

Fast Randomized Test-and-Set and Renaming

Dan Alistarh, Hagit Attiya, Seth Gilbert,
Andrei Giurgiu, and Rachid Guerraoui



The Problem: Renaming

- n identical processes
- each should get an unique identity



- renaming can be either **tight** (1 to n) or loose
- asynchronous system with crash failures

What was known

- Deterministic wait-free renaming is *impossible* in a namespace smaller than $(2n - 1)$ [HS, CR]
- Many deterministic solutions for namespace of size $\geq 2n - 1$ [AM, AM2, AF, CK, etc.]
- Two randomized solutions
 - Panconesi et al. [ISAAC92]: $(1+\epsilon)$ -renaming
 - Eberly et al. [DISC98]: **tight** renaming in $O(n^3)$ steps

Our results

Algorithm	Problem	Step Complexity
ReShuffle	tight renaming	total $O(n \log^2 n)$

Our results

Algorithm	Problem	Step Complexity
ReShuffle	tight renaming	total $O(n \log^2 n)$
AdaptiveSearch	adaptive $(1+\varepsilon)$ -renaming	total $O(k \log^4 k)$ local $O(\log^4 k)$

Our results

Algorithm	Problem	Step Complexity
ReShuffle	tight renaming	total $O(n \log^2 n)$
AdaptiveSearch	adaptive $(1+\epsilon)$ -renaming	total $O(k \log^4 k)$ local $O(\log^4 k)$
RatRace	adaptive test-and-set	total $O(k \log^2 k)$ local $O(\log^2 k)$

Our results

- ReShuffle: the first tight renaming algorithm with step complexity $O(n \text{ polylog } n)$
- AdaptiveSearch: the first adaptive (loose) renaming algorithm; has step complexity $O(k \text{ polylog } k)$
- RatRace: adaptive test-and-set with local step complexity $O(\text{polylog } k)$

Outline

- Problem
- Related Work
- Results
- **Our Approach**
- **RatRace: Adaptive Test-and-Set**
- **ReShuffle: Tight Randomized Renaming**
- **Open questions**

The idea



What if processes simply choose names
at random?

It would work (whp) if the names are chosen
from a large namespace (e.g., $1 \dots n^3$)

The idea



What if processes choose names
at random from 1 to n , and *try again* if they fail?

- Two problems:
- How does a process know it *won* a name?
 - Is this *efficient*?

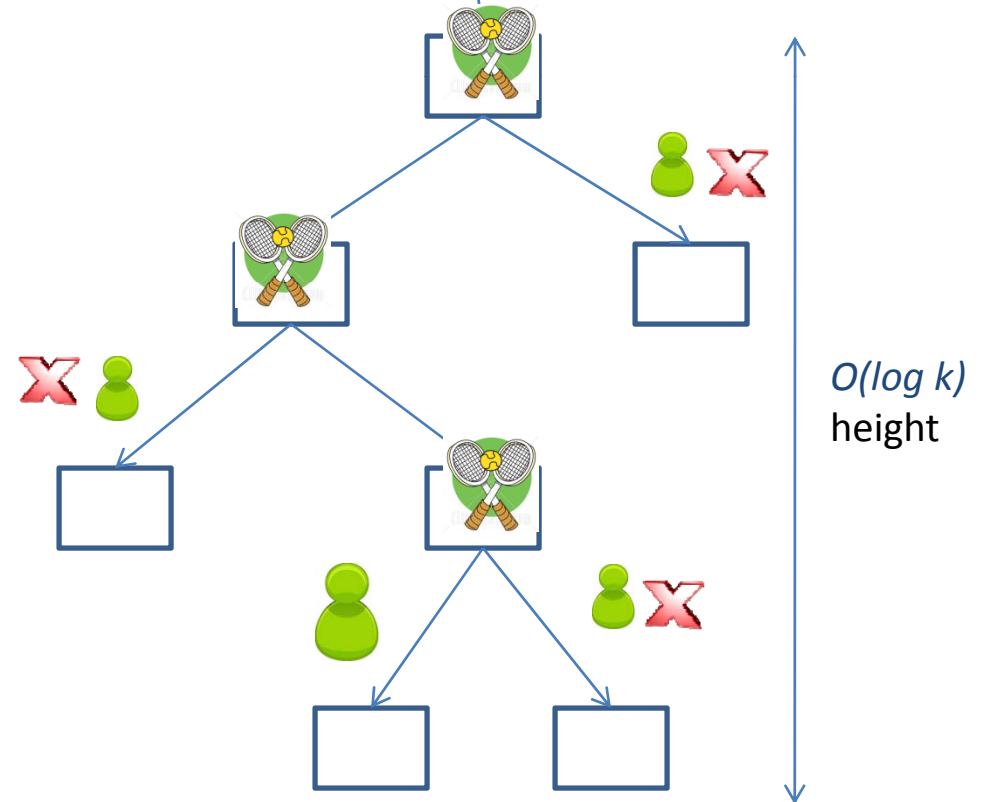
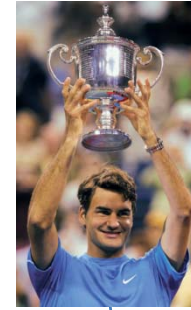
RatRace: getting a name

