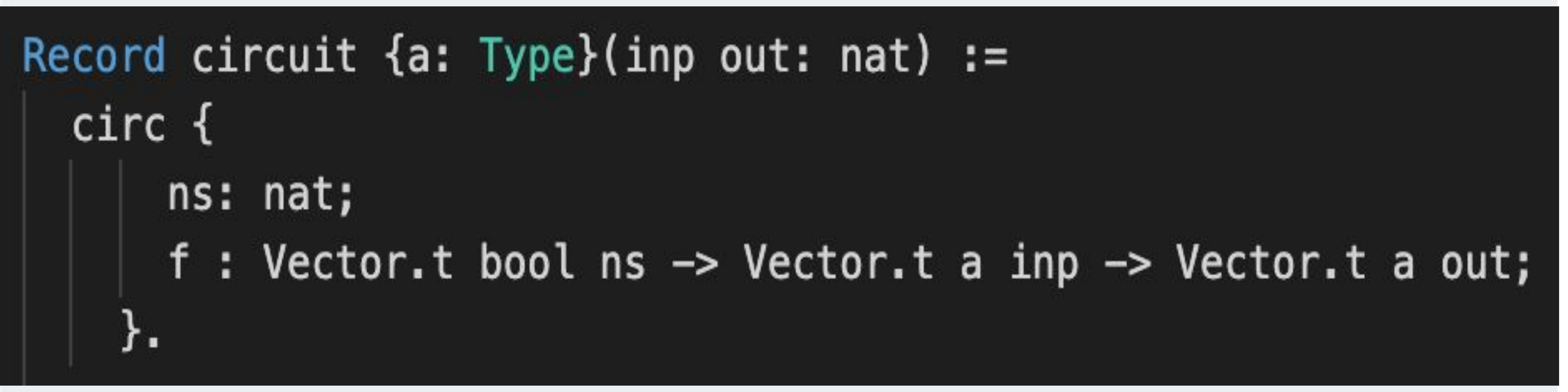
The Formalization of Permutation Networks

Christa E. Simaan - SEAS 2024

What is a Permutation Network?

Permutation networks are circuits of configurable switches, such that their output is always a permutation of the input.



```
Record circuit {a: Type}(inp out: nat) :=
  circ {
      ns: nat;
      f : Vector.t bool ns -> Vector.t a inp -> Vector.t a out;
    }.
```

Mutex (Mutual Exclusion)

