Verification of Immediate Observation Population Protocols

Chana Weil-Kennedy

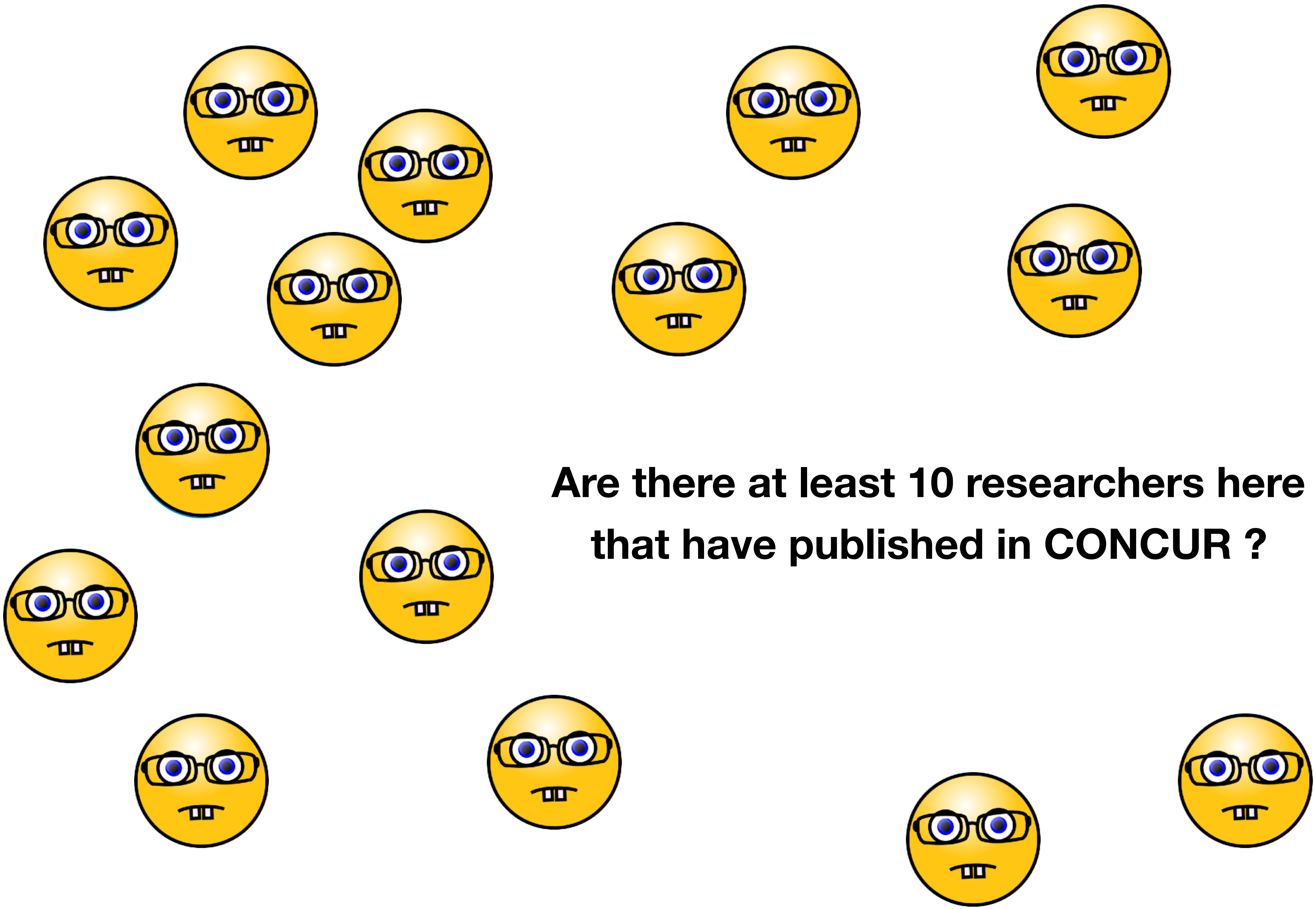
joint work with Javier Esparza, Pierre Ganty, Rupak Majumdar

European Research Council
Established by the European Commission

Introduction

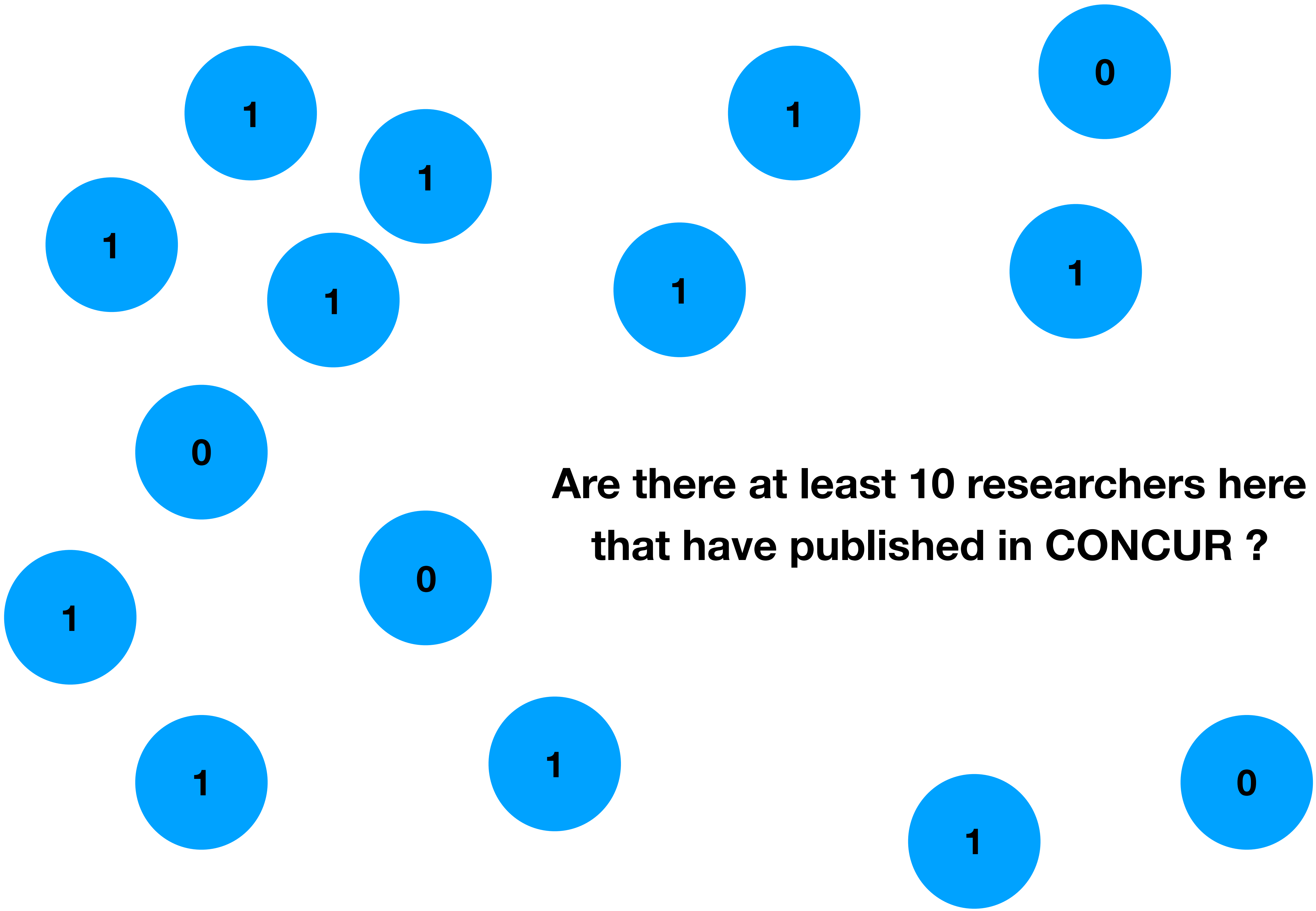
- Population protocols were introduced in 2004 by Angluin et al.
- They are a model of distributed computation by anonymous, identical, finite-state mobile agents with no global knowledge
- Motivating scenarios : networks of passively mobile sensors, propagation of trust, distributed computation in chemical reactions

Researchers in a Room



Are there at least 10 researchers here that have published in CONCUR ?

Researchers in a Room



Are there at least 10 researchers here that have published in CONCUR ?

Researchers in a Room

Are there at least 10 researchers here that have published in CONCUR ?

Researchers in a Room

Are there at least 10 researchers here that have published in CONCUR ?

Researchers in a Room

Are there at least 10 researchers here that have published in CONCUR ?

Researchers in a Room

Are there at least 10 researchers here that have published in CONCUR ?

Researchers in a Room

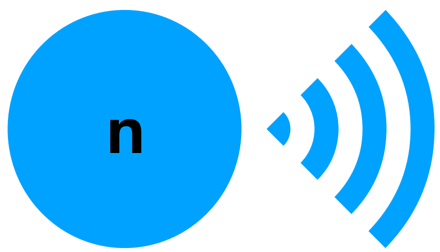
Are there at least 10 researchers here that have published in CONCUR ?

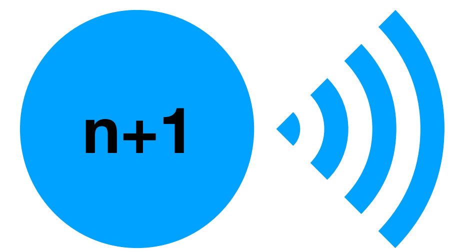
1 $\leq$ n $<$ 10

Researchers in a Room

Are there at least 10 researchers here that have published in CONCUR ?

$1 \leq n < 10$





Researchers in a Room

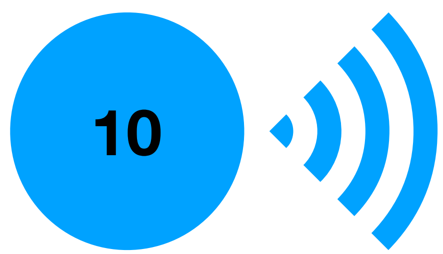
Are there at least 10 researchers here that have published in CONCUR ?

