

# Anonymous Networks: Randomization = 2-Hop Coloring

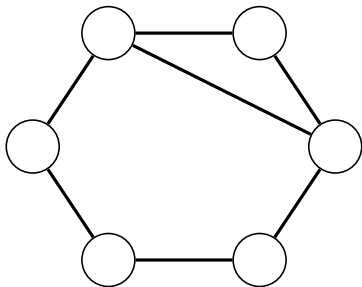


*Yuval Emek*<sup>1</sup>   *Christoph Pfister*<sup>2</sup>   ***Jochen Seidel***<sup>2</sup>   *Roger Wattenhofer*<sup>2</sup>

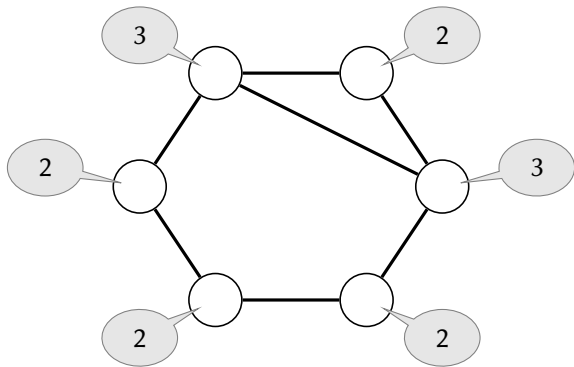
<sup>1</sup>*Technion – Faculty of Industrial Engineering and Management – [ie.technion.ac.il](http://ie.technion.ac.il)*

<sup>2</sup>*ETH Zurich – Distributed Computing Group – [www.disco.ethz.ch](http://www.disco.ethz.ch)*