- Randomized Test-and-Set or...

procedure TryName (name i)

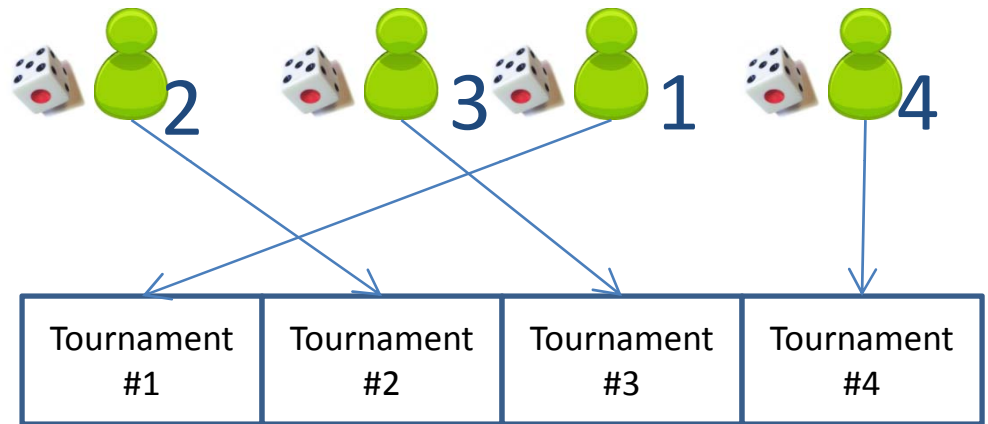
1. Get a place on the “tournament draw”
2. Win your way to the top
3. If you lost, leave

Lemma. RatRace implements randomized Test-and-Set, with local step complexity $O(\log^2 k)$



ReShuffle: the algorithm

```
n processes, n tournaments  
procedure GetName ()  
while( true )  
  1. compete in a new  
    random tournament from 1 to n  
  2. if winner,  
    return tournament's number
```



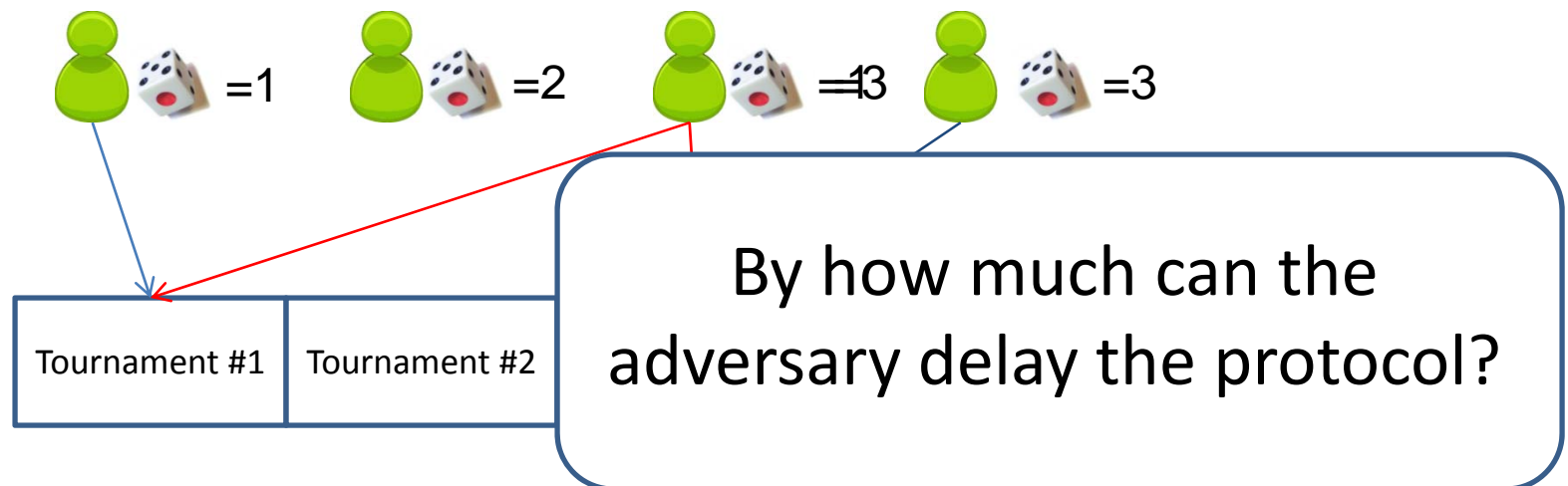
Why this works:

