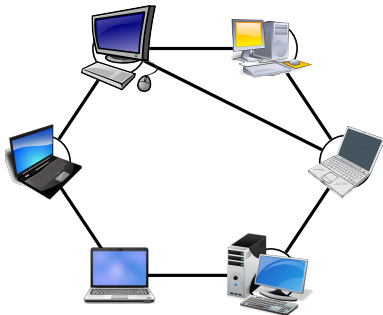


# Randomness vs. Time in Anonymous Networks

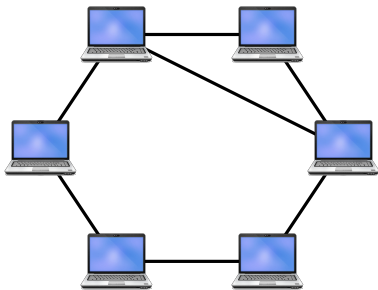


*Jochen Seidel   Jara Uitto   Roger Wattenhofer*

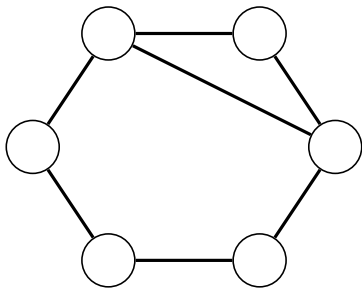
# Anonymous Networks



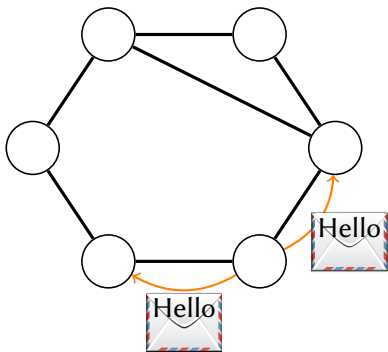
# Anonymous Networks



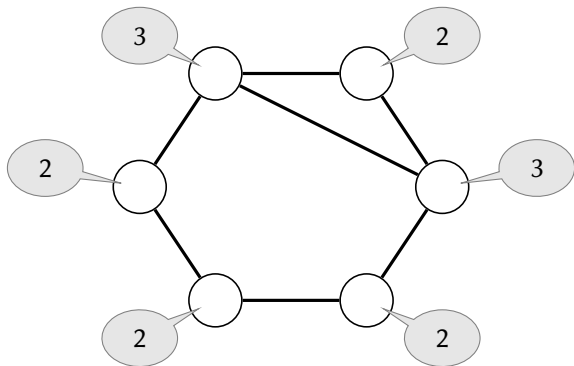
# Anonymous Networks



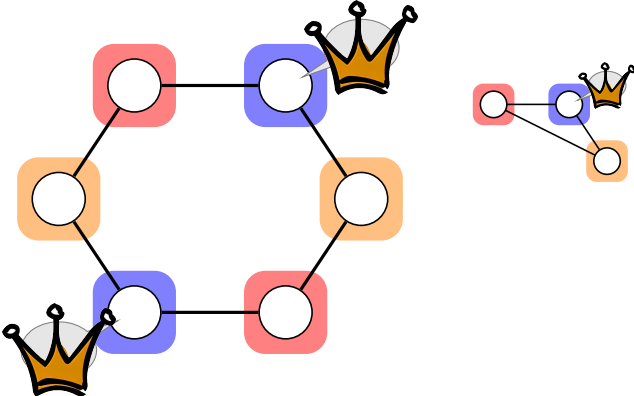
# Anonymous Networks



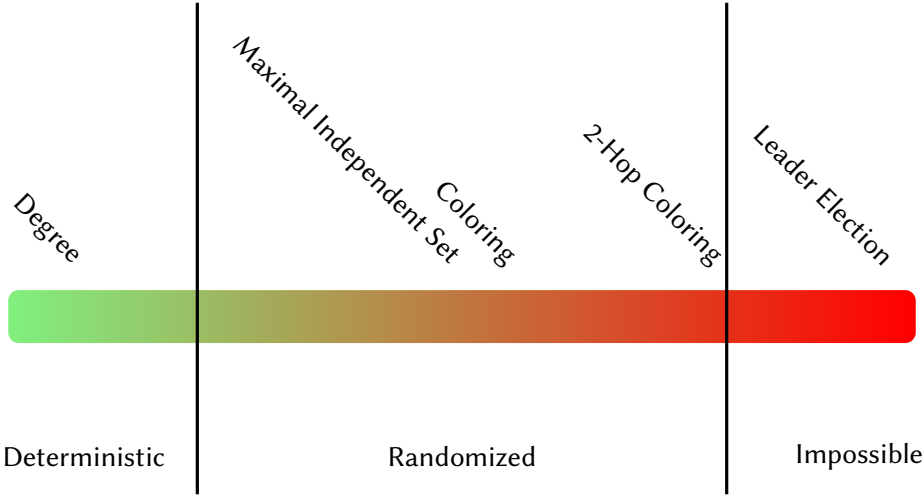
# Anonymous Networks



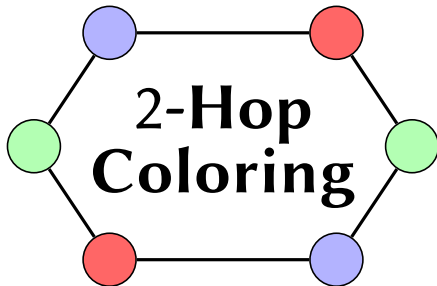
# Leader Election



# Computability







# Theorem

[PODC 2014]

```
mersenne_twister_engine<UIntType,w,n,m,r,a,u,d,s,b,t,c,l,f>::twist()
{
    const UIntType upper_mask = (-static_cast<UIntType>(0)) << r;
    const UIntType lower_mask = ~upper_mask;

    const std::size_t unroll_factor = 6;
    const std::size_t unroll_extra = (n-m) % unroll_factor;
    const std::size_t unroll_extra2 = (m-1) % unroll_factor;

    // split loop to avoid costly modulo operations
    // extra scope for MSVC brokenness w.r.t. for scope
    for(std::size_t j = 0; j < n-m-unroll_extra; j++) {
        for(UIntType y = (x[j] & upper_mask) | (x[j+1] & lower_mask);
            UIntType y = (x[j] & upper_mask) | (x[j+1] & lower_mask);
            x[j] = x[j+m] ^ (y >> 1) ^ ((x[j+1]&l) * a);
        }

        for(std::size_t j = n-unroll_extra; j < n-m; j++) {
            for(UIntType y = (x[j] & upper_mask) | (x[j+1] & lower_mask);
                UIntType y = (x[j] & upper_mask) | (x[j+1] & lower_mask);
                x[j] = x[j+m] ^ (y >> 1) ^ ((x[j+1]&l) * a);
            }
        }

        for(std::size_t j = n-1-unroll_extra2; j < n-1; j++) {
            for(UIntType y = (x[j] & upper_mask) | (x[j+1] & lower_mask);
                UIntType y = (x[j] & upper_mask) | (x[j+1] & lower_mask);
                x[j] = x[j-(n-m)] ^ (y >> 1) ^ ((x[j+1]&l) * a);
            }
        }

        // last iteration
        UIntType y = (x[n-1] & upper_mask) | (x[0] & lower_mask);
        x[n-1] = x[m-1] ^ (y >> 1) ^ ((x[0]&l) * a);
        i = 0;
    }
}

template<class UIntType,
        std::size_t m, std::size_t n, std::size_t w, std::size_t r,
        UIntType a, std::size_t u, UIntType d, std::size_t s,

```

# Randomized Algorithm

=



+

# Deterministic Algorithm

```
#include <string>
int main(void)
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```





# Theorem

[PODC 2014]

