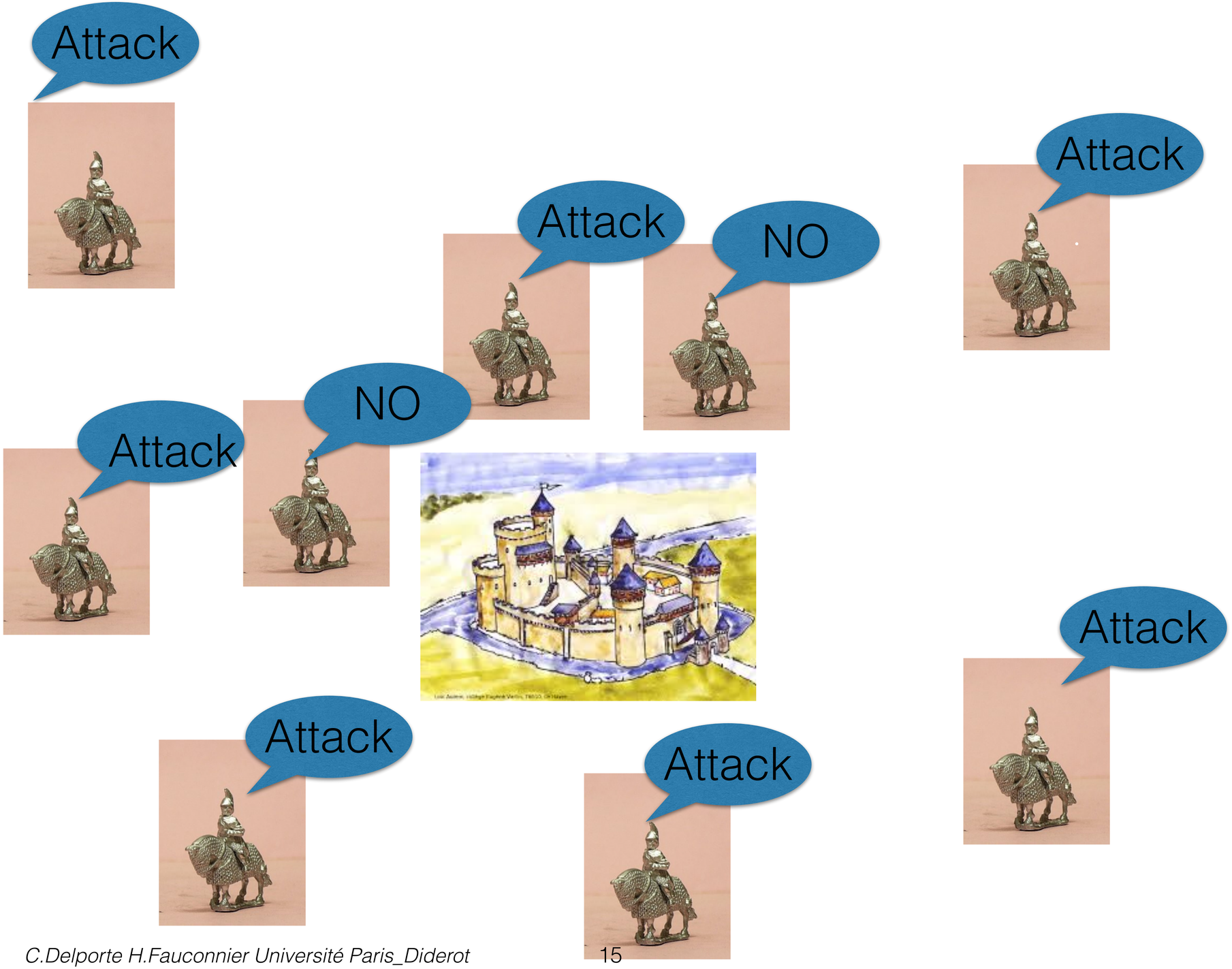




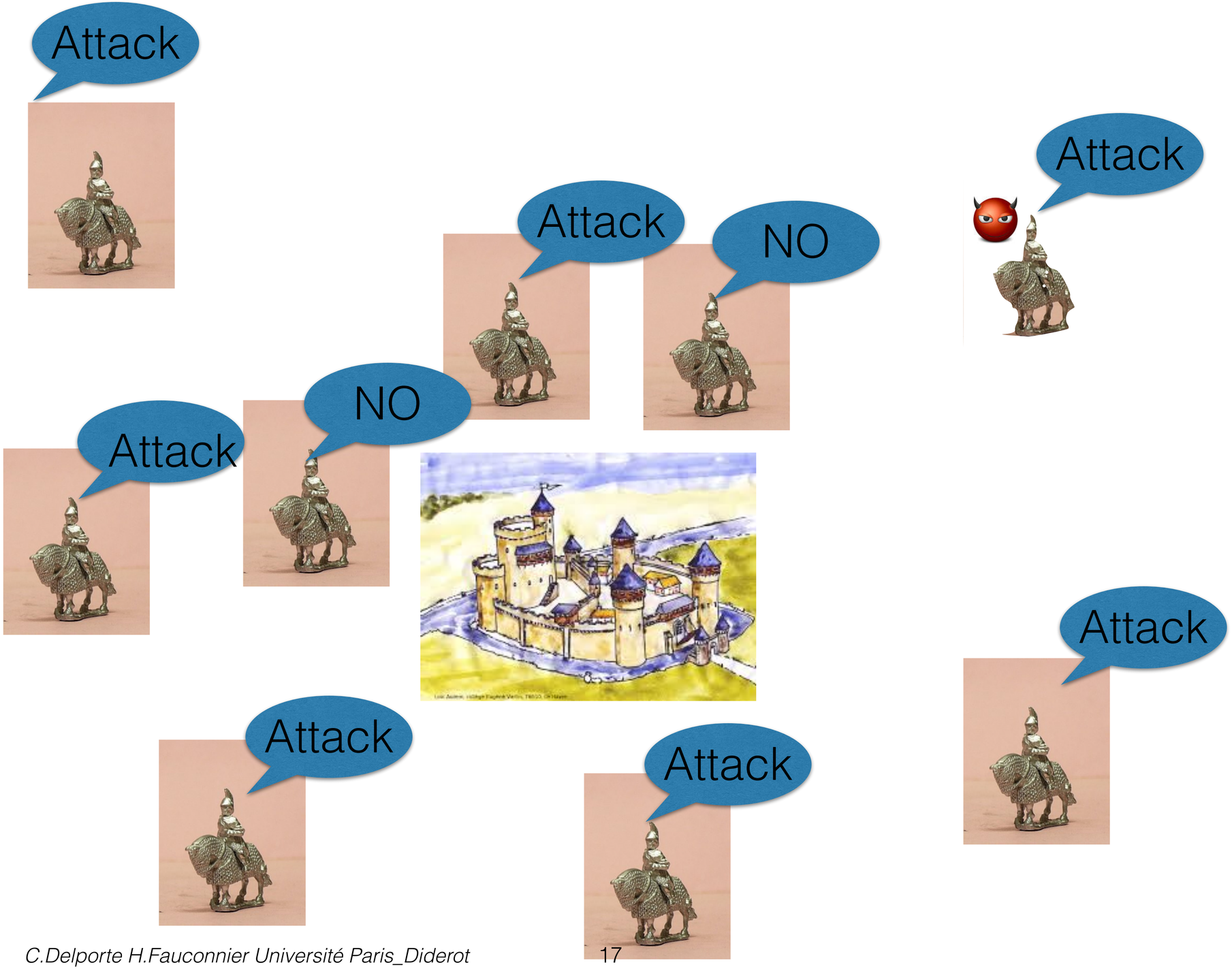


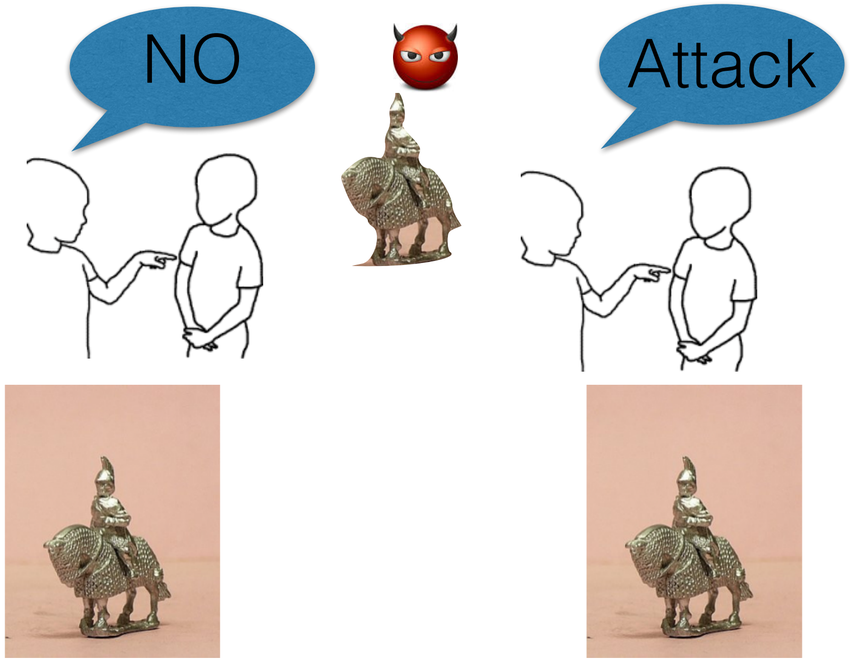



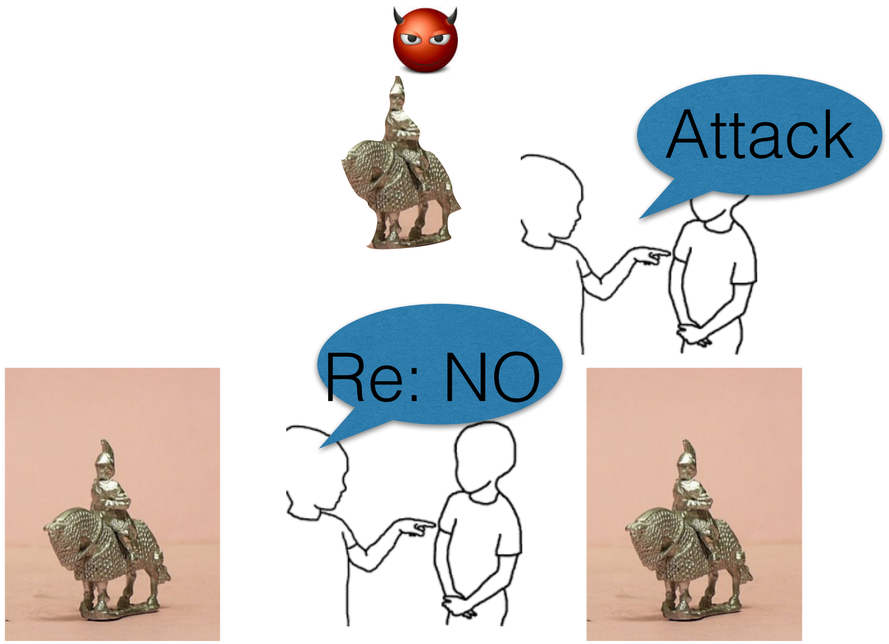