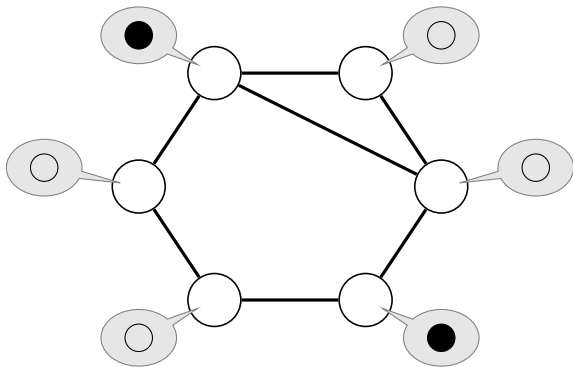
# Anonymous Networks



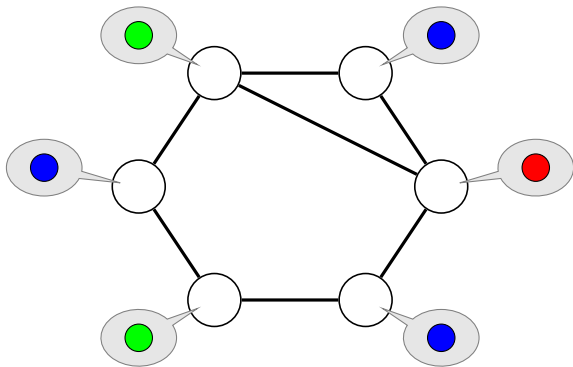
# Anonymous Networks



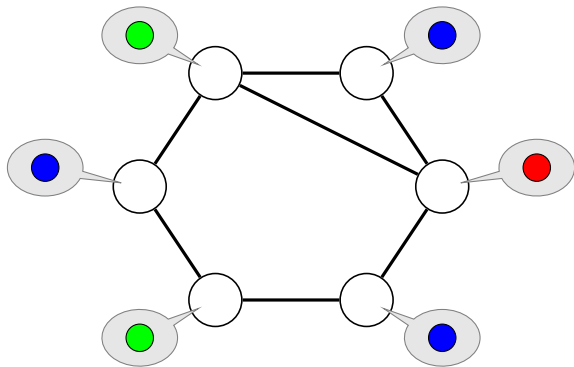
## Maximal Independent Set



# Coloring

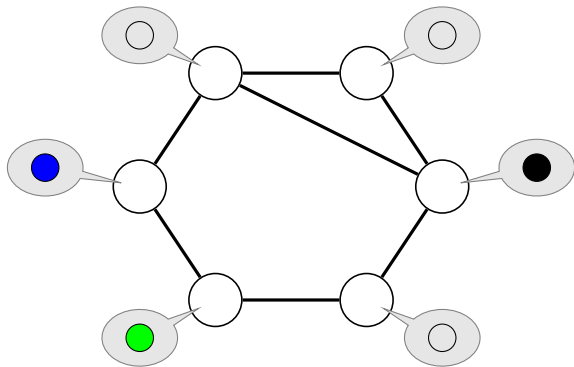


## Coloring $\rightarrow$ Maximal Independent Set



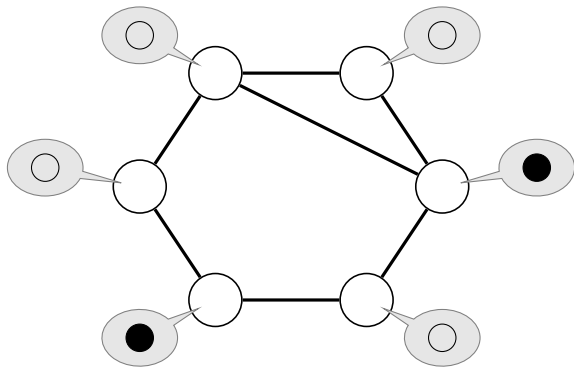
● < ● < ●