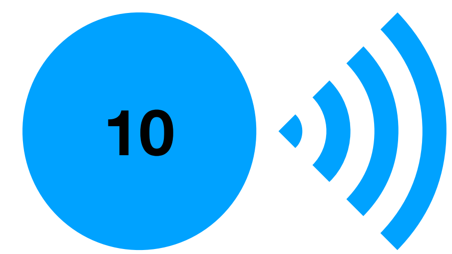
$0 \leq n < 10$

Researchers in a Room

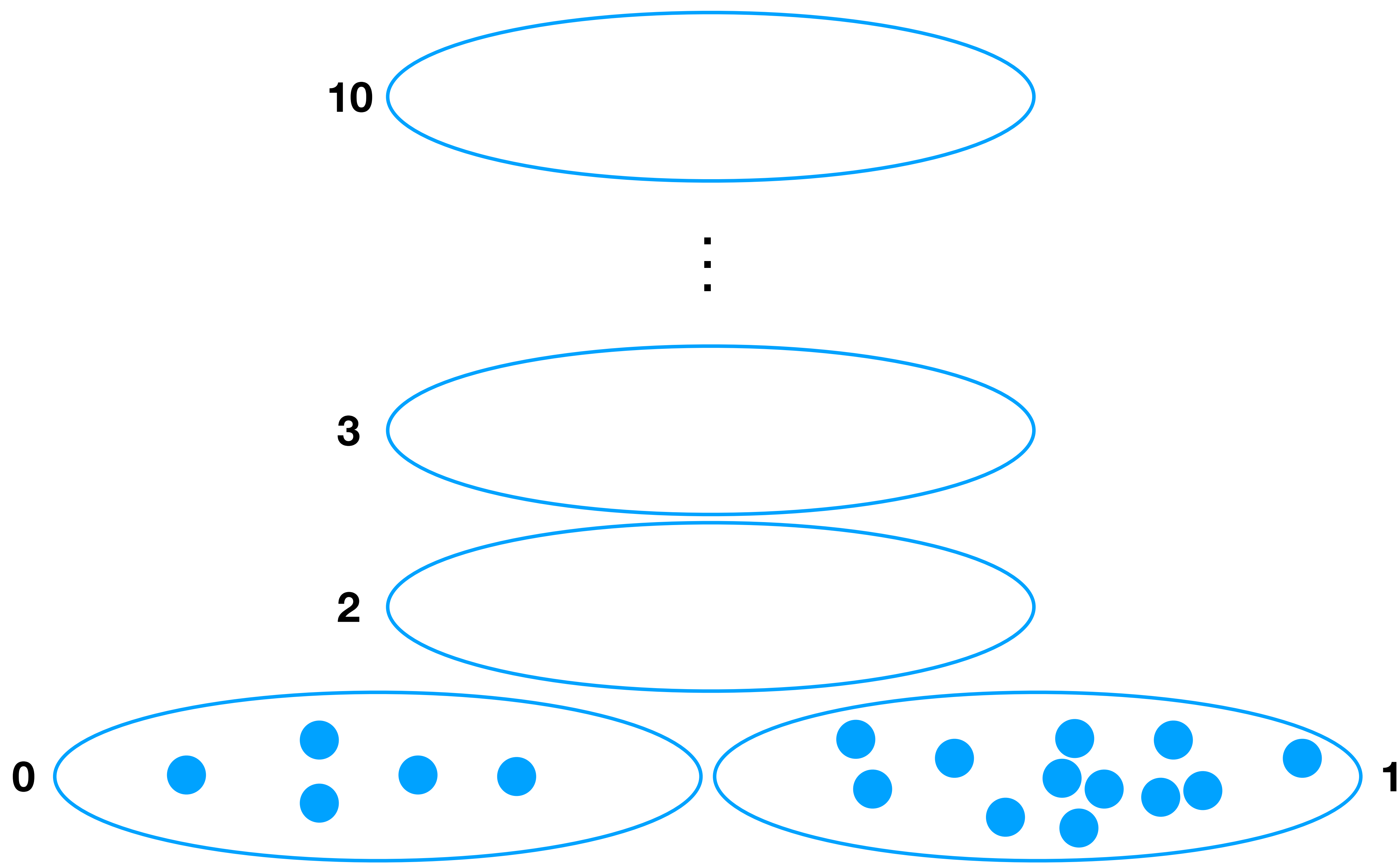
Are there at least 10 researchers here that have published in CONCUR ?