```
mersenne_twister_engine<UIntType,w,n,m,r,a,u,d,s,b,t,c,l,f>::twist()
{
    const UIntType upper_mask = (-static_cast<UIntType>(0)) << r;
    const UIntType lower_mask = ~upper_mask;

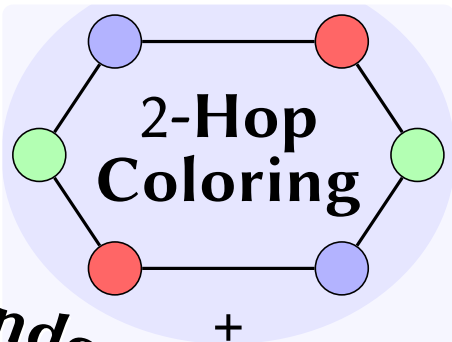
    const std::size_t unroll_factor = 6;
    const std::size_t unroll_extra = (n-m) % unroll_factor;
    const std::size_t unroll_extra2 = (m-1) % unroll_factor;

    // split loop to avoid costly modulo operations
    // extra scope for MSVC brokenness w.r.t. for scope
    for(std::size_t i = 0; i < n-m-unroll_factor; i++) {
        for(std::size_t j = 0; j < n-m-unroll_extra; j++) {
            UIntType x[i] = 5 * upper_mask | (x[j+1] & lower_mask);
            x[j] = x[i] >> 1 - ((x[j+1]&1) * a);
        }
        for(std::size_t j = 0; j < m; j++) {
            UIntType y = (x[j] & upper_mask) | (x[j+1] & lower_mask);
            x[j] = x[j] >> 1 - ((x[j]&1) * a);
        }
        for(std::size_t j = n-1-unroll_extra2; j < n-1; j++) {
            UIntType y = (x[j] & upper_mask) | (x[j+1] & lower_mask);
            x[j] = x[j] >> 1 - ((x[j]&1) * a);
        }
        // last iteration
        UIntType y = (x[n-1] & upper_mask) | (x[0] & lower_mask);
        x[n-1] = x[n-1] >> 1 - ((x[0]&1) * a);
        i = 0;
    }
}

template<class UIntType,
        std::size_t w, std::size_t n, std::size_t m, std::size_t r,
        UIntType a, std::size_t u, UIntType d, std::size_t s,
        std::size_t b, std::size_t t, std::size_t c, std::size_t l, std::size_t f>
```

How many  
Randomized  
Algorithm =

How fast?

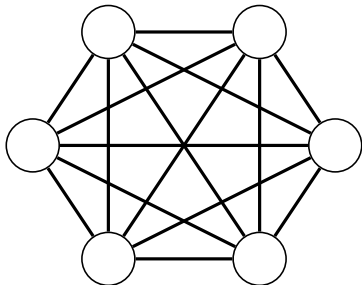


Deterministic  
Algorithm

```
#include <string>
using namespace std;