## Coloring $\rightarrow$ Maximal Independent Set



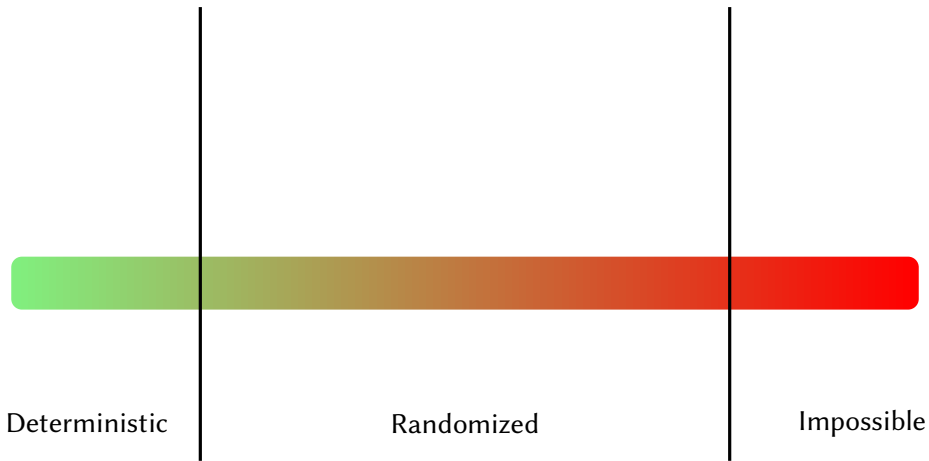
● < ● < ●

## Coloring $\rightarrow$ Maximal Independent Set

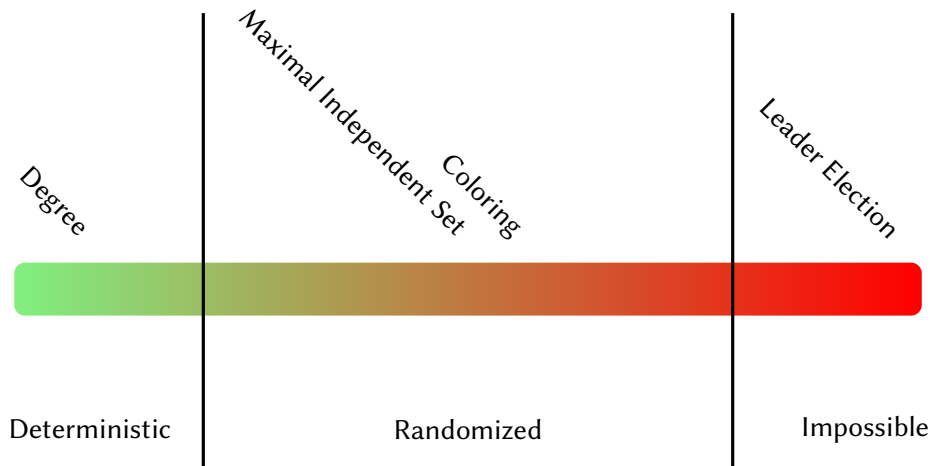




# Computability



# Computability

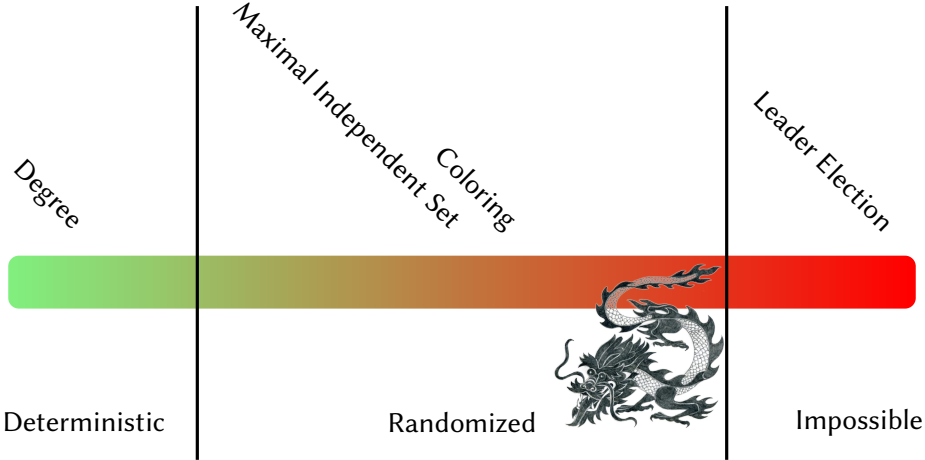


D. Angluin.

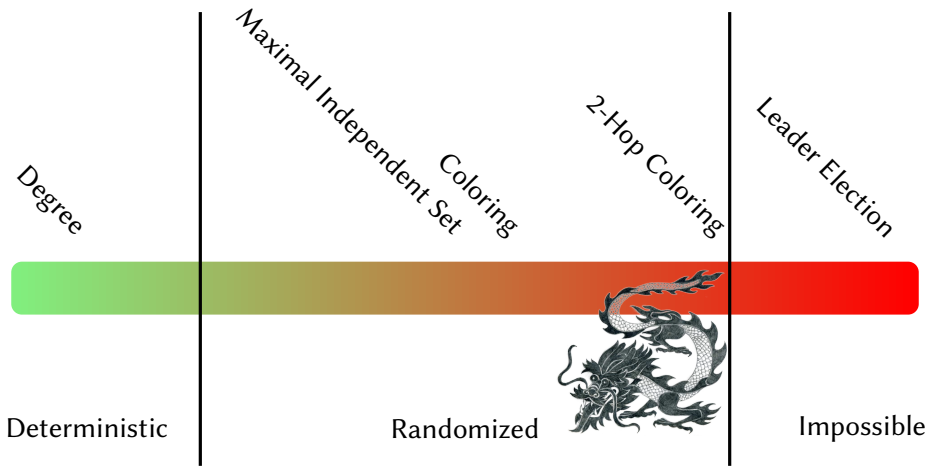
Local and global properties in networks of processors (extended abstract).