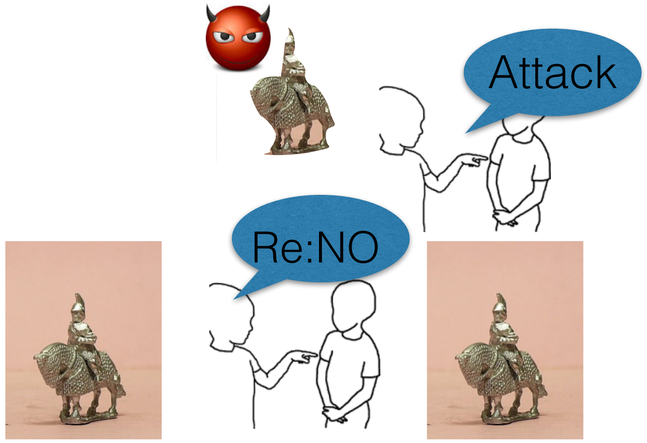

Attack
Attack
NO
Attack
NO
Attack
Attack
Attack
Attack

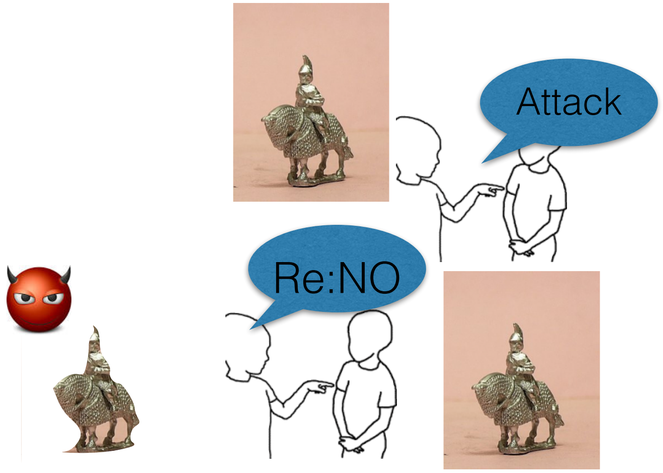

Attack
Attack
NO
Attack
NO