int main(void)
{
    cout << "Hello, World!" << endl;
    return 0;
}
```

## How Many Random Bits?



$\log n$

## How Many Random Bits?



## How Many Random Bits?

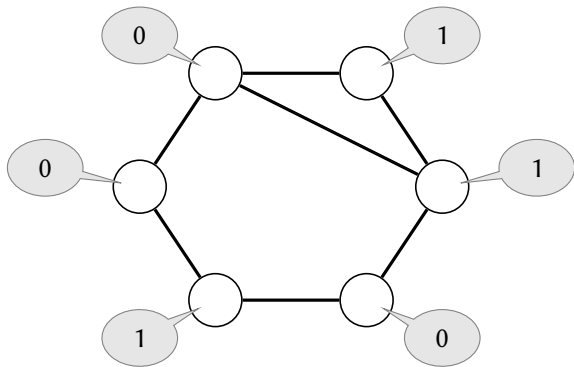


$\log n$

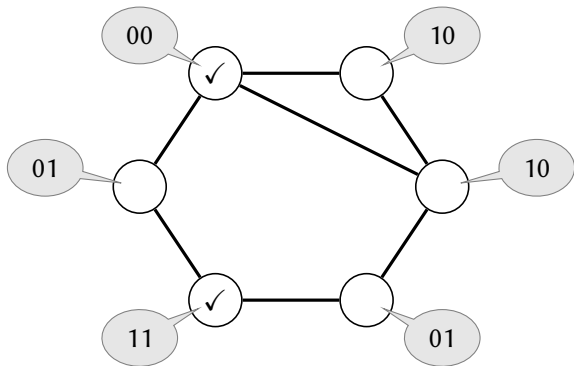




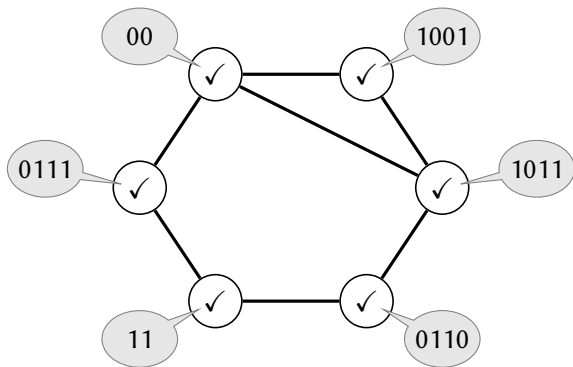
## How Fast?



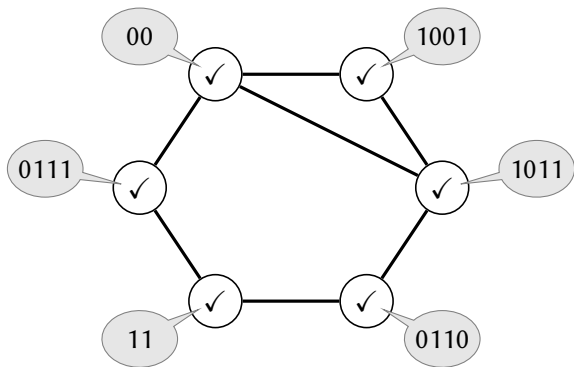
## How Fast?



## How Fast?

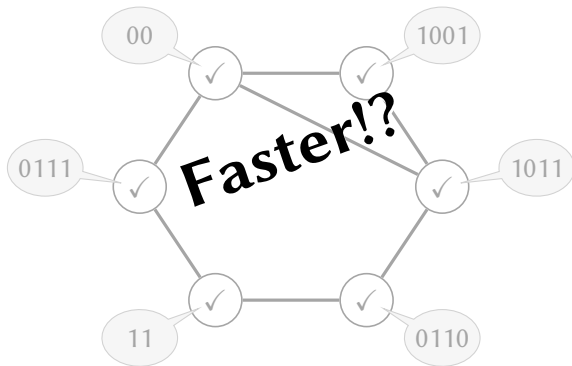


## How Fast?

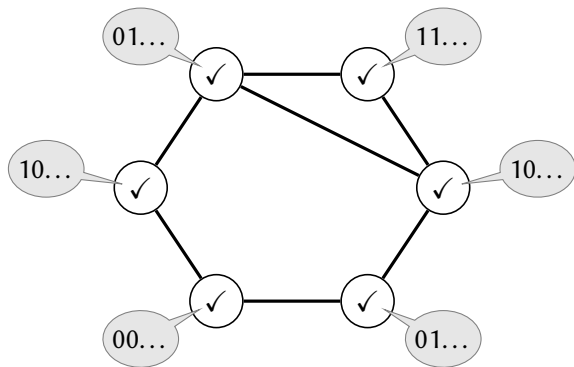


$\log \log n$  rounds,  $\log n$  bits  
(w.h.p. & in expectation)

## How Fast?

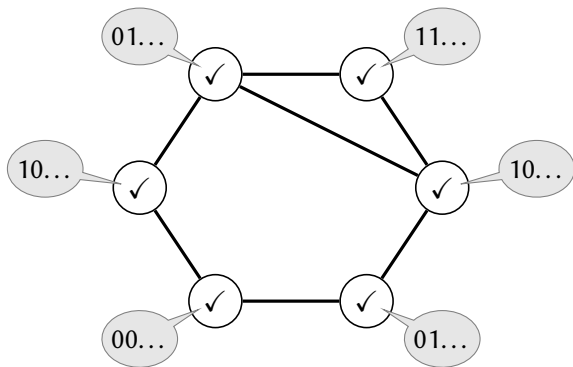


## How Fast?



$\infty$  random bits  $\Rightarrow$  1 round