*STOC*, 1980.

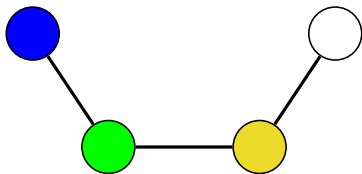
# Computability



# Computability



## 2-Hop Coloring?!



# Theorem

```
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-n-unroll_extra; j++) {
            UIntType y = (x[j] & upper_mask) | (x[j+1] & lower_mask);
            x[j] = x[j+1] ^ (y >> 1) ^ ((x[j+1]&l) * a);
        }
        for(std::size_t j = n-unroll_extra; j < n-m; j++) {
            UIntType y = (x[j] & upper_mask) | (x[j+1] & lower_mask);
            x[j] = x[j+1] ^ (y >> 1) ^ ((x[j+1]&l) * 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] ^ (y >> 1) ^ ((x[j+1]&l) * a);
        }
        // last iteration
        UIntType y = (x[n-1] & upper_mask) | (x[0] & lower_mask);
        x[n-1] = x[0] ^ (y >> 1) ^ ((x[0]&l) * 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,
        UIntType b, std::size_t t, UIntType c, l, f, std::size_t f>
```

# Randomized Algorithm

=<sup>1</sup>



+

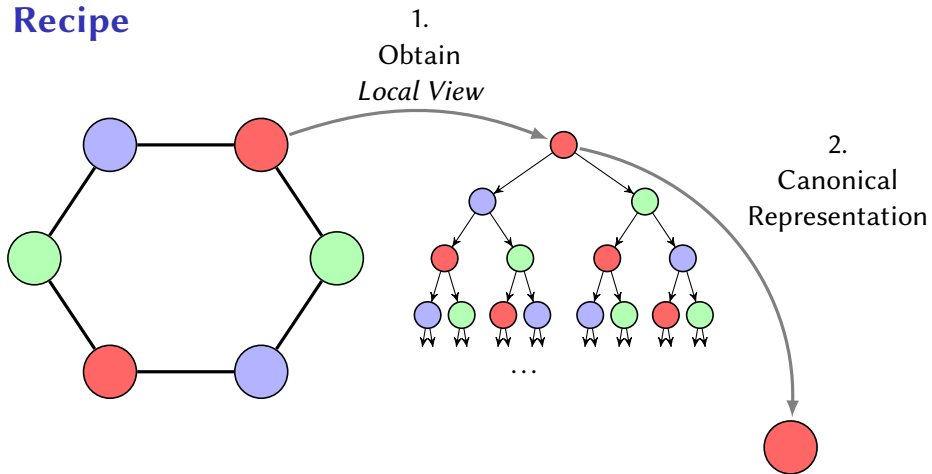
# Deterministic Algorithm

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

<sup>1</sup>Subject to Restrictions

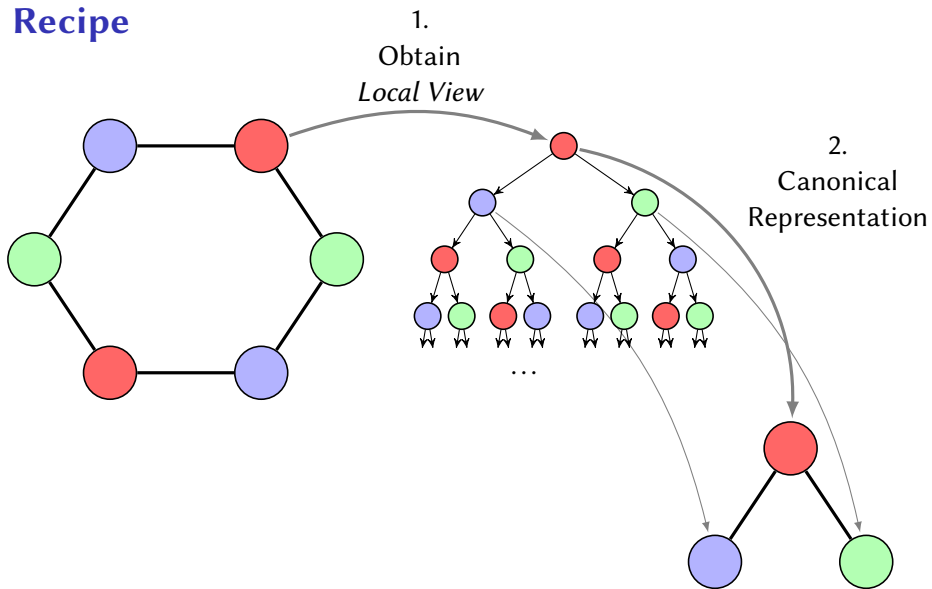


# Recipe

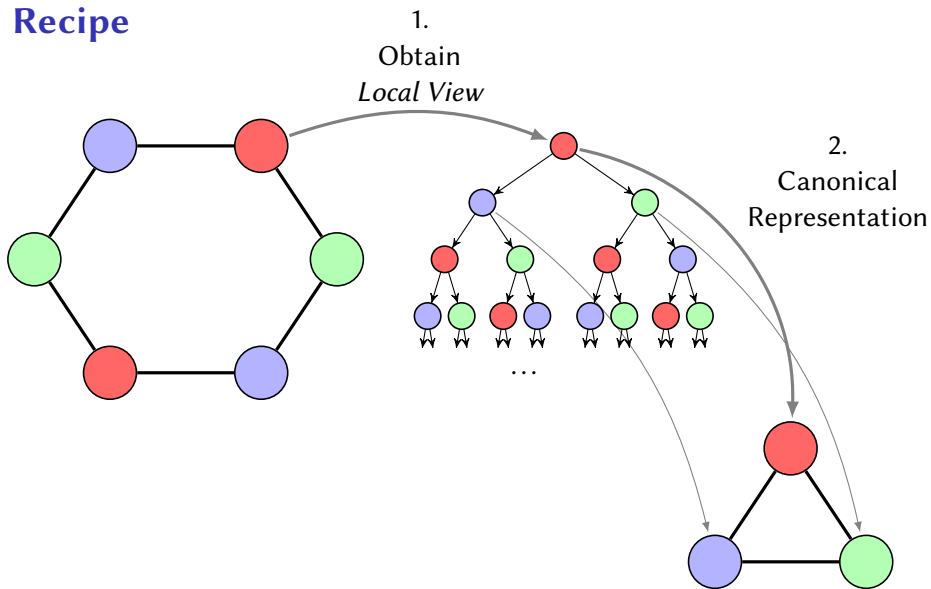




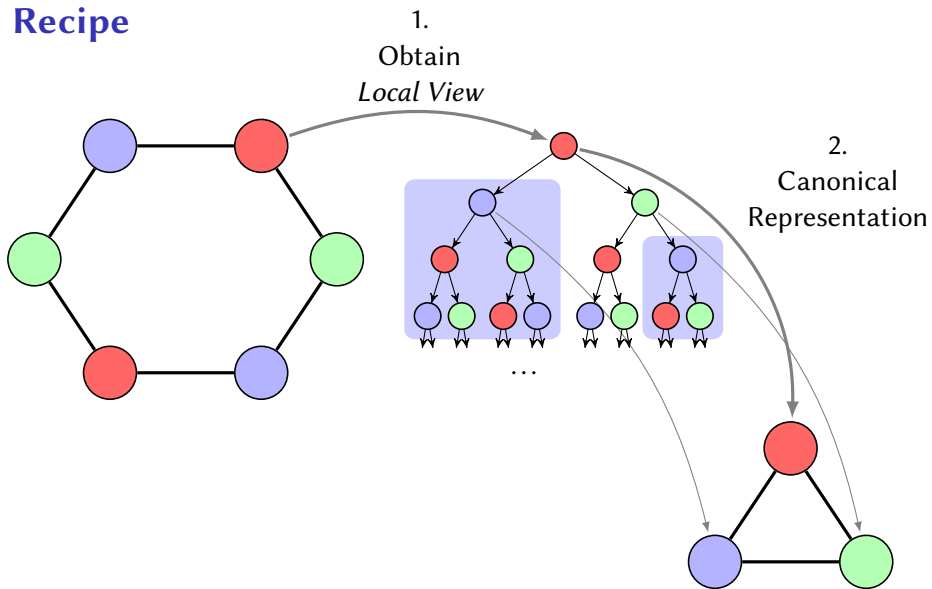
# Recipe



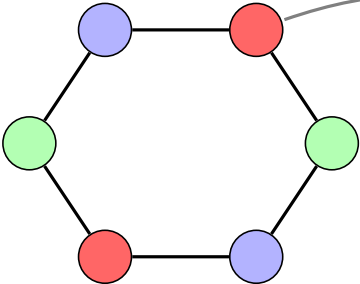
# Recipe



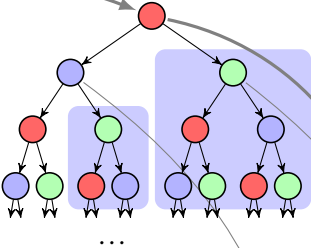
# Recipe



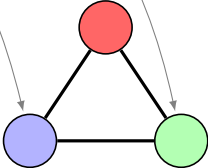