Attack
Attack
Attack
Attack


Attack
Attack
NO
Attack
NO
Attack
Attack
Attack
Attack


NO
Attack

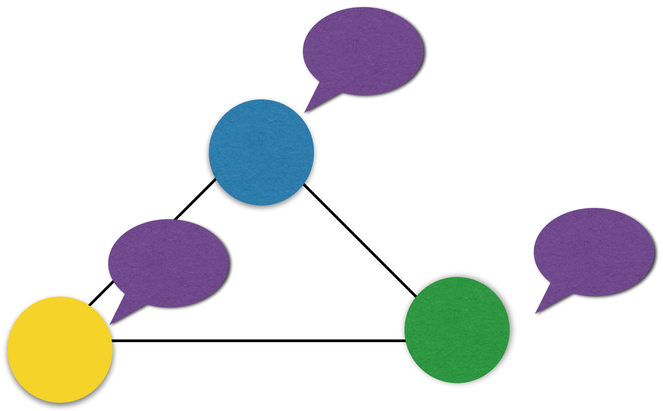

Attack
Re: NO


Attack
Re:NO

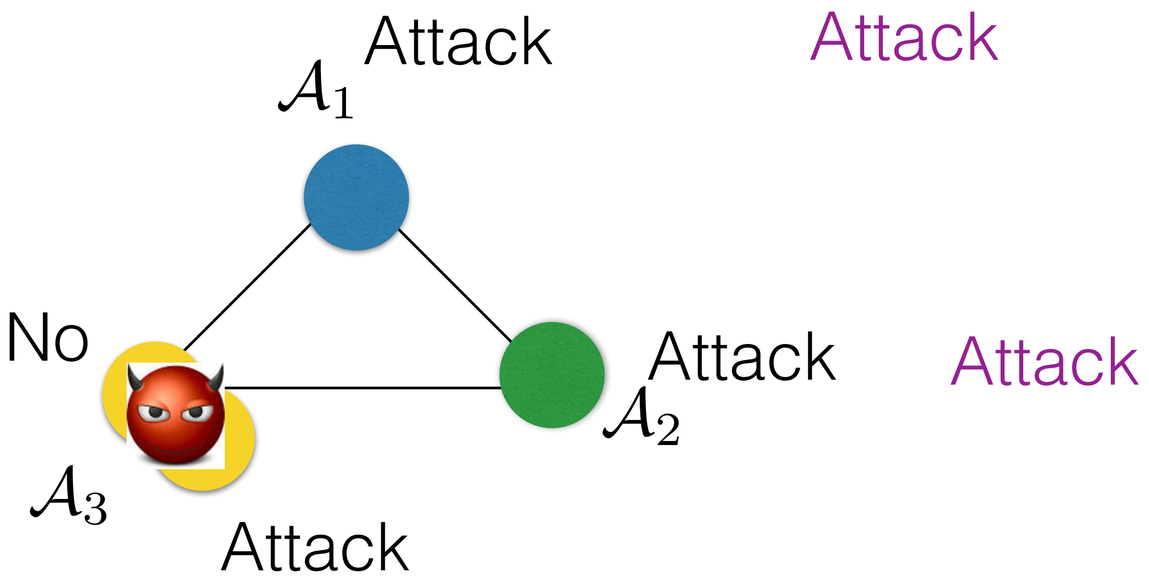

Attack
Re:NO

Impossibility result

- Theorem: If $n \leq 3t$ then it is impossible to solve byzantine agreement
- proof n=3 t =1