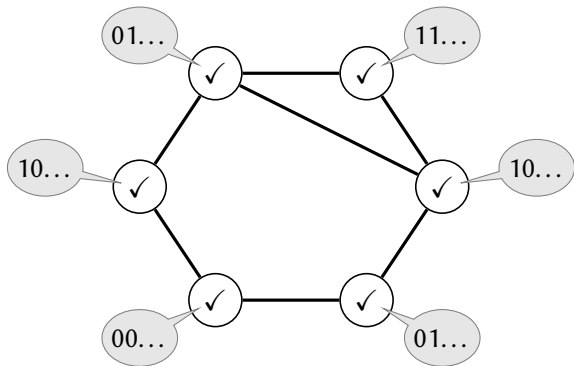
## How Fast?



$< \log^* n$  rounds  $\Rightarrow$    $\Rightarrow \infty$  random bits

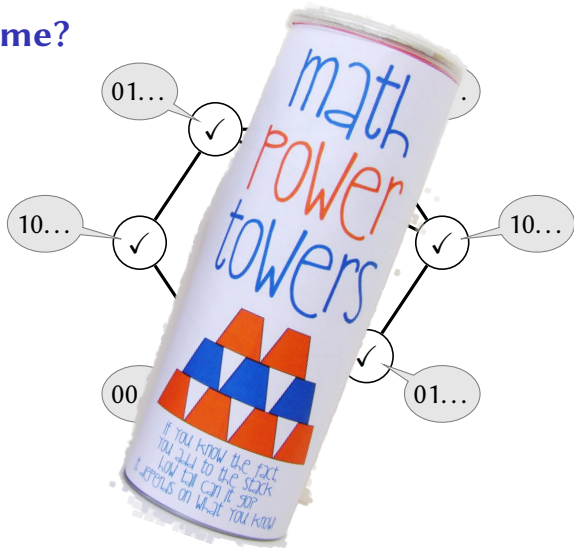


## $\log^*$ Runtime?



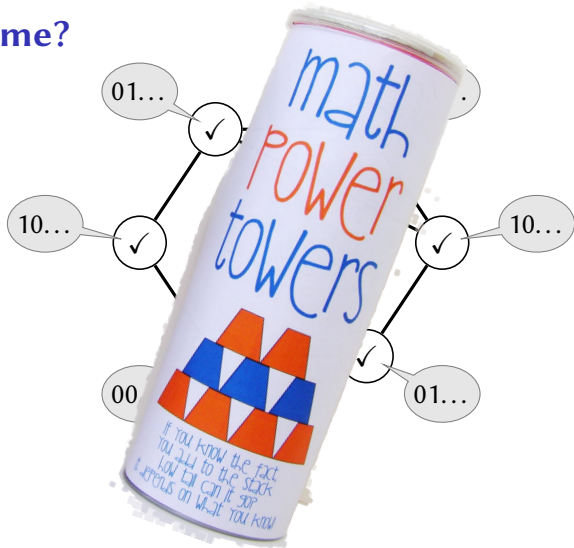
Idea: Use the inverse of  $\log^*$

# $\log^*$ Runtime?



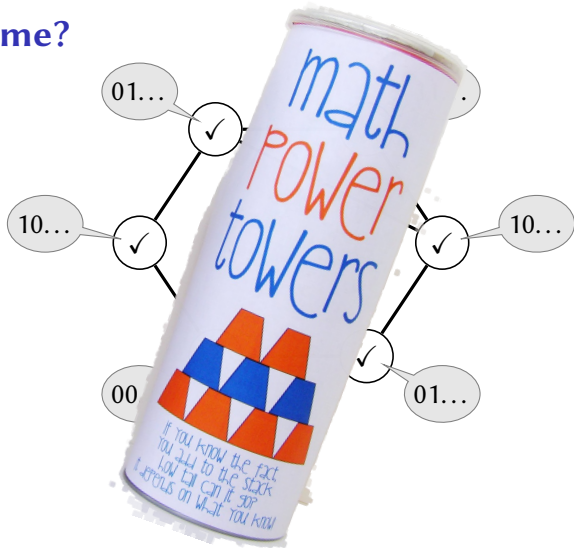
Idea: Use the inverse of  $\log^*$

# $\log^*$ Runtime?



use  $\log_2^{\dots^2}$  bits upto round  $i \Rightarrow \log^* n$  rounds,  $n$  bits

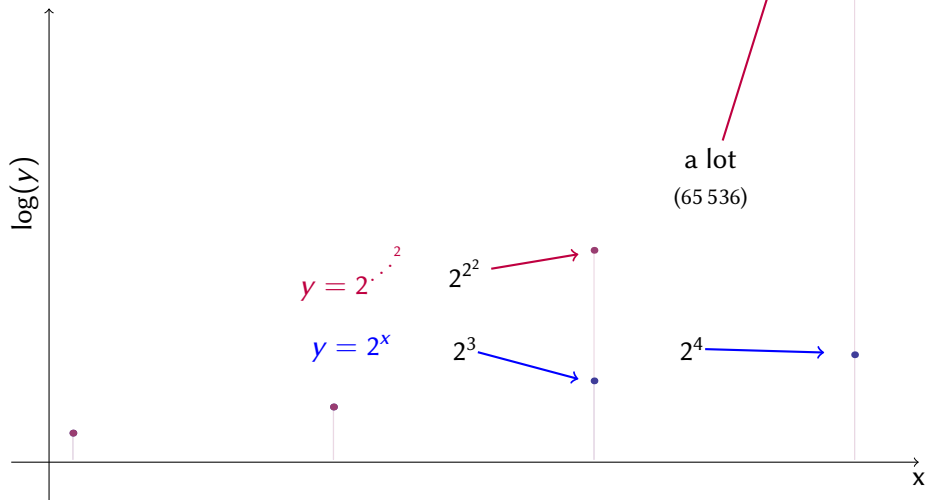
# $\log^*$ Runtime?



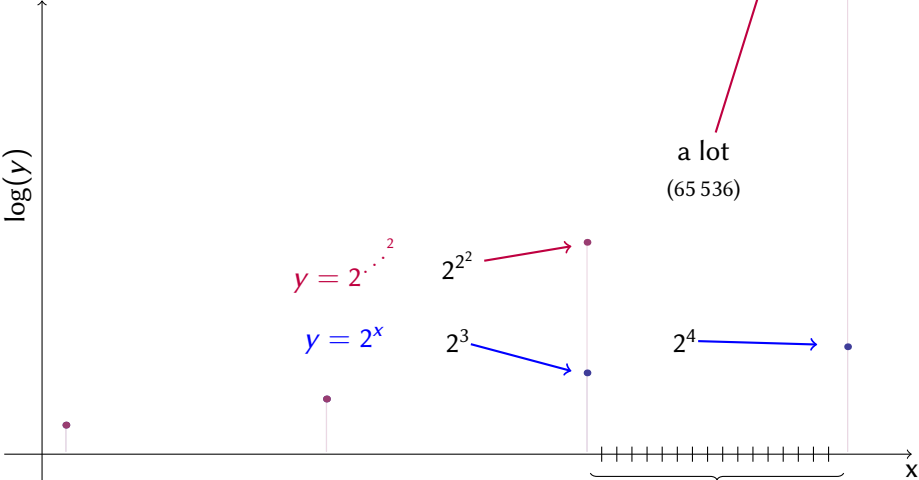
use  $\log_2^{\dots^2}$  bits upto round  $i \Rightarrow \log^* n$  rounds,  $n$  bits