$0 \leq n < 10$

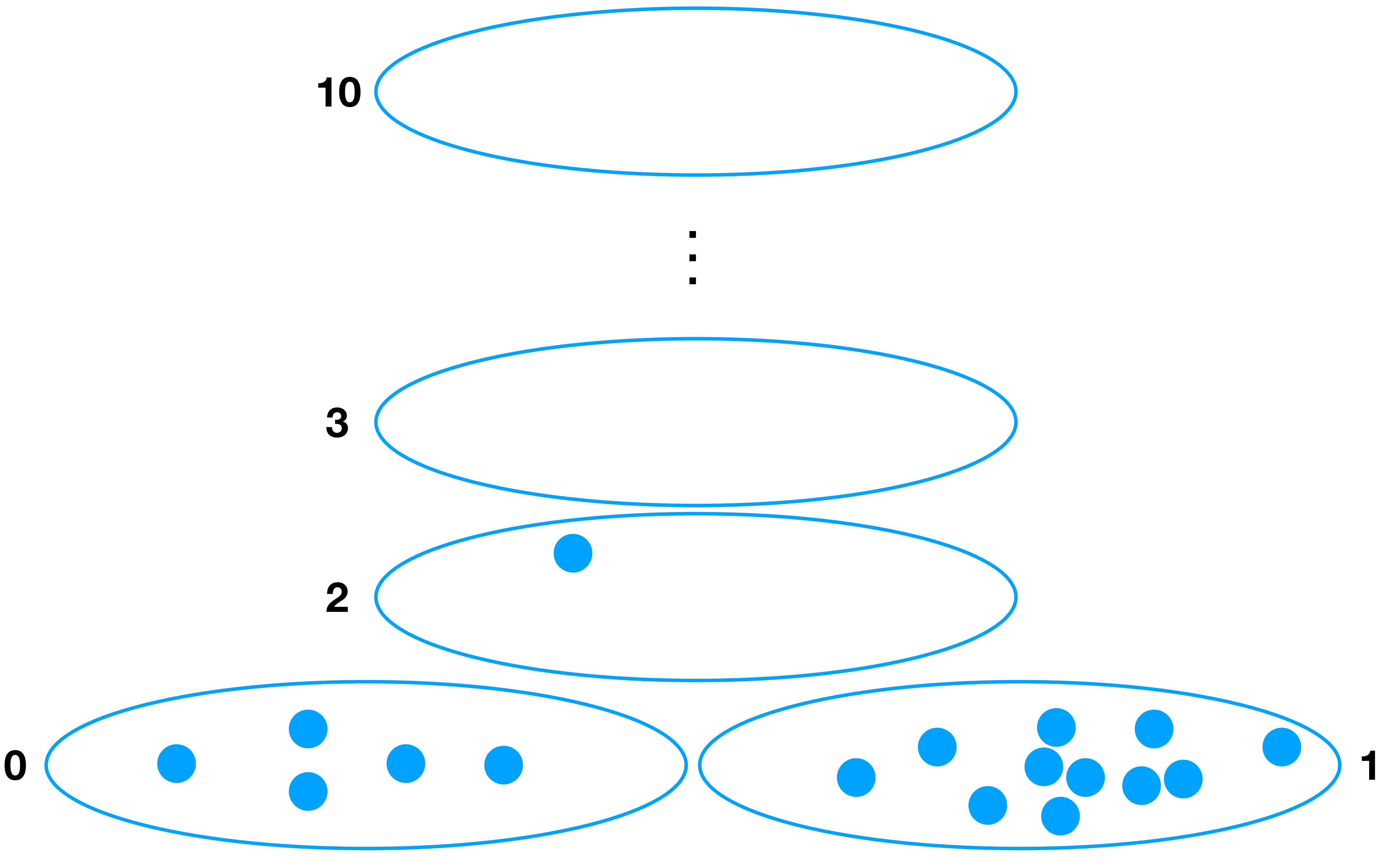




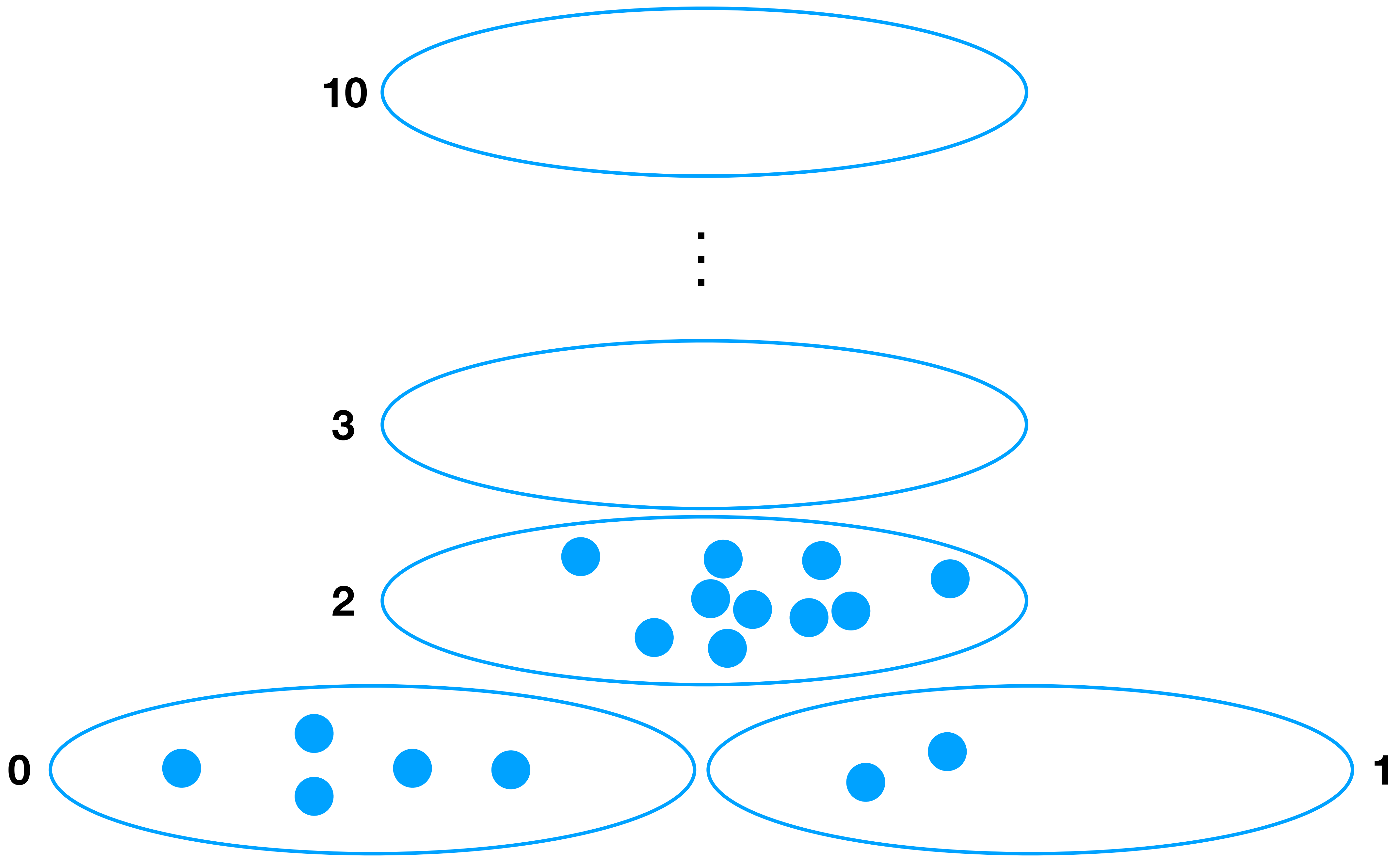
Counting Researchers Protocol



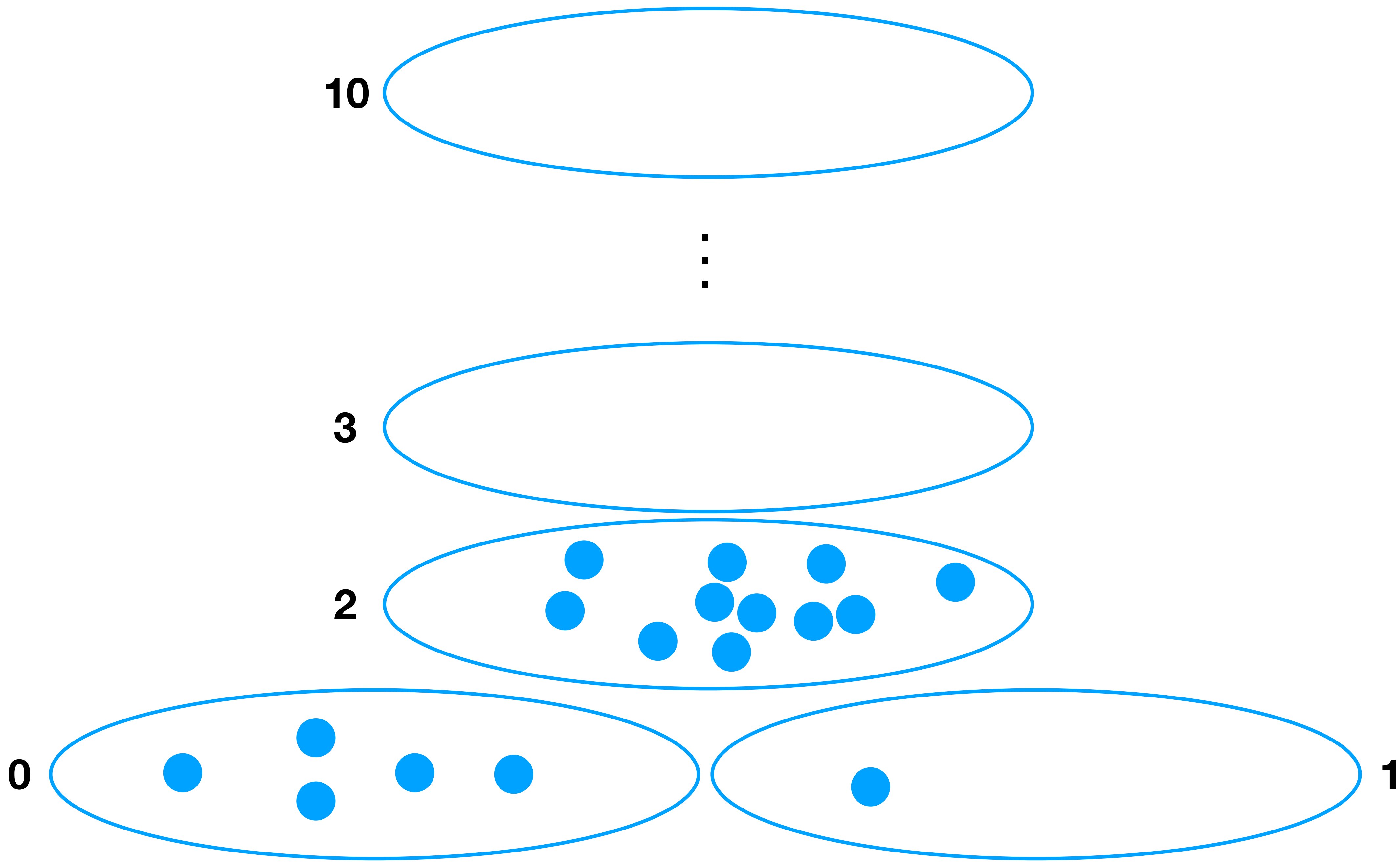
Counting Researchers Protocol



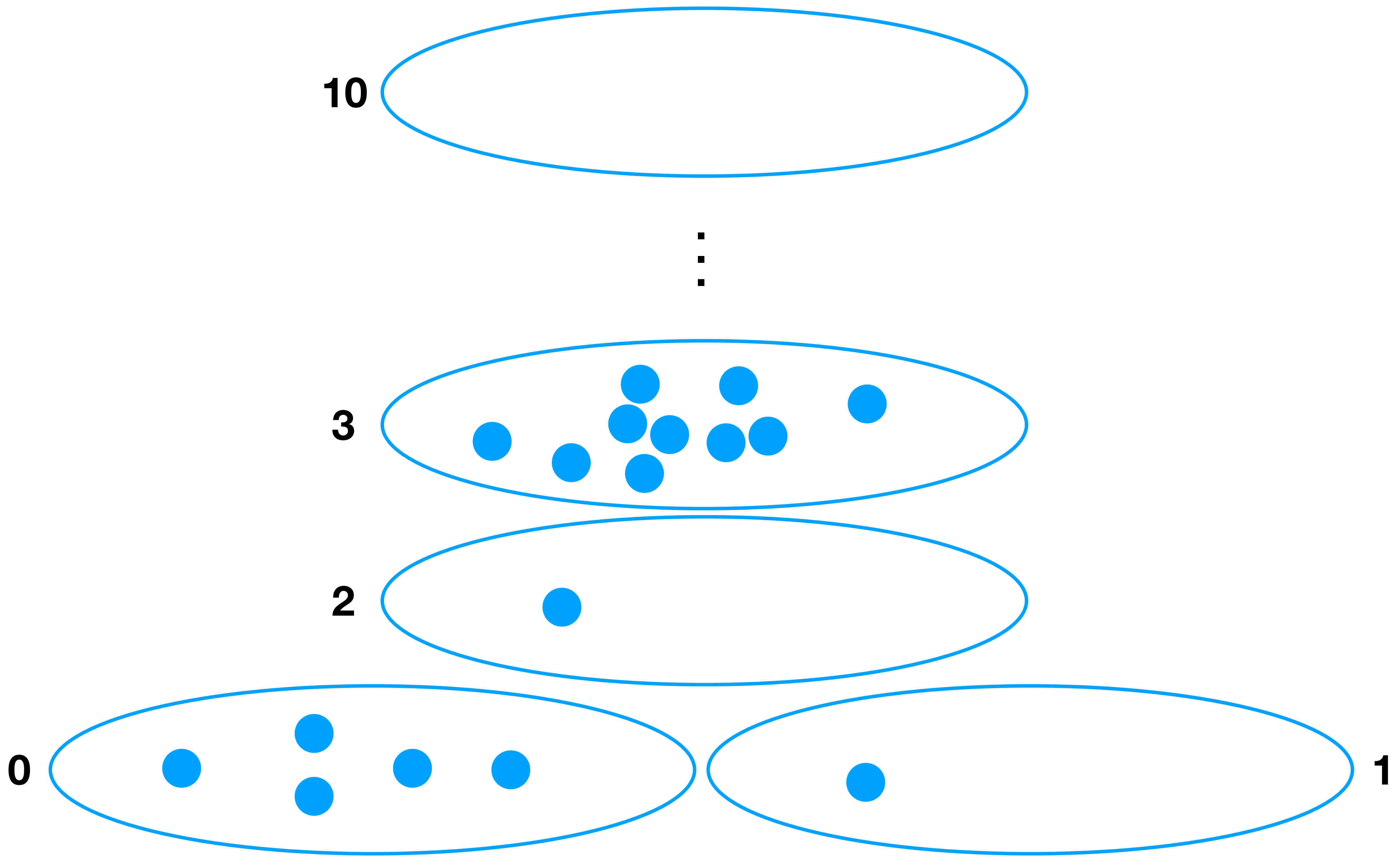
Counting Researchers Protocol



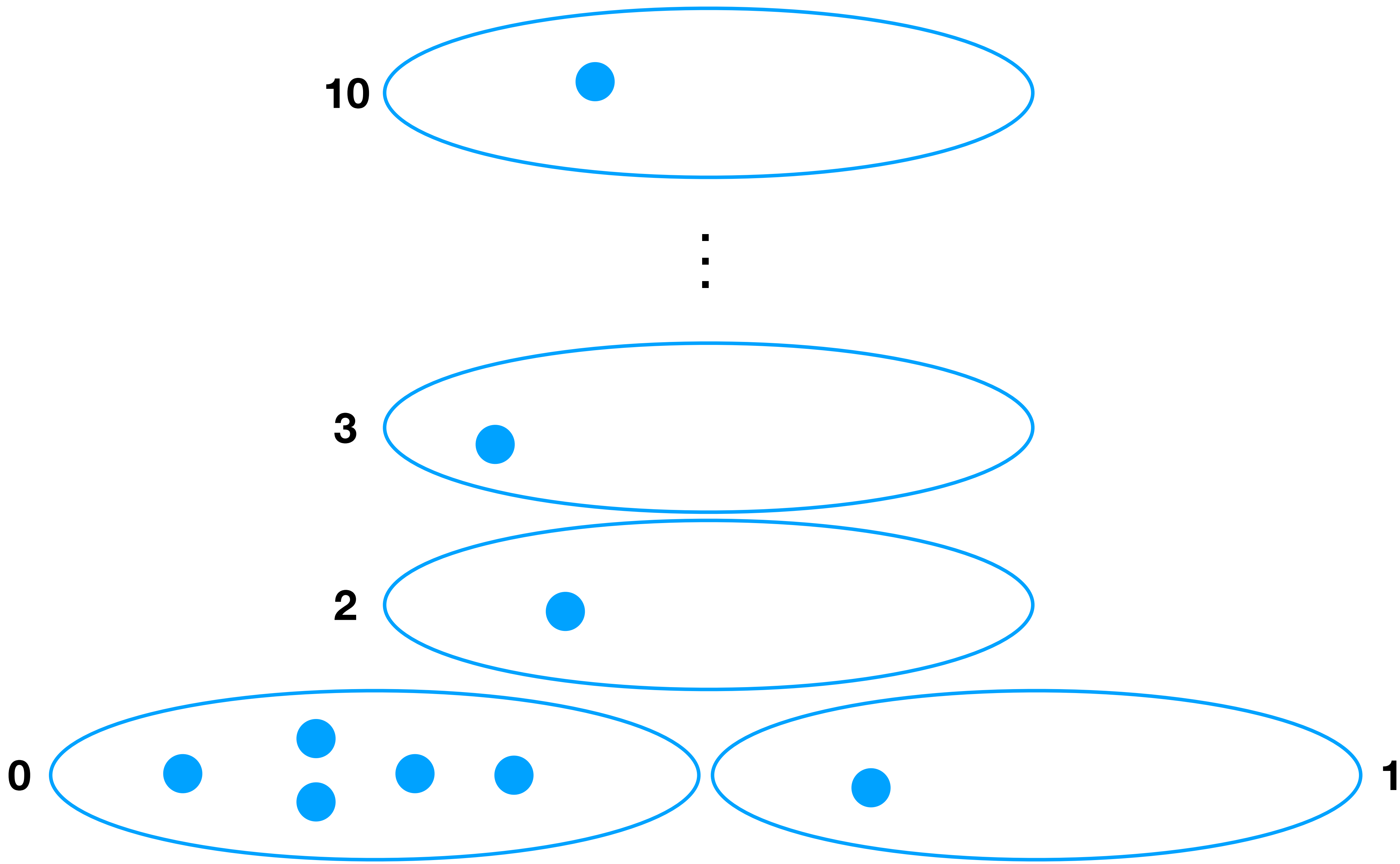
Counting Researchers Protocol



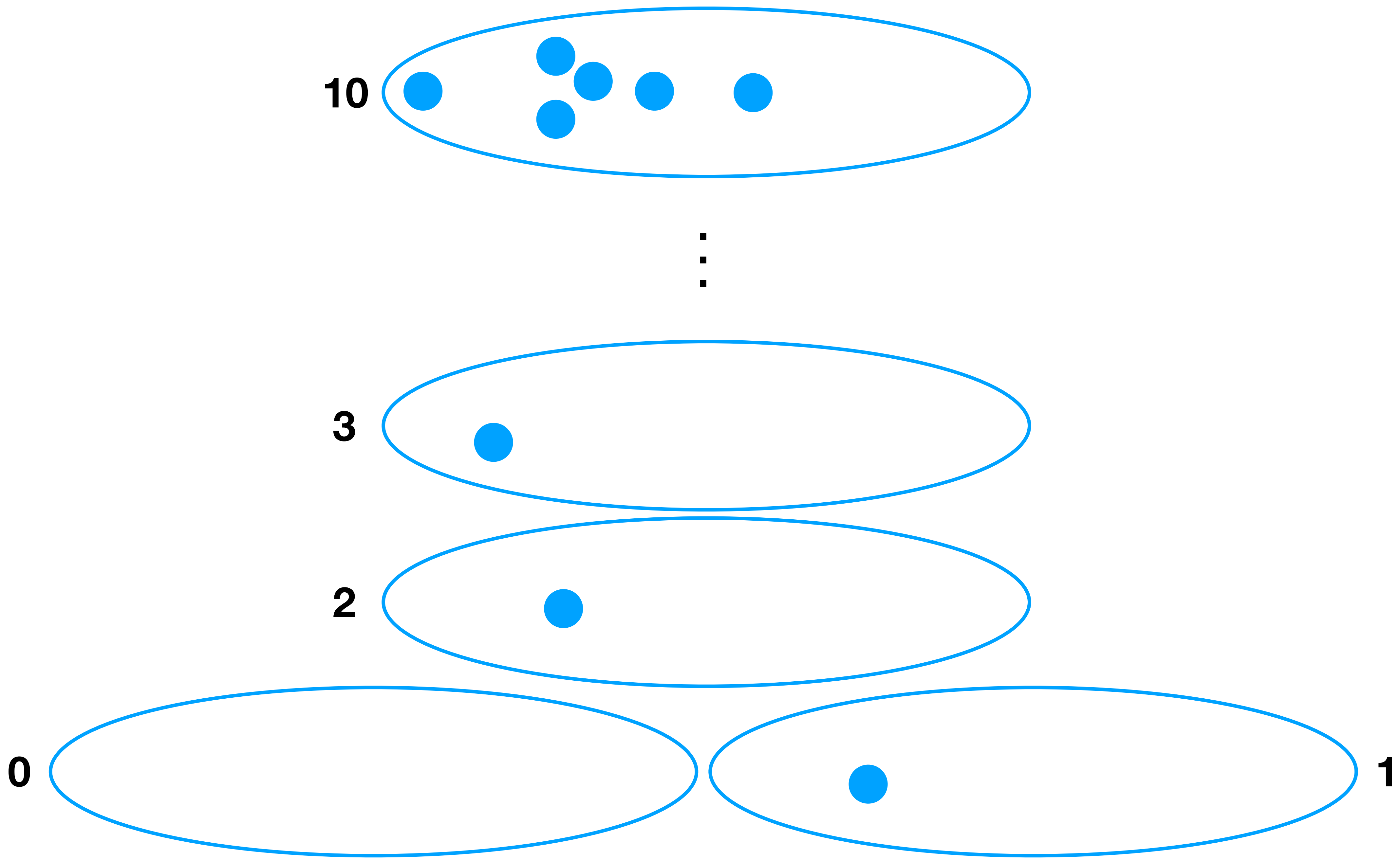
Counting Researchers Protocol



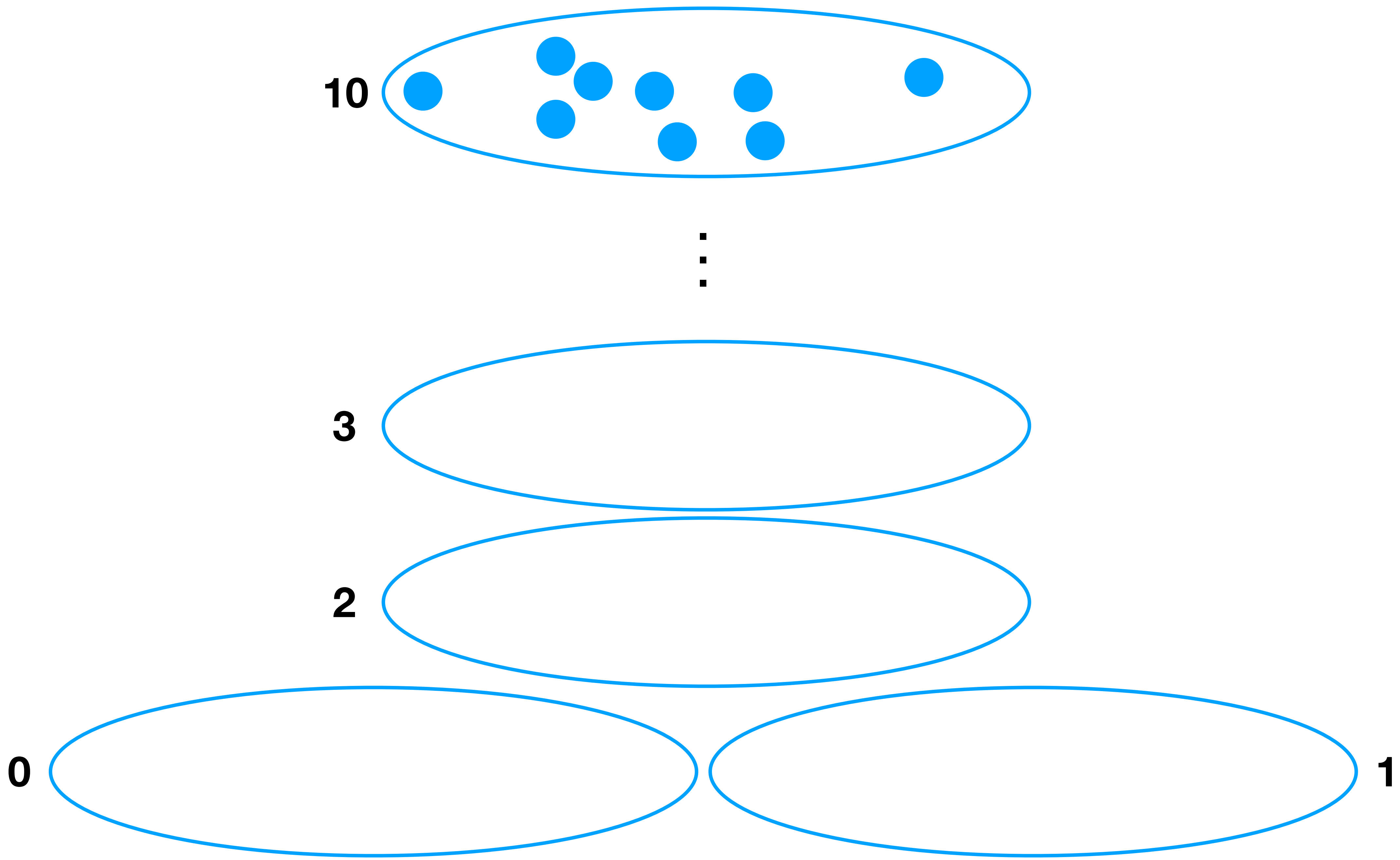
Counting Researchers Protocol



Counting Researchers Protocol



Counting Researchers Protocol

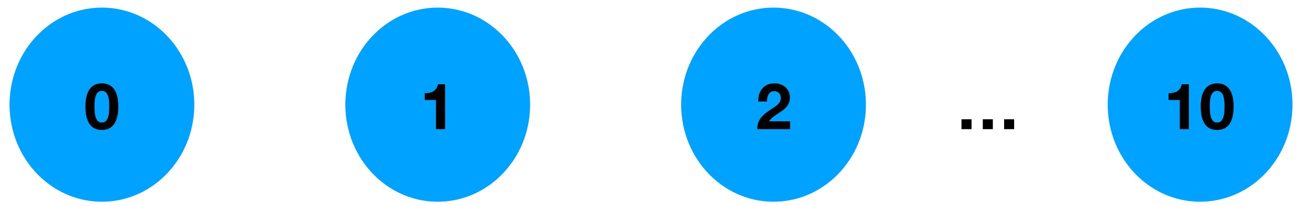


Population Protocol

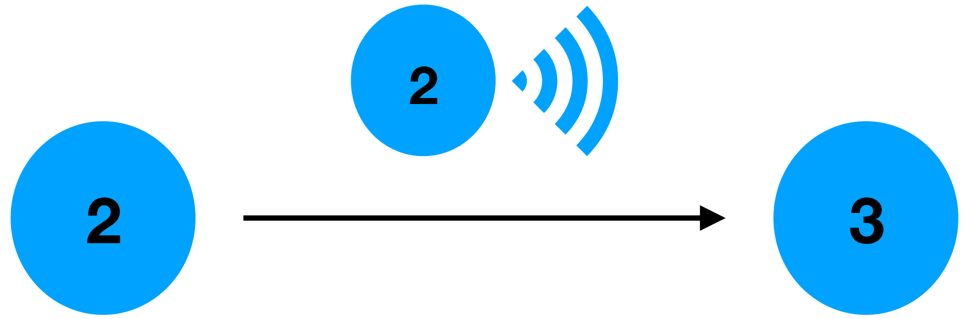
States 0 1 2 ... 10

Population Protocol

States



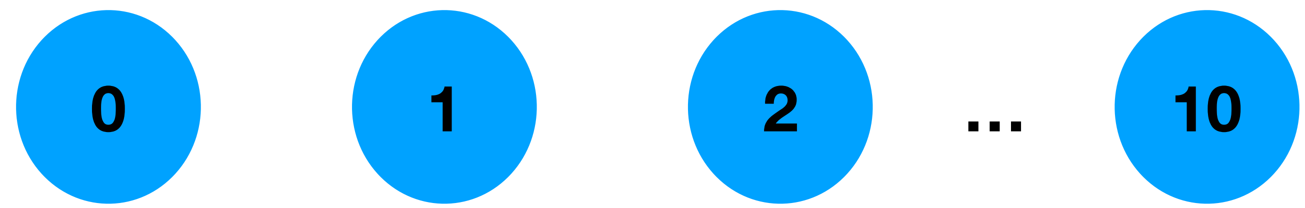
Transitions



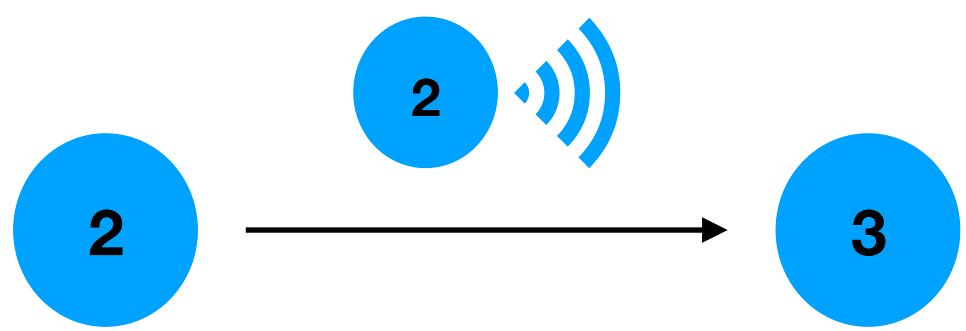
i.e. $2, 2 \mapsto 3, 2$

Population Protocol

States