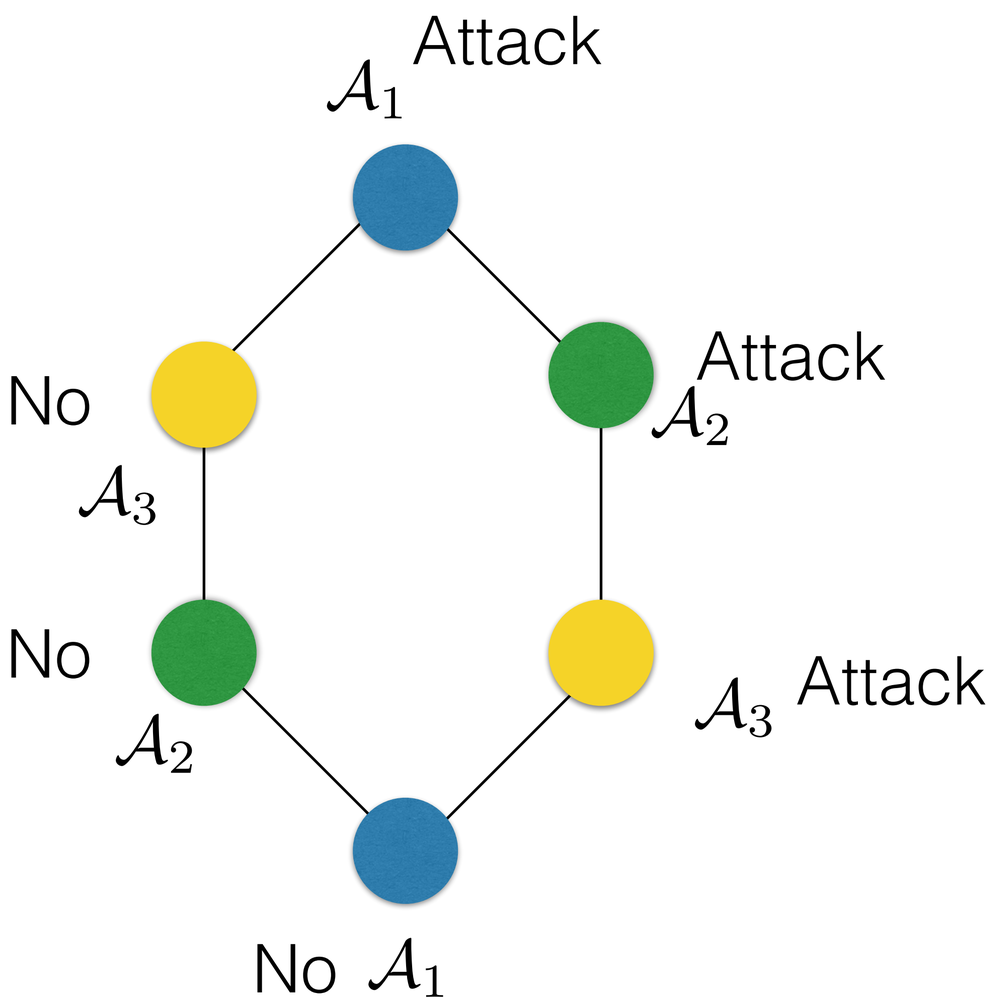


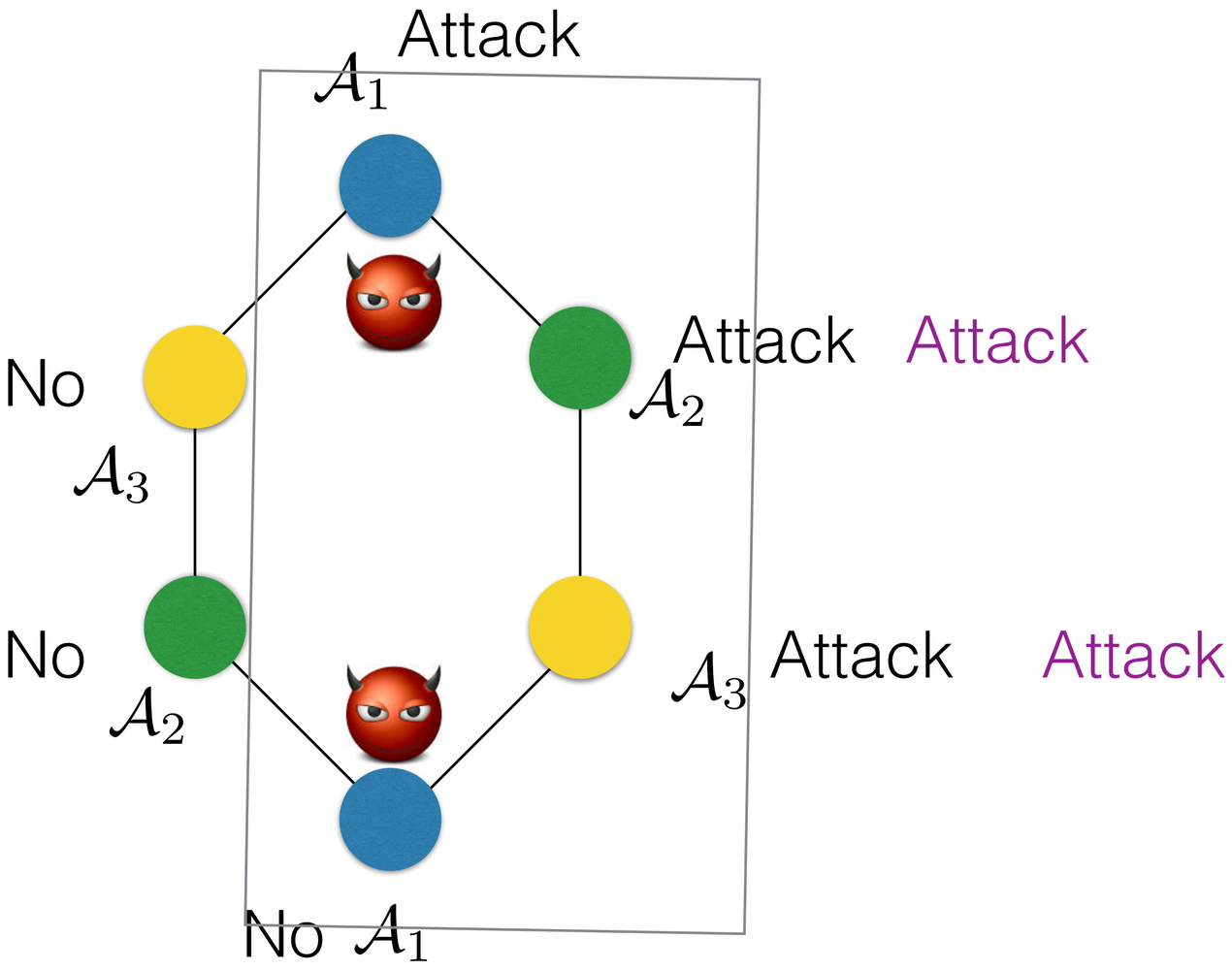
Decide