- A process will win at most one tournament
- After n tries, a process has competed in all the tournaments, so it has to win one (or crash)

The challenge



- Strong adaptive adversary
- Controls scheduling and crash failures
- Can read the random choices the processes make, and decide accordingly



The claim

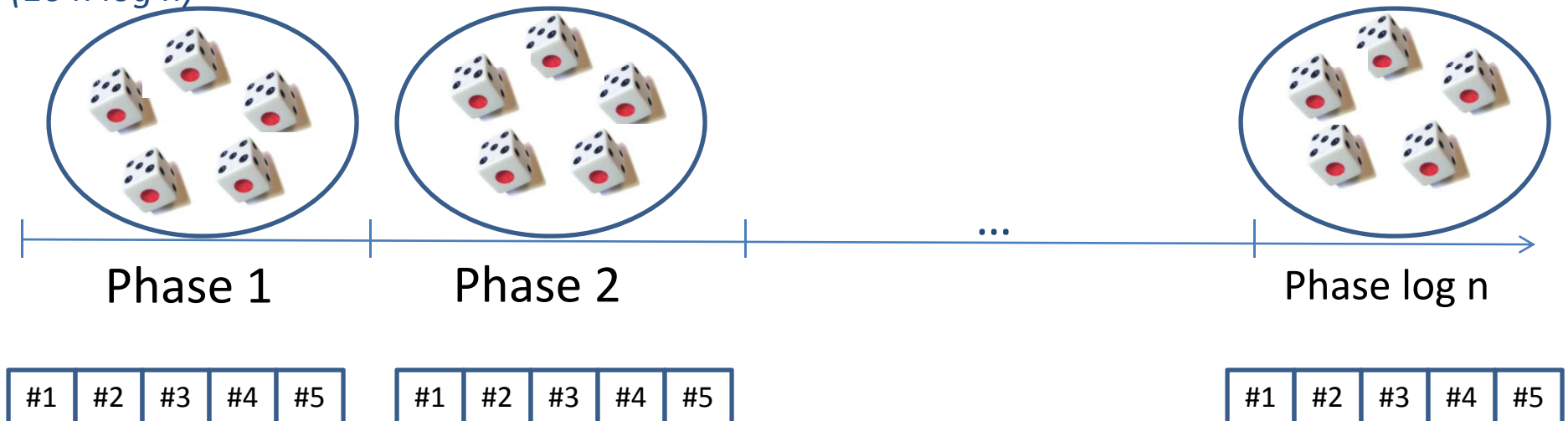
Theorem. After the adversary schedules $O(n \log^2 n)$ total process choices, each Tournament object has been accessed at least once.

- Once this occurs, ReShuffle will terminate after $O(n \log^2 n)$ extra steps.

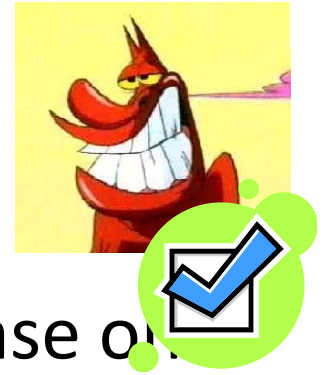
A proof sketch

- We split the execution into *phases*: each phase contains $(10 n \log n)$ Tournament accesses
- We prove that, during a phase, the number of free (non-accessed) Tournaments is *halved*, whp!

$(10 n \log n)$



Proof sketch: phase one



- The adversary sees at most $(10 n \log n + n)$ process choices during phase one.
- Out of these, it has to schedule $(10 n \log n)$!



The tournament vector

The basic idea:

The adversary can re-order processes, but it cannot influence their random choices!

The result

Theorem. ReShuffle solves strong renaming ensuring

- termination with probability 1
- name uniqueness
- total step complexity $O(n \log n \log^4 \log n)$

- Within log factors from the trivial lower bound of $\Omega(n)$

Result #2: AdaptiveSearch

- An **adaptive** renaming algorithm
(processes do not know n)
- Renames into a namespace of size $(1 + \epsilon) k$
- Total step complexity $O(k \log^4 k)$ whp.
- Local step complexity $O(\text{polylog } k)$ whp.

Open questions

- *Locally-efficient* tight renaming
- Tight *and* adaptive?
- Lower bound
- Applications to scheduling problems

Last slide

We can do efficient asynchronous test-and-set and tight renaming using randomization.

There exists an efficient adaptive solution as well, that does not guarantee a tight namespace.

Tight and adaptive? Lower bound?

Backup