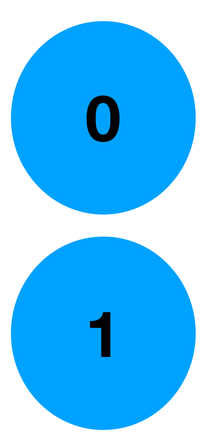
Transitions



i.e. $2, 2 \mapsto 3, 2$

Initial configurations

Only agents in



and in

Population Protocol

States 0 1 2 ... 10

Transitions 2 —(2)→ 3 i.e. $2, 2 \mapsto 3, 2$

Initial configurations

Only agents in 0

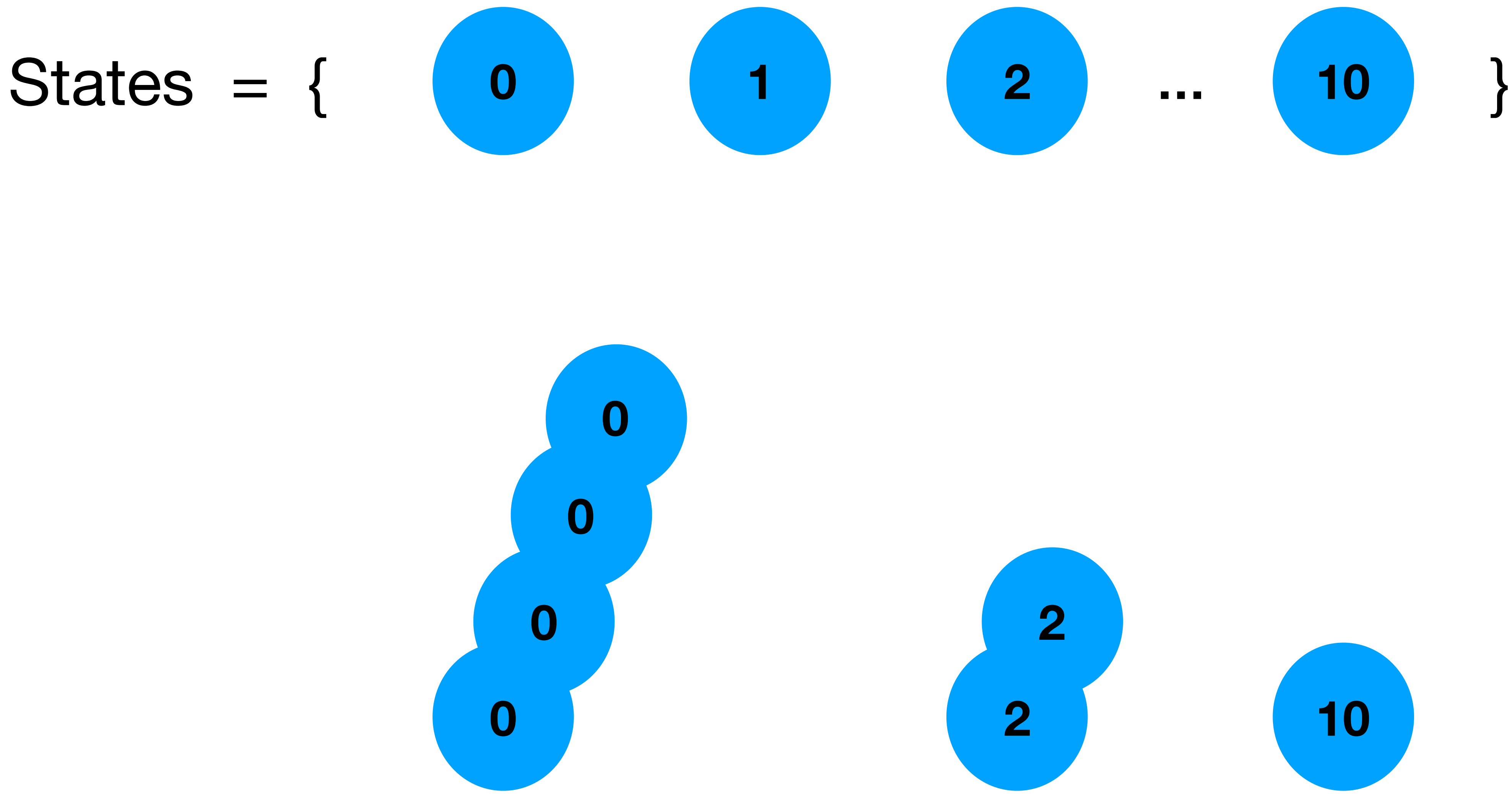
and in 1

Output function

10 → **true**

n → **false** **0 ≤ n < 10**

Configuration



(4 , 0 , 2 , … , 1)

Counting Researchers Protocol

We assume that at each step of a **run** (sequence of configurations), the two agents that interact are chosen uniformly at random

Counting Researchers Protocol

We assume that at each step of a **run** (sequence of configurations), the two agents that interact are chosen uniformly at random

This protocol has the good property :

- Runs stabilize to only one **output** (true or false)

And it is such that the output is true if and only if there are at least 10 researchers who have published at CONCUR

Counting Researchers Protocol

We assume that at each step of a **run** (sequence of configurations), the two agents that interact are chosen uniformly at random

This protocol has the good property :

- Runs stabilize to only one **output** (true or false)

And it is such that the output is true if and only if there are at least 10 researchers who have published at CONCUR

This protocol **computes** the predicate “at least 10 researchers have published at CONCUR”

Predicate Computation

Definition :

A population protocol **computes a predicate**

if and only if

every run starting in an initial configuration eventually reaches a configuration in which everyone agrees on the same output and does so forever

The Problem

Given $\mathcal{P}$ a population protocol, does $\mathcal{P}$ compute a predicate ?