Lower bound?

# Power Tower

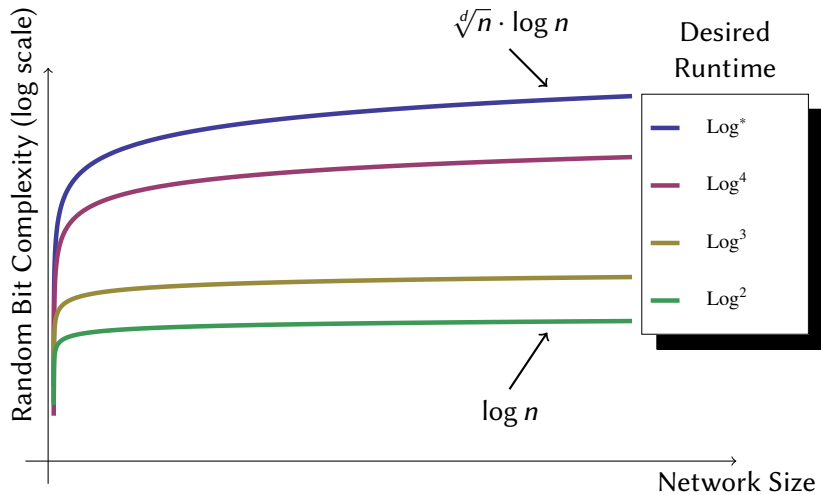


# Power Tower



Mind the gap  
Take more than one step!

# Time vs. Random Bit Complexity



# Time vs. Random Bit Complexity

## Theorem

*Anonymous networks can be 2-hop colored in  $O(d \cdot f(n))$  rounds.<sup>1</sup>*

*The achieved random bit complexity is*

$$O\left(\sqrt[d]{\frac{\lceil \log f^{-1}(f(n) + 1) \rceil}{\lceil \log f^{-1}(f(n)) \rceil}} \cdot \log n\right).$$

---

<sup>1</sup>With high probability and in expectation. For reasonable desired runtime  $f$ , i.e., between  $\log^*$  and  $\log \log$ .