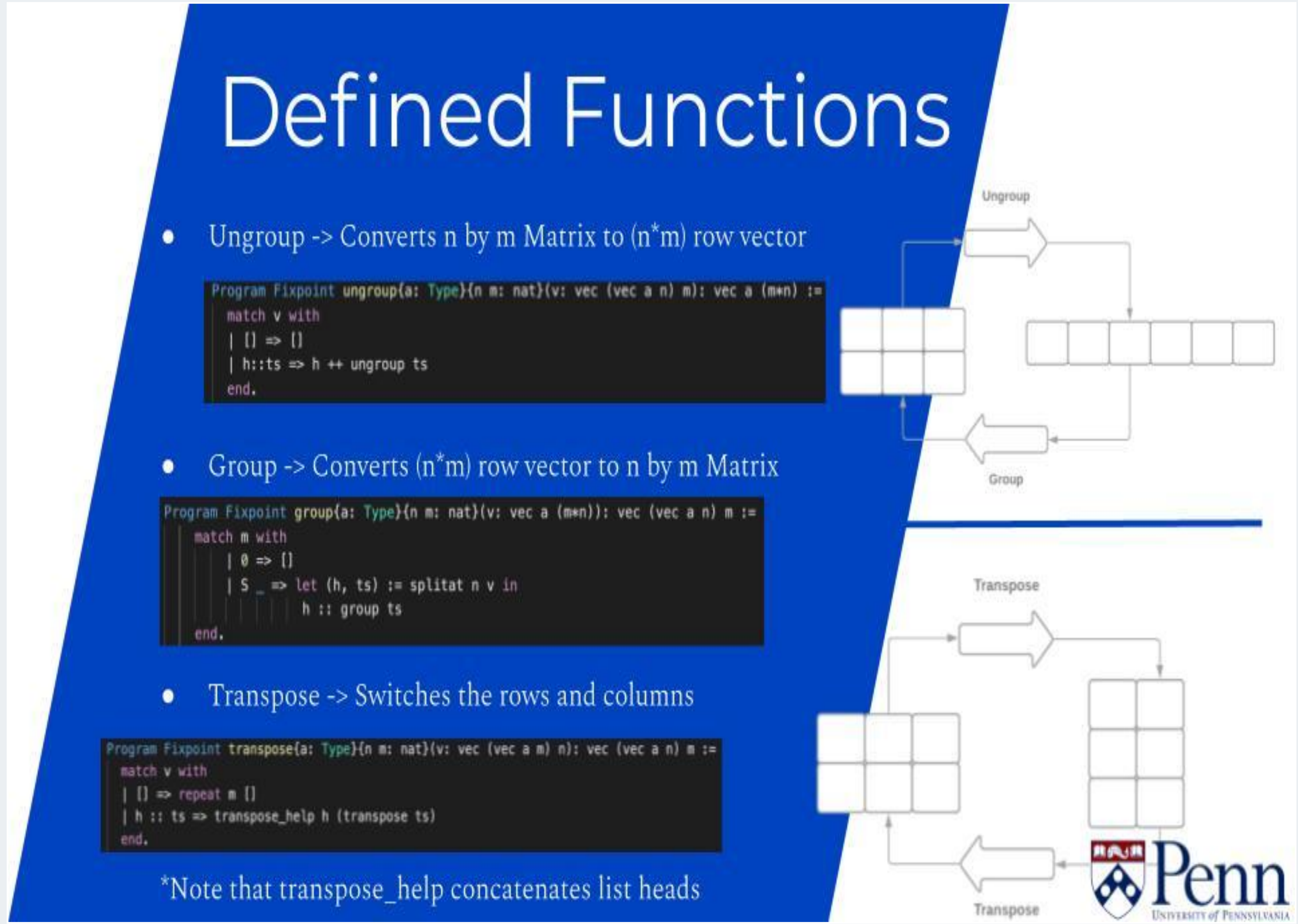
A mutex is a column of switches. It is appended to both left and right side of a benes network in the inductive definition. A mutex of size n has n switches, 2*n inputs, and 2*n outputs.

Our choice of permutation network...

We chose to use the benes network. The size of the benes network is exponential to `n` which we pass as an argument to the construction of the circuit. The `n` in benes is proportional to the size of the circuit, number of switches. The number of inputs = outputs which is $2^{S\ n}$, S n denoting the successor of n, or $2*2^n$. The number of switches is equal to $(2*n + 1) * 2^n$. A benes network is composed of two mutexes, multiple stages of switches, and the proper wirings that connect them all.

Permutation Network Properties:

1. Configurable → n configuration bits, n! permutations
2. Reversible → involution
3. Scalable → Poly-logarithmic number of configurations
4. Parametric polymorphic → Generalization of types
5. # Inputs = # Outputs → Must be a power of 2



Defined Functions

- Ungroup -> Converts n by m Matrix to (n*m) row vector

```
Program Fixpoint ungroup{a: Type}{n m: nat}(v: vec (vec a n) m): vec a (m*n) :=
  match v with
  | [] => []
  | h::ts => h ++ ungroup ts
  end.
```

- Group -> Converts (n*m) row vector to n by m Matrix

```
Program Fixpoint group{a: Type}{n m: nat}(v: vec a (m*n)): vec (vec a n) m :=
    match m with
        | 0 => []
        | S _ => let (h, ts) := splitat n v in
                   h :: group ts
    end.
```

- Transpose -> Switches the rows and columns

```
Program Fixpoint transpose{a: Type}{n m: nat}(v: vec (vec a m) n): vec (vec a n) m :=
  match v with
  | [] => repeat m []
  | h :: ts => transpose_help h (transpose ts)
  end.
```

*Note that transpose_help concatenates list heads