EXPSPACE-hard — non primitive-recursive

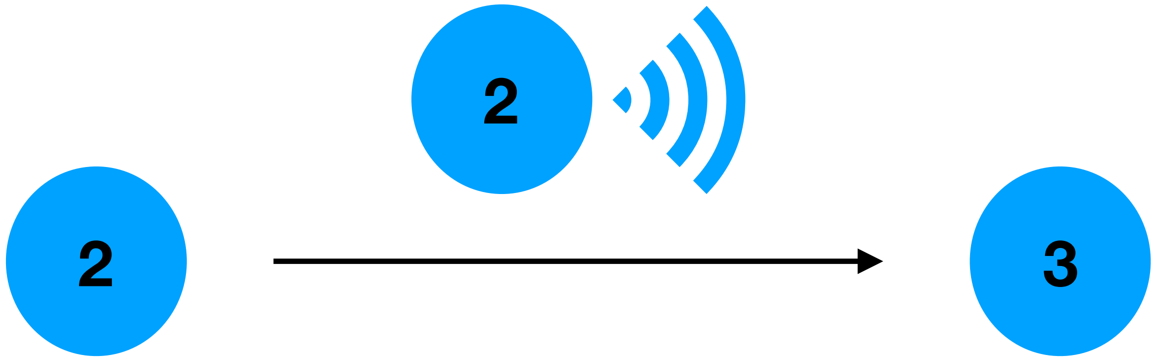
[J. Esparza, P. Ganty, J. Leroux, R. Majumdar, ’16]

Immediate Observation Population Protocols

[D. Angluin, J. Aspnes, D. Eisenstat, E. Ruppert, ’07]

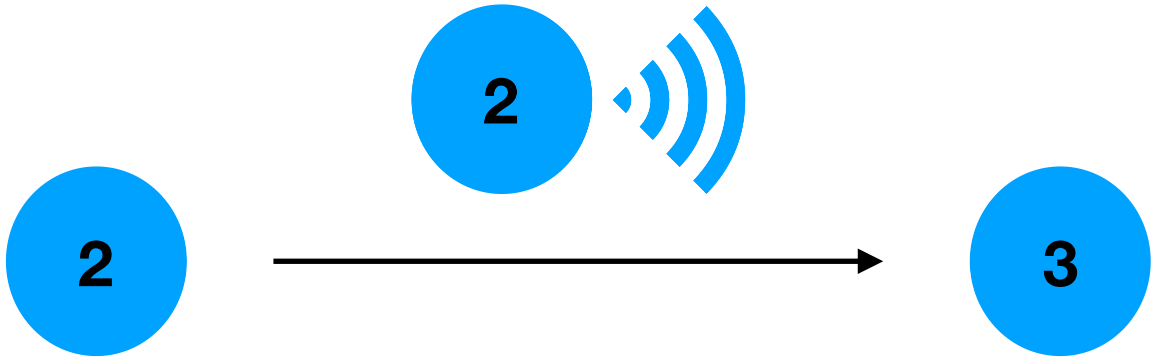
Immediate Observation Population Protocols

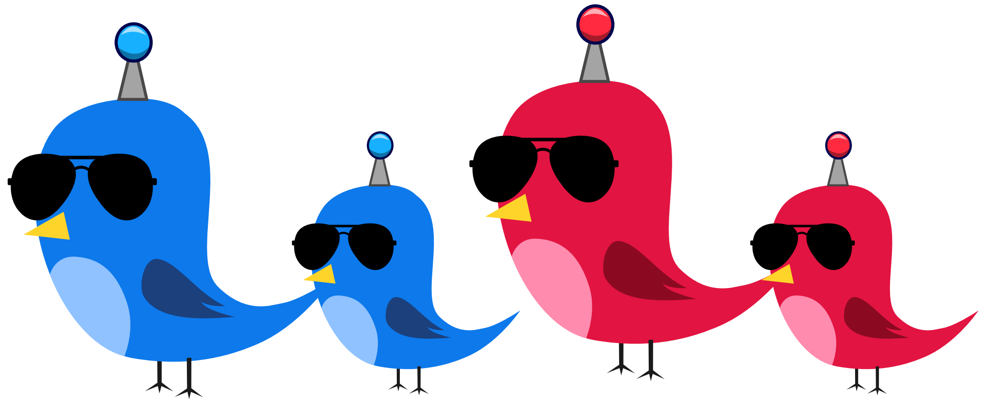
[D. Angluin, J. Aspnes, D. Eisenstat, E. Ruppert, ’07]



Immediate Observation Population Protocols

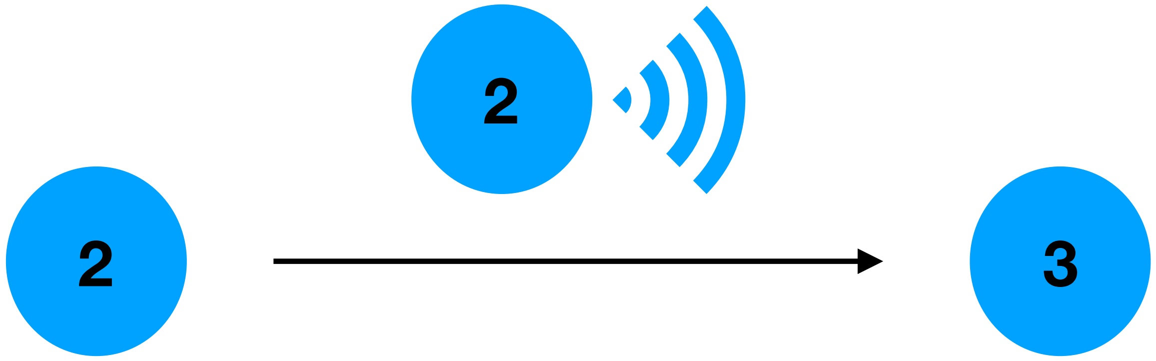
[D. Angluin, J. Aspnes, D. Eisenstat, E. Ruppert, ’07]

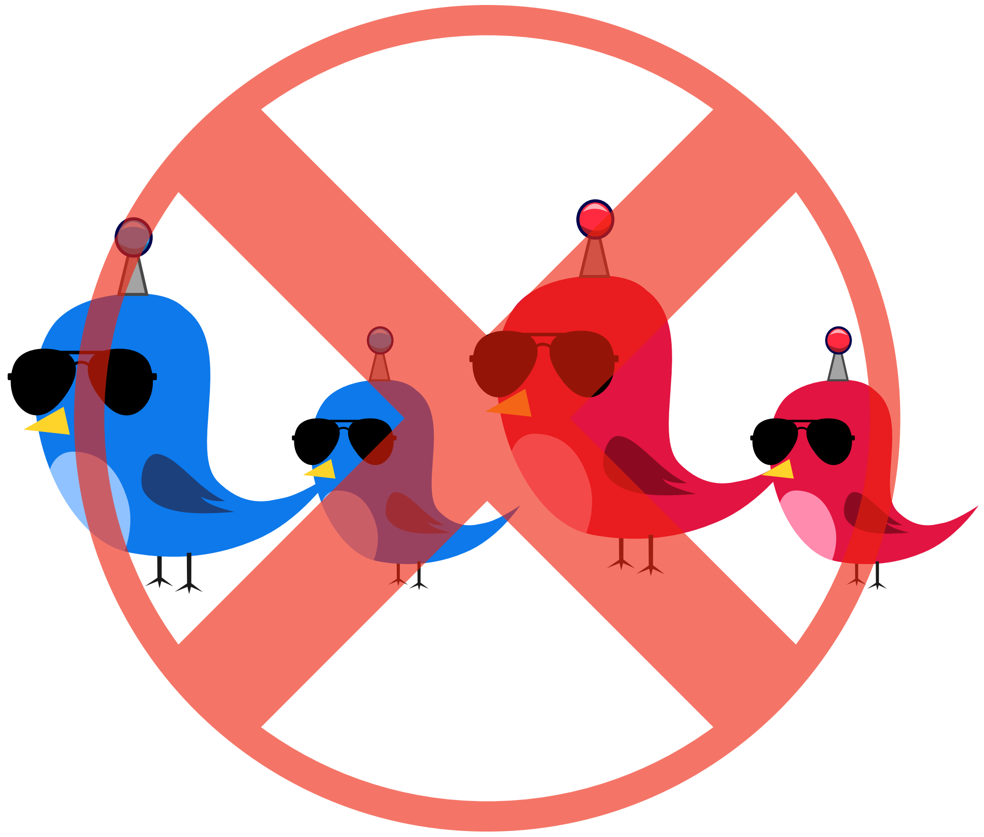




Immediate Observation Population Protocols

[D. Angluin, J. Aspnes, D. Eisenstat, E. Ruppert, ’07]





Immediate Observation Population Protocols

[D. Angluin, J. Aspnes, D. Eisenstat, E. Ruppert, ’07]

- Disadvantage of IOPP : less expressivity

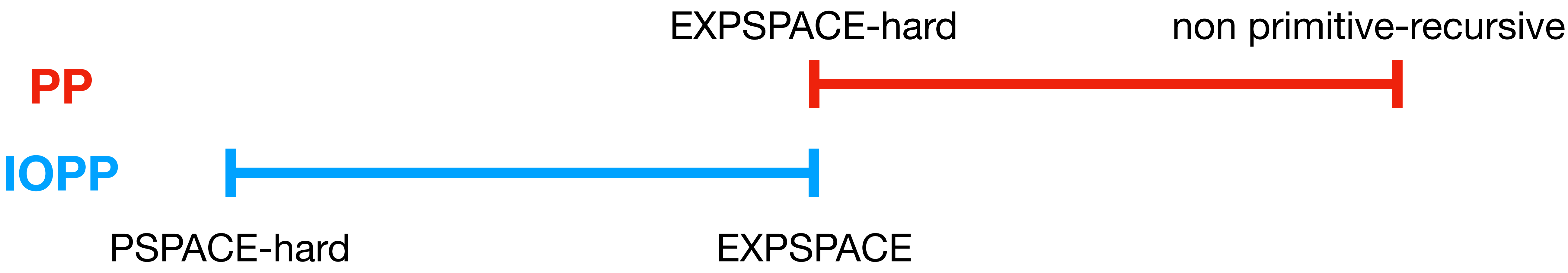
Immediate Observation Population Protocols

[D. Angluin, J. Aspnes, D. Eisenstat, E. Ruppert, '07]

- Disadvantage of IOPP : less expressivity
- Advantage of IOPP : can be implemented on top of one-way communication models
  - e.g. sensor networks
  - e.g. networks with unidirectional communication channels