# Time vs. Random Bit Complexity

## Theorem

*Anonymous networks can be 2-hop colored in  $O(d \cdot f(n))$  rounds.<sup>1</sup>*

*The achieved random bit complexity is*

$$O\left(d \sqrt{\frac{\lceil \log f^{-1}(f(n) + 1) \rceil}{\lceil \log f^{-1}(f(n)) \rceil}} \cdot \log n\right).$$

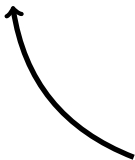
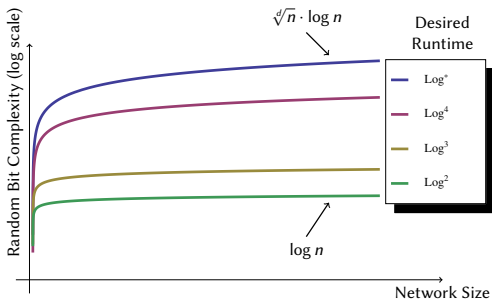
## Theorem

*This is asymptotically optimal.*

---

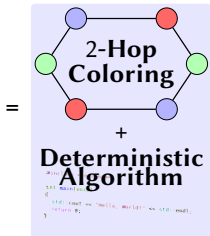
<sup>1</sup>With high probability and in expectation. For reasonable desired runtime  $f$ , i.e., between  $\log^*$  and  $\log \log$ .

# Recap

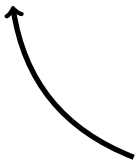
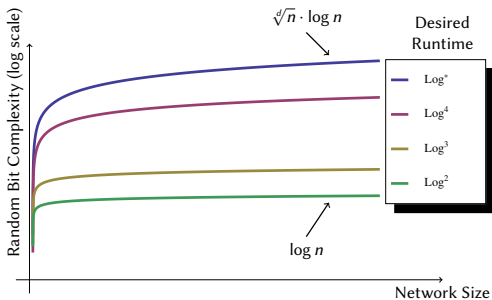