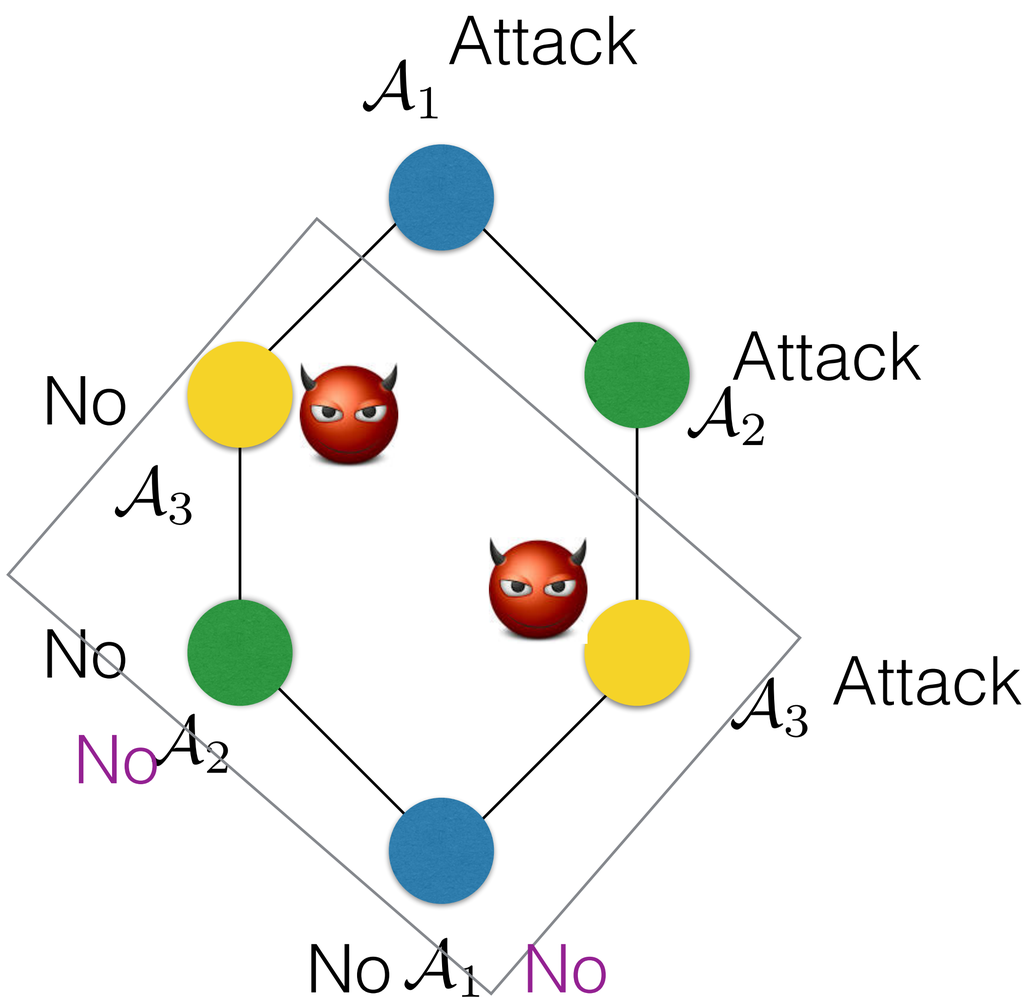


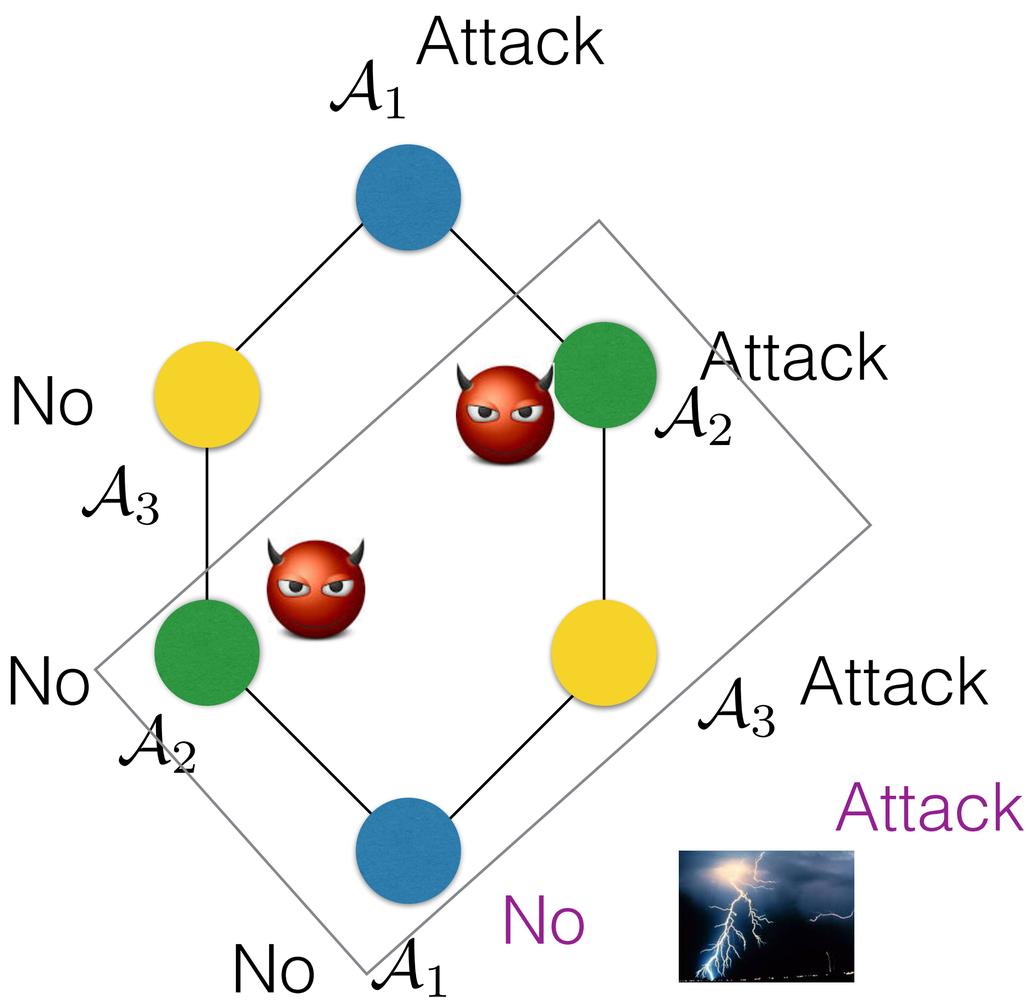
Attack

Attack


Attack
$\mathcal{A}_1$
No
$\mathcal{A}_3$
Attack
$\mathcal{A}_2$
No
$\mathcal{A}_2$
$\mathcal{A}_3$ Attack
No $\mathcal{A}_1$