# Recipe



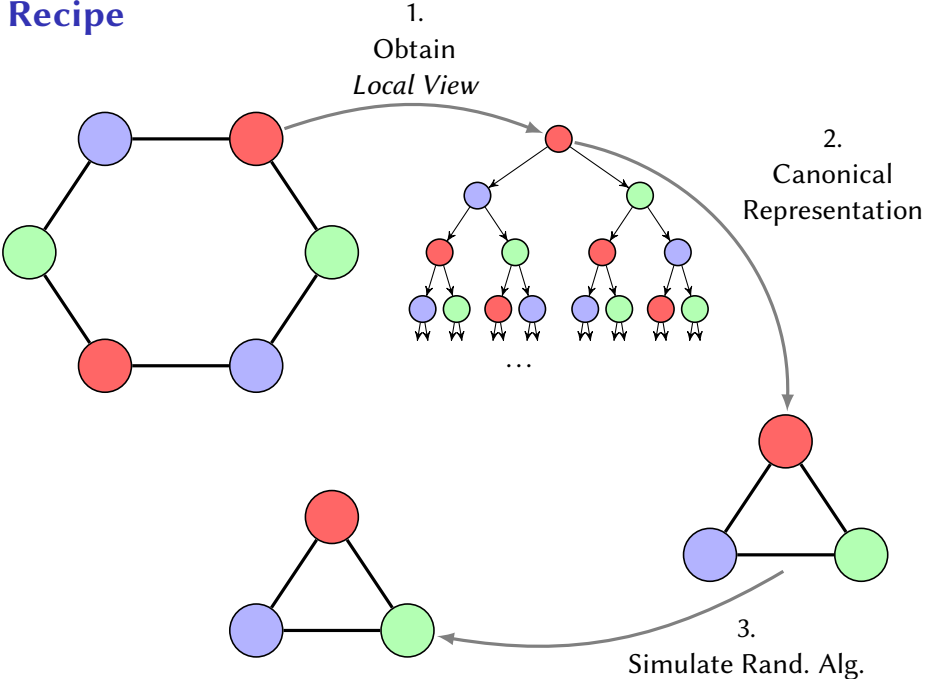
1.  
Obtain  
*Local View*



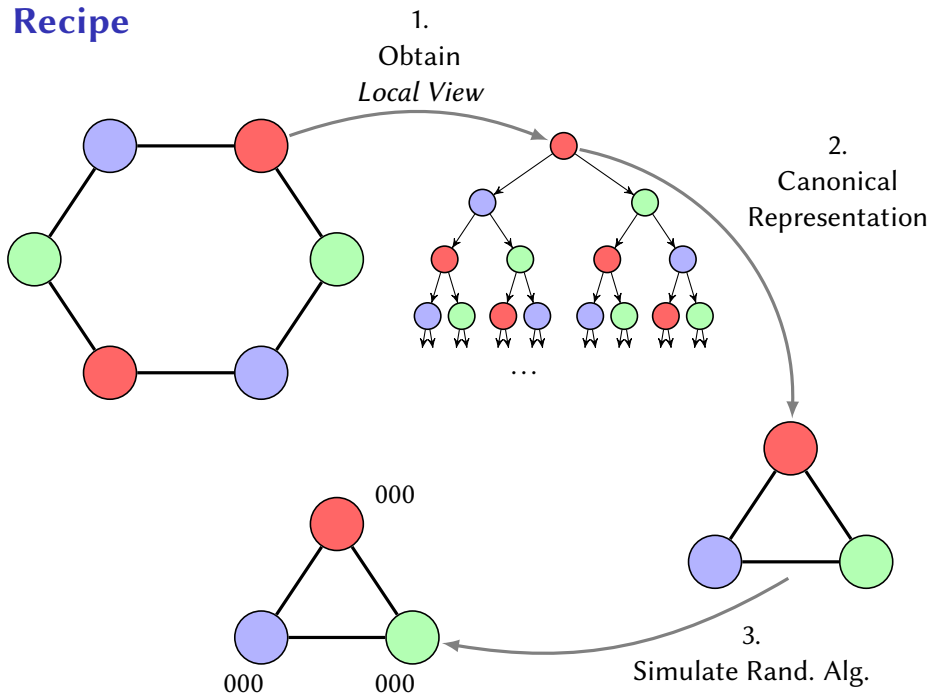
2.  
Canonical  
Representation



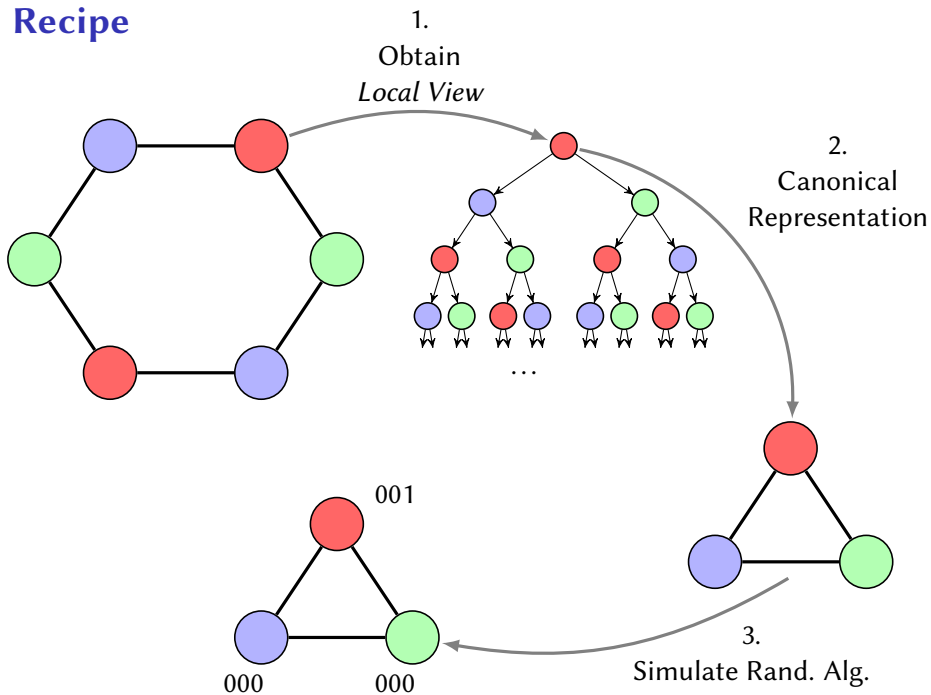
# Recipe



# Recipe



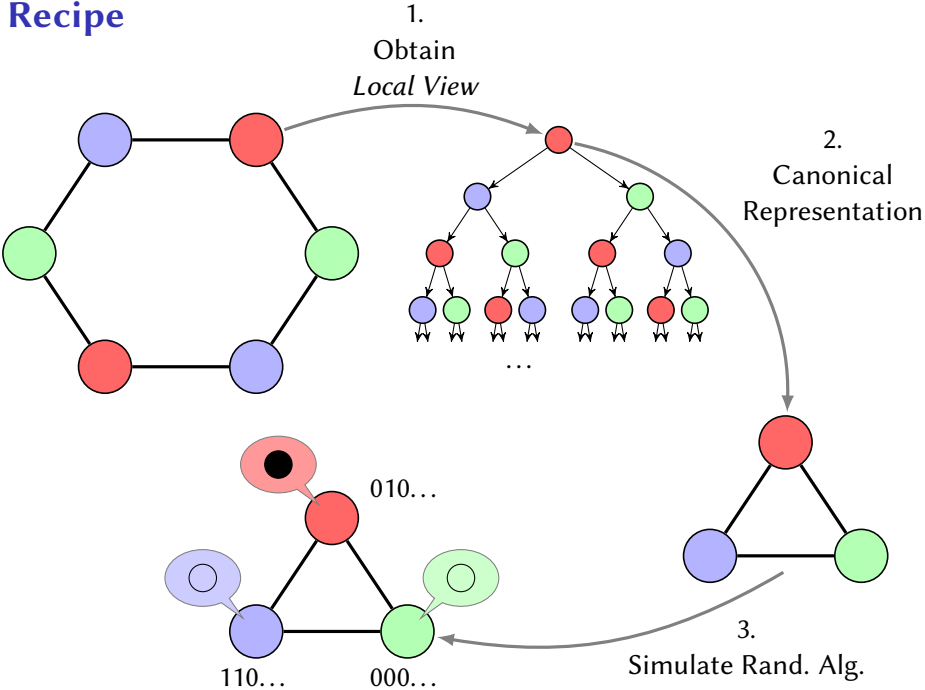
# Recipe



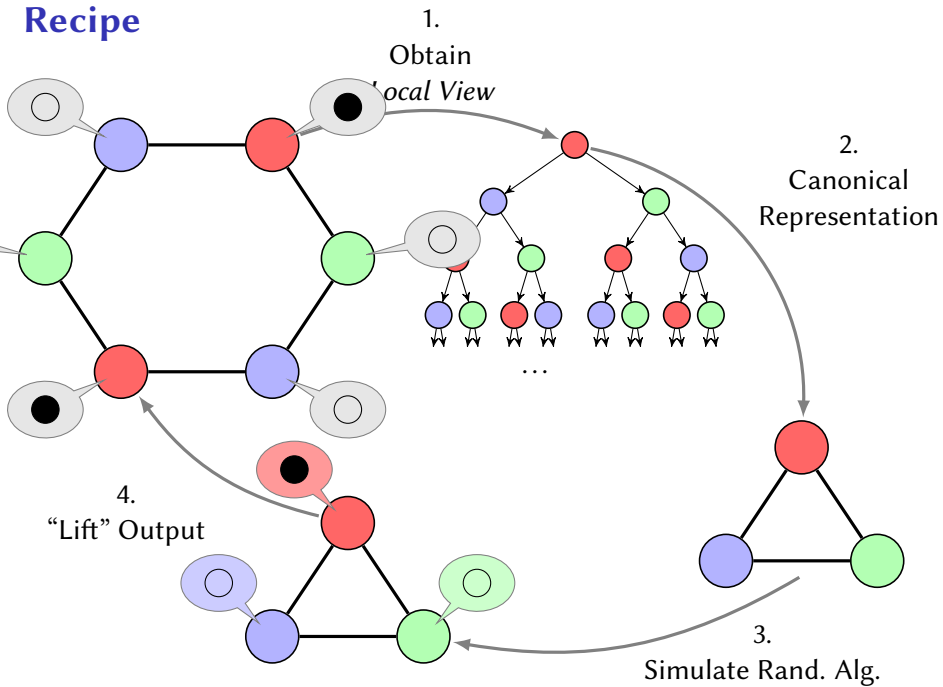




# Recipe



# Recipe





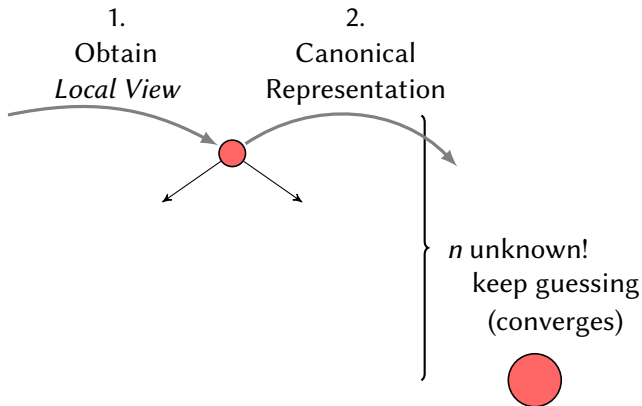






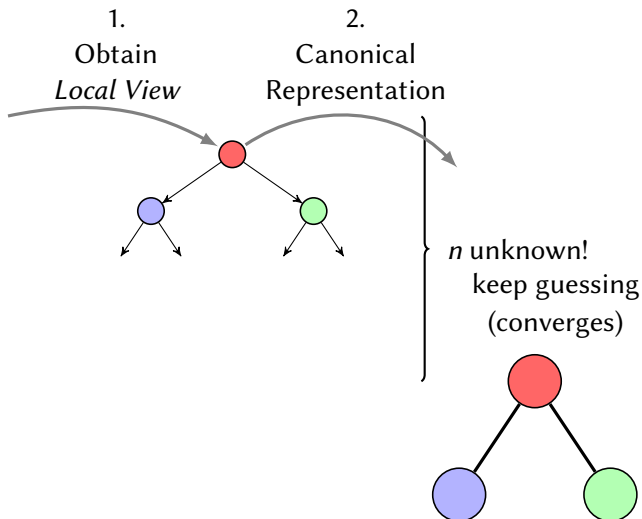


# Obstacles



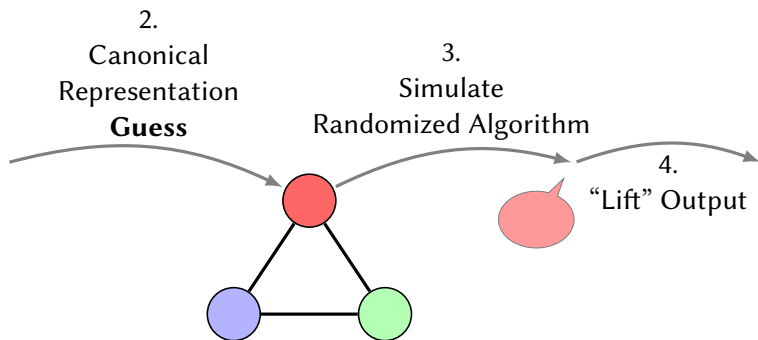


# Obstacles

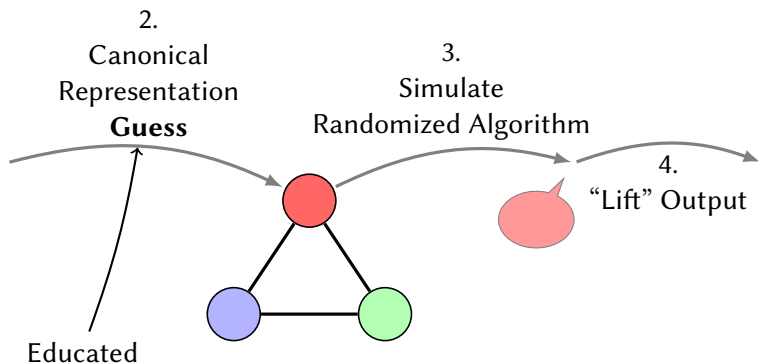




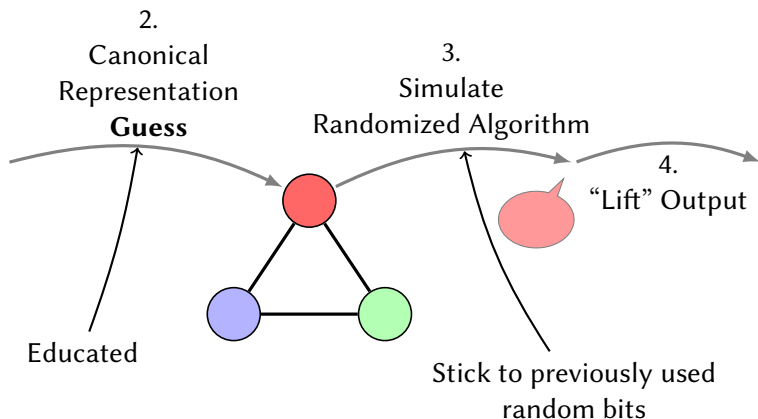
# Obstacles



# Obstacles



# Obstacles



# Theorem

```
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-n-unroll_extra; j++) {
            UIntType y = (x[j] & upper_mask) | (x[j+1] & lower_mask);
            x[j] = x[j+1] ^ (y >> 1) ^ ((x[j+1]&l) * a);
        }
        for(std::size_t j = n-unroll_extra; j < n-m; j++) {
            UIntType y = (x[j] & upper_mask) | (x[j+1] & lower_mask);
            x[j] = x[j+1] ^ (y >> 1) ^ ((x[j+1]&l) * 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] ^ (y >> 1) ^ ((x[j+1]&l) * a);
        }
        // last iteration
        UIntType y = (x[n-1] & upper_mask) | (x[0] & lower_mask);
        x[n-1] = x[0] ^ (y >> 1) ^ ((x[0]&l) * 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,
        UIntType b, std::size_t t, UIntType c, l, f>
```