```
1 // Randomized algorithm for 2-hop coloring
2 // ...
3 ...
4 ...
5 ...
6 ...
7 ...
8 ...
9 ...
10 ...
11 ...
12 ...
13 ...
14 ...
15 ...
16 ...
17 ...
18 ...
19 ...
20 ...
21 ...
22 ...
23 ...
24 ...
25 ...
26 ...
27 ...
28 ...
29 ...
30 ...
31 ...
32 ...
33 ...
34 ...
35 ...
36 ...
37 ...
38 ...
39 ...
40 ...
41 ...
42 ...
43 ...
44 ...
45 ...
46 ...
47 ...
48 ...
49 ...
50 ...
51 ...
52 ...
53 ...
54 ...
55 ...
56 ...
57 ...
58 ...
59 ...
60 ...
61 ...
62 ...
63 ...
64 ...
65 ...
66 ...
67 ...
68 ...
69 ...
70 ...
71 ...
72 ...
73 ...
74 ...
75 ...
76 ...
77 ...
78 ...
79 ...
80 ...
81 ...
82 ...
83 ...
84 ...
85 ...
86 ...
87 ...
88 ...
89 ...
90 ...
91 ...
92 ...
93 ...
94 ...
95 ...
96 ...
97 ...
98 ...
99 ...
100 ...
```

**Randomized Algorithm**

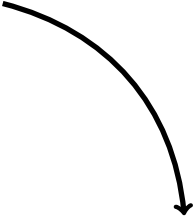
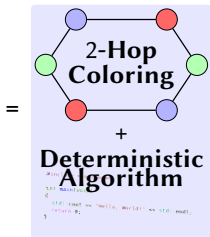


# Recap