Result

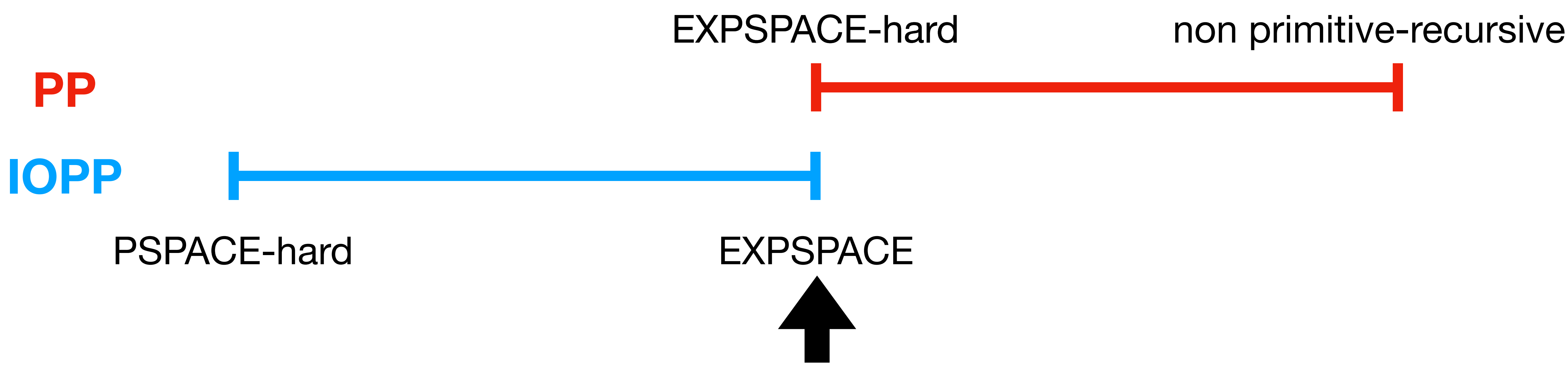
Given $\mathscr{P}$ a population protocol, does $\mathscr{P}$ compute a predicate ?



where $\mathscr{P}$ is an immediate observation population protocol (IOPP)

Result

Given $\mathcal{P}$ a population protocol, does $\mathcal{P}$ compute a predicate ?



where $\mathcal{P}$ is an immediate observation population protocol (IOPP)

Key Idea

We will :

- **Reformulate** the problem of “computing a predicate”
- Find a **good representation** for sets of configurations
- **Bound the number of iterations** needed to calculate *pre** and *post**

Reformulating the Problem

Definition :

A population protocol **computes a predicate**

if and only if

every run starting in an initial configuration eventually reaches
a configuration in which everyone agrees on the same output
and does so forever

Reformulating the Problem

Definition :

A population protocol **computes a predicate**

if and only if

every run starting in an initial configuration eventually reaches a configuration in which everyone agrees on the same output and does so forever

$$post^*(\mathcal{I}) \subseteq pre^*(\mathcal{ST}_0 \cup \mathcal{ST}_1)$$

$$\wedge$$

$$pre^*(\mathcal{ST}_0) \cap pre^*(\mathcal{ST}_1) \cap \mathcal{I} = \emptyset$$

$\mathcal{I}$ the initial configurations

$\mathcal{ST}_b$ the stable *b*-consensus configurations for *b* $\in \{0, 1\}$

Key Idea

We will :

- **Reformulate** the problem of “computing a predicate” ☑
- Find a **good representation** for sets of configurations ☐
- **Bound the number of iterations** needed to calculate *pre** and *post** ☐

Key Idea

We will :

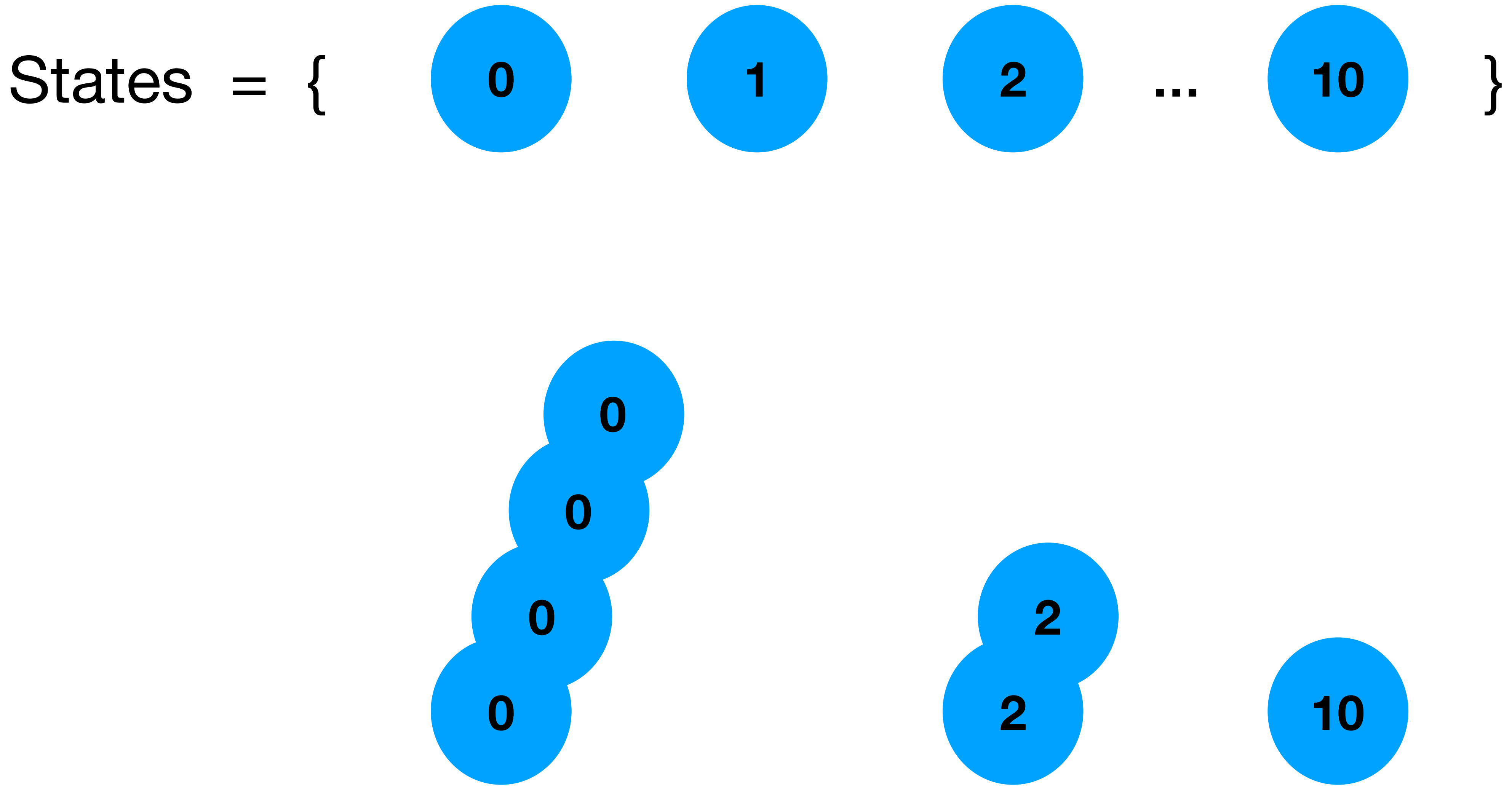
- **Reformulate** the problem of “computing a predicate” ☑
- Find a **good representation** for sets of configurations ☐
- **Bound the number of iterations** needed to calculate *pre** and *post** ☐

$$post^*(\mathcal{I}) \subseteq pre^*(\mathcal{ST}_0 \cup \mathcal{ST}_1)$$

$$\wedge$$

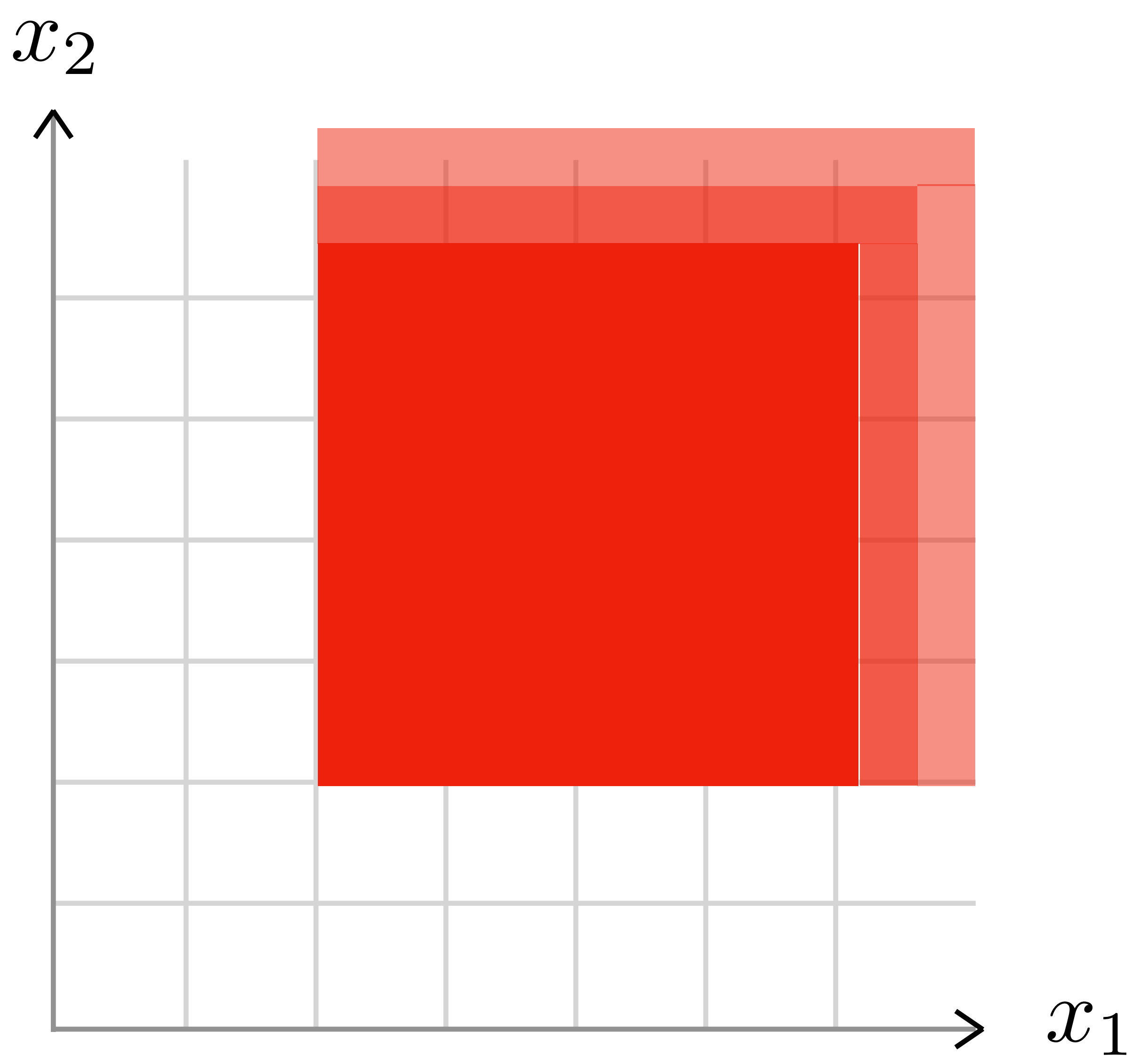
$$pre^*(\mathcal{ST}_0) \cap pre^*(\mathcal{ST}_1) \cap \mathcal{I} = \emptyset$$