# Randomized Algorithm

=<sup>1</sup>



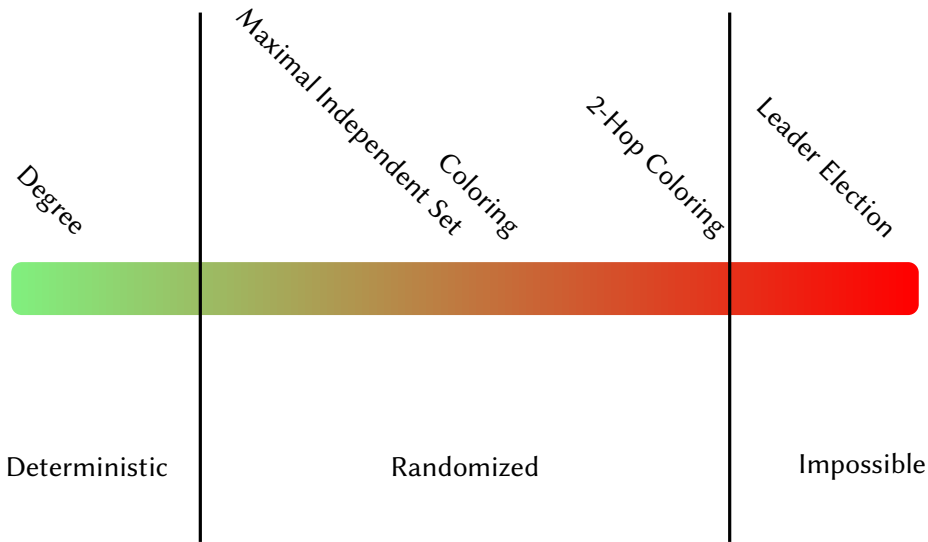
+

# Deterministic Algorithm

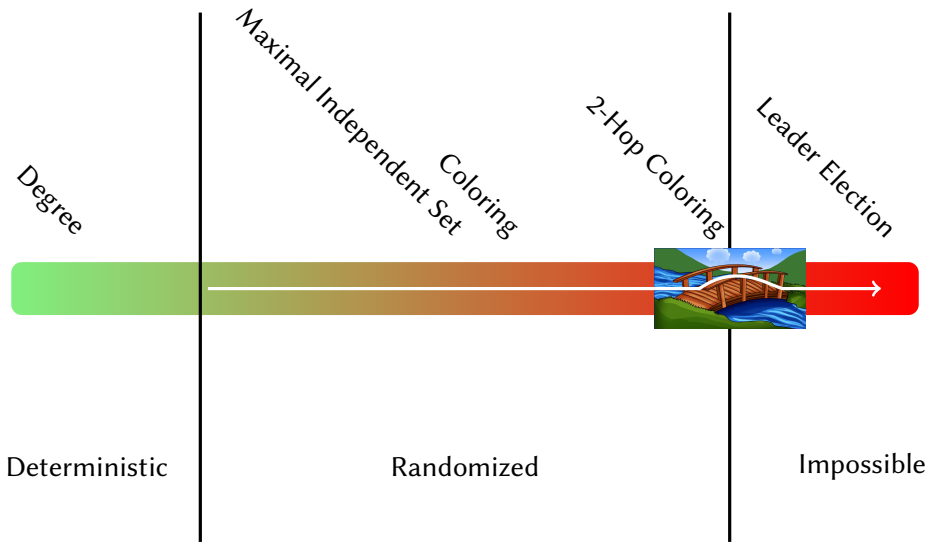
```
#include <string>
#include <iostream>
int main(void)
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

<sup>1</sup>Subject to Restrictions

# Computability

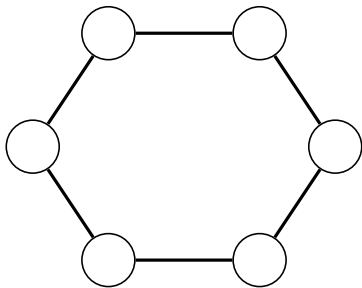


# Computability

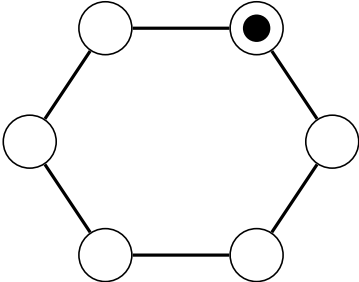




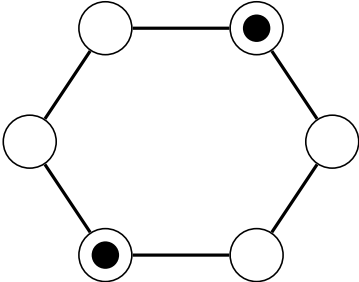
# Leader Election



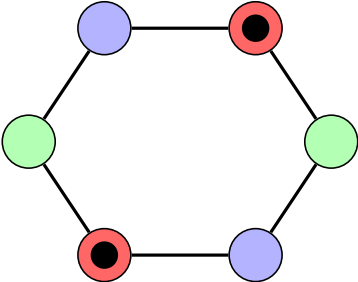
# Leader Election



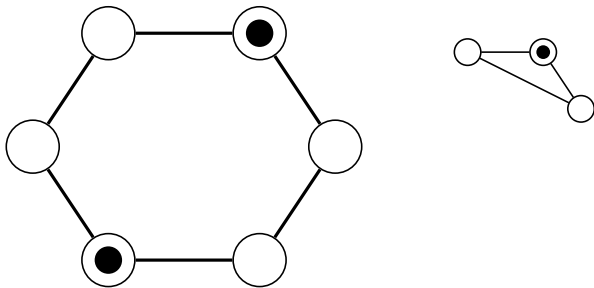
# Leader Election



# Leader Election

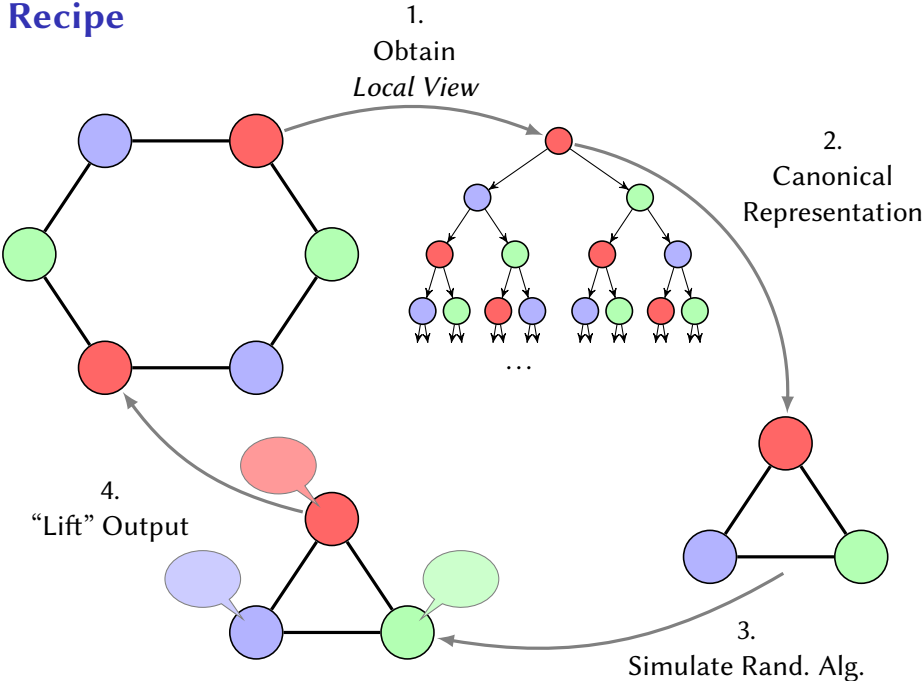


# Leader Election



- ▶ “Promise” must be *decidable* with anonymous algorithm

# Recipe



# Summary

```
using namespace std;
int main() {
    // Randomized Algorithm
    // ...
}

```

**Randomized Algorithm**

=

**2-Hop Coloring**

+

**Deterministic Algorithm**

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

```