```
1 // Randomized Algorithm
2 // ...
3 // ...
4 // ...
5 // ...
6 // ...
7 // ...
8 // ...
9 // ...
10 // ...
11 // ...
12 // ...
13 // ...
14 // ...
15 // ...
16 // ...
17 // ...
18 // ...
19 // ...
20 // ...
21 // ...
22 // ...
23 // ...
24 // ...
25 // ...
26 // ...
27 // ...
28 // ...
29 // ...
30 // ...
31 // ...
32 // ...
33 // ...
34 // ...
35 // ...
36 // ...
37 // ...
38 // ...
39 // ...
40 // ...
41 // ...
42 // ...
43 // ...
44 // ...
45 // ...
46 // ...
47 // ...
48 // ...
49 // ...
50 // ...
51 // ...
52 // ...
53 // ...
54 // ...
55 // ...
56 // ...
57 // ...
58 // ...
59 // ...
60 // ...
61 // ...
62 // ...
63 // ...
64 // ...
65 // ...
66 // ...
67 // ...
68 // ...
69 // ...
70 // ...
71 // ...
72 // ...
73 // ...
74 // ...
75 // ...
76 // ...
77 // ...
78 // ...
79 // ...
80 // ...
81 // ...
82 // ...
83 // ...
84 // ...
85 // ...
86 // ...
87 // ...
88 // ...
89 // ...
90 // ...
91 // ...
92 // ...
93 // ...
94 // ...
95 // ...
96 // ...
97 // ...
98 // ...
99 // ...
100 // ...
```

**Randomized Algorithm**



**Thank You**