Configuration



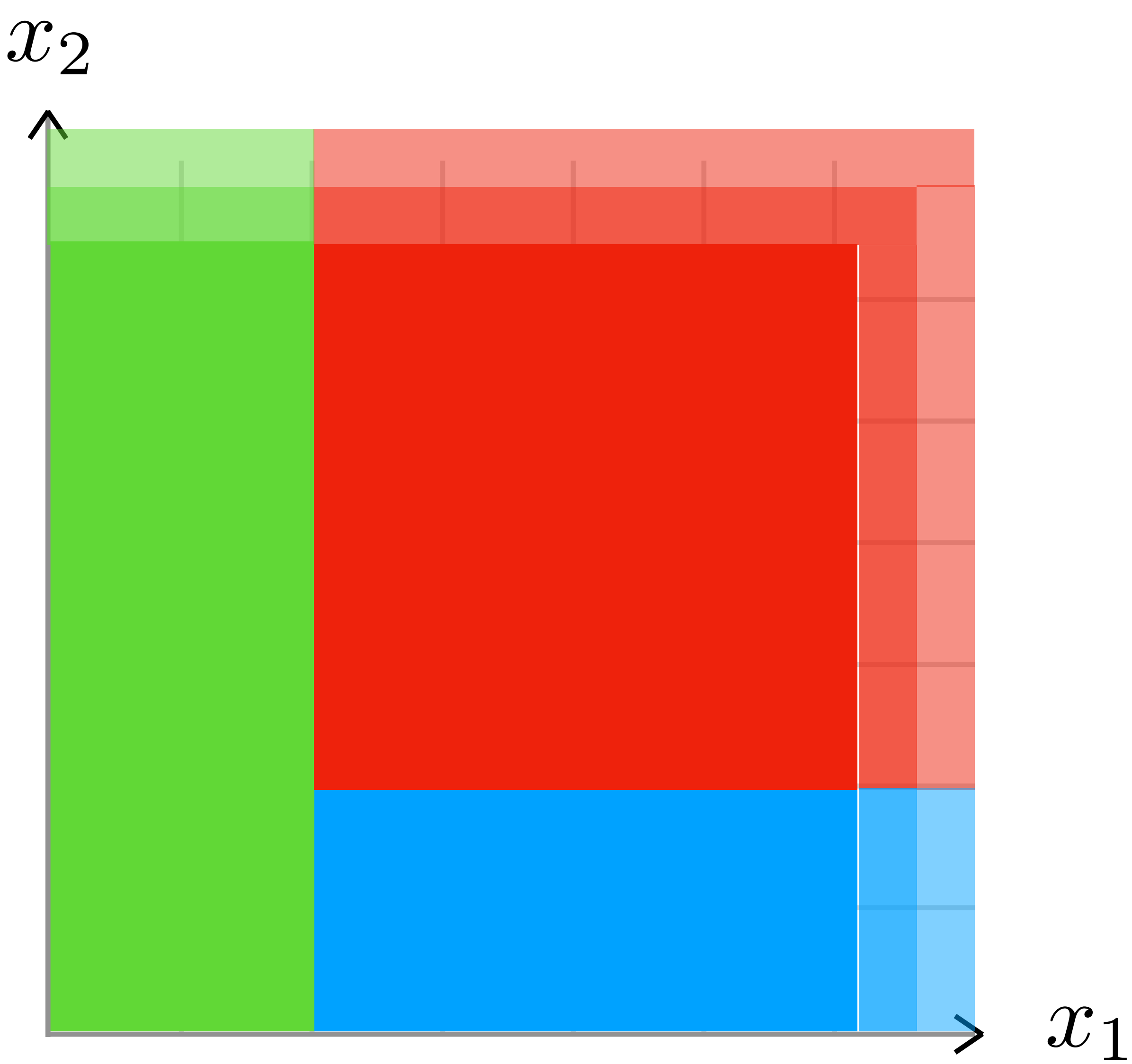
(4 , 0 , 2 , … , 1)

Counting Constraints



$$2 \leq x_1 \leq \infty$$
$$2 \leq x_2 \leq \infty$$

Counting Constraints

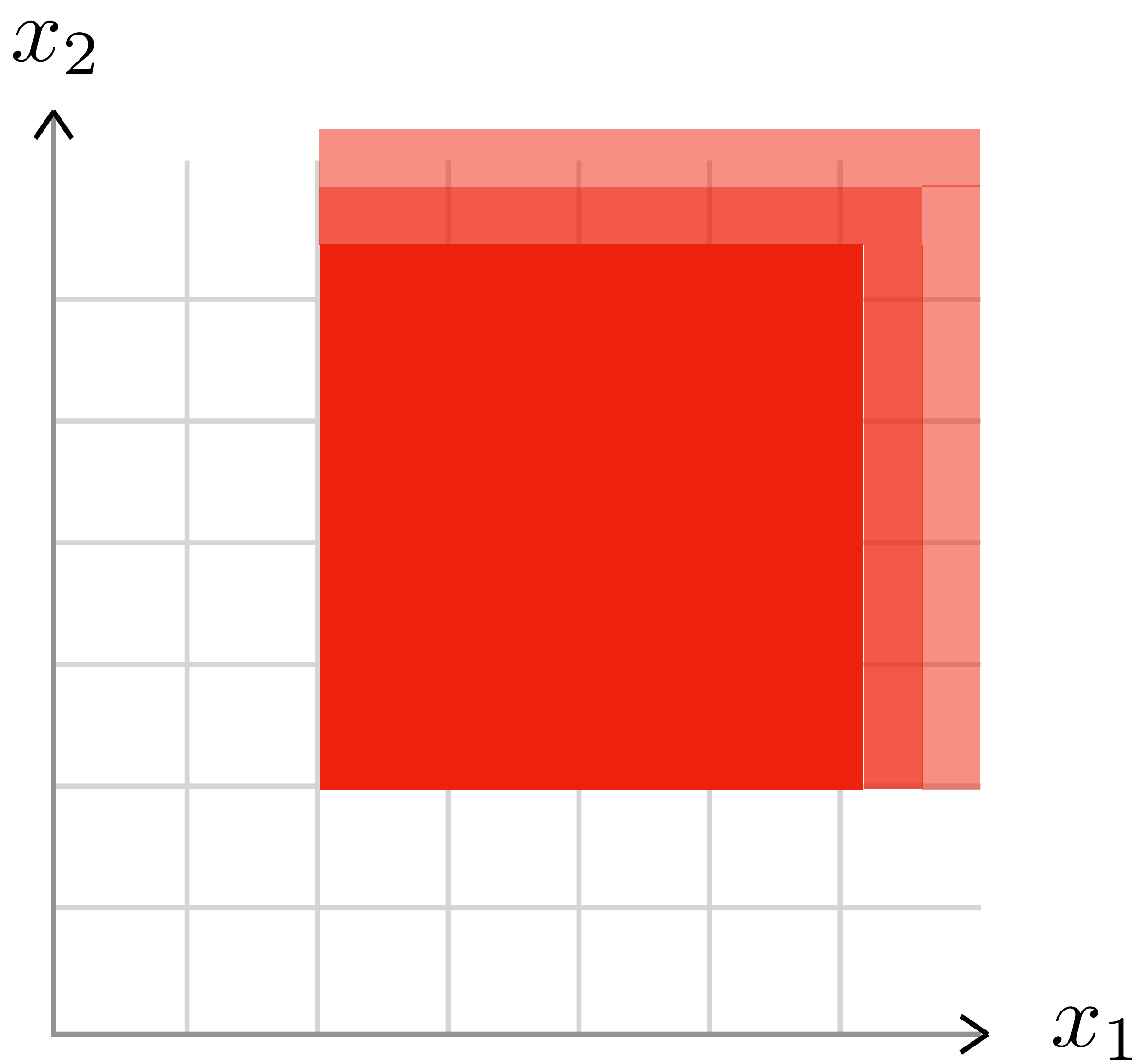


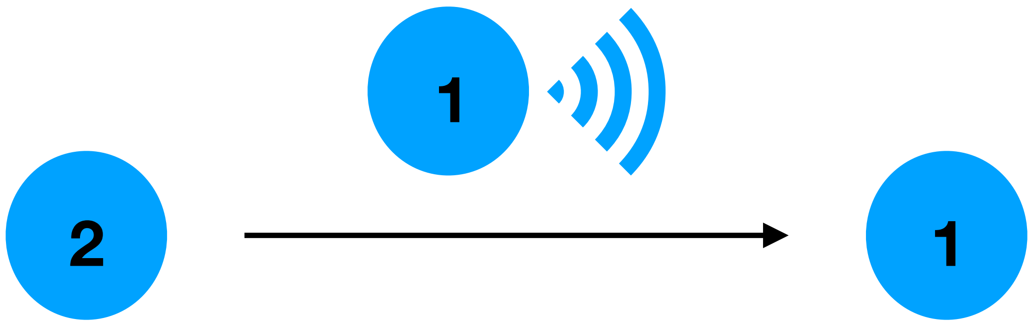
$$\neg \begin{array}{c} 2 \leq x_1 \leq \infty \\ 2 \leq x_2 \leq \infty \end{array}$$

$$\Downarrow$$

$$\begin{array}{c} 2 \leq x_1 \leq \infty \\ 0 \leq x_2 \leq 2 \end{array} \vee \begin{array}