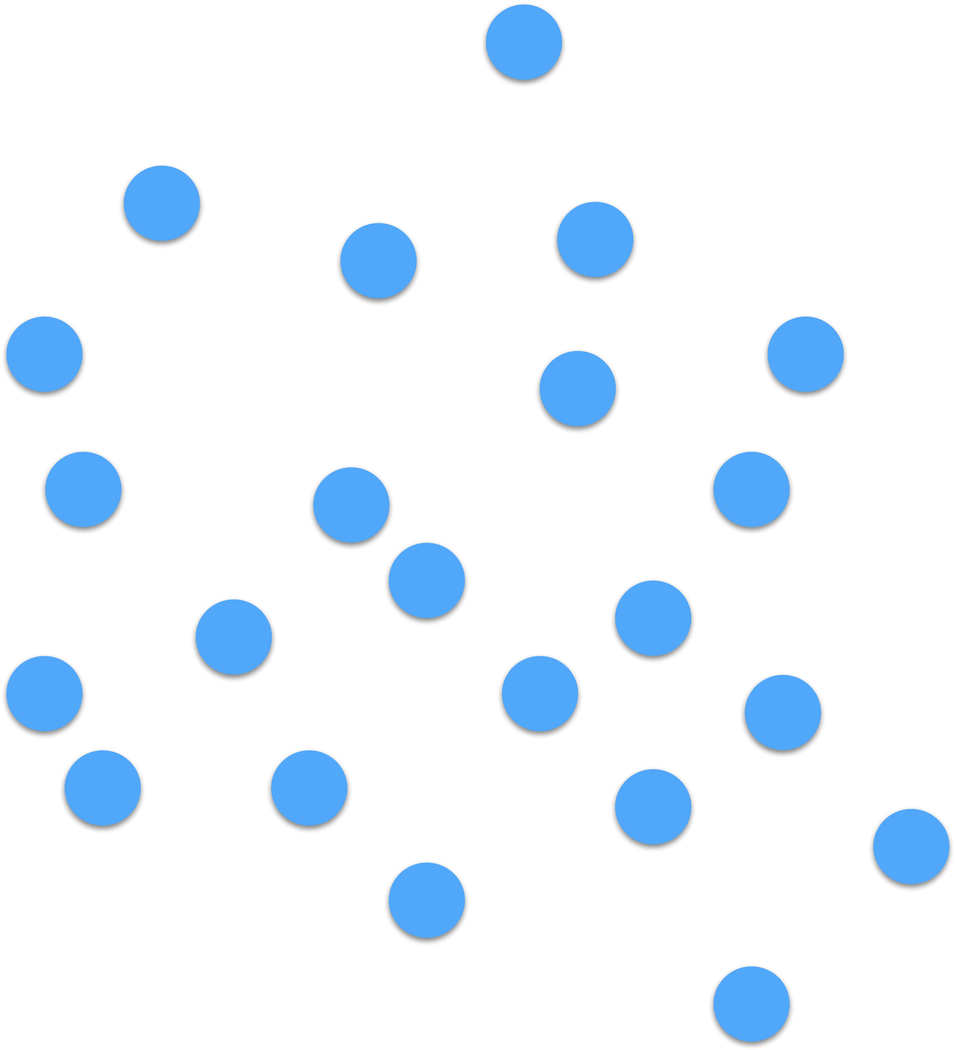

Attack
$\mathcal{A}_1$
No
$\mathcal{A}_3$
Attack
$\mathcal{A}_2$
Attack
No
$\mathcal{A}_2$
$\mathcal{A}_3$
Attack
Attack
No
$\mathcal{A}_1$


Attack
$\mathcal{A}_1$
Attack
$\mathcal{A}_2$
No
$\mathcal{A}_3$
No
No
$\mathcal{A}_2$
Attack
$\mathcal{A}_3$
No
$\mathcal{A}_1$
No

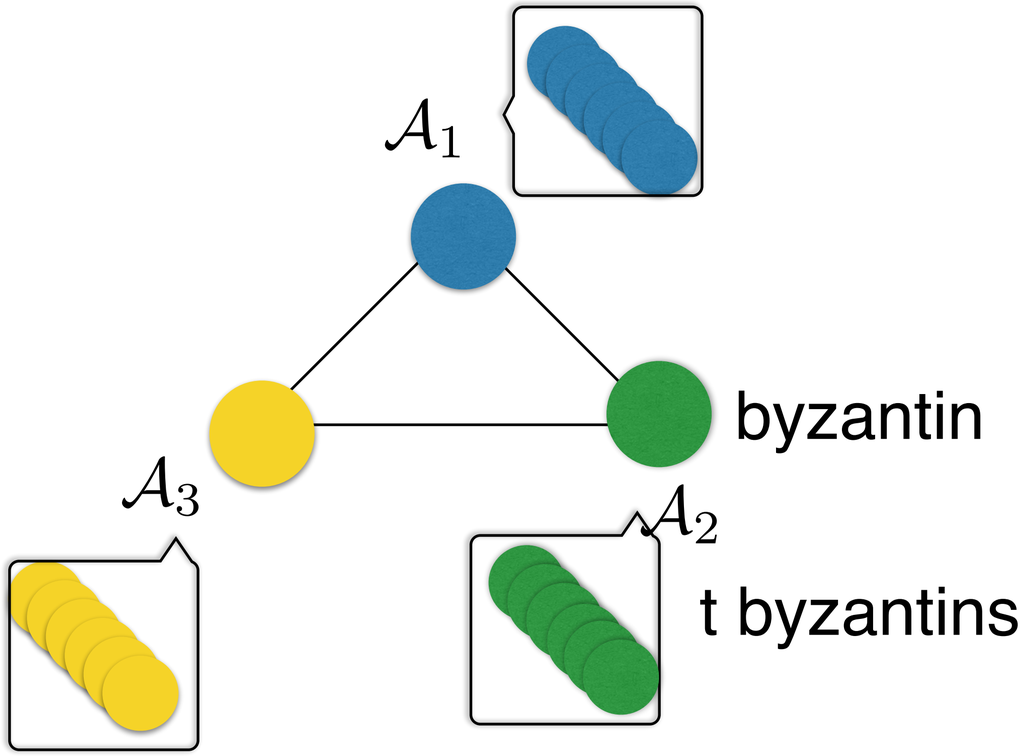

Attack
$\mathcal{A}_1$
No
$\mathcal{A}_3$
Attack
$\mathcal{A}_2$
No
$\mathcal{A}_2$
$\mathcal{A}_3$ Attack
Attack
No
No $\mathcal{A}_1$

- The proof may be easily extends to any value of n

$n \geq 3t$



$n \geq 3t$


$\mathcal{A}_1$
byzantin
$\mathcal{A}_3$
$\mathcal{A}_2$
t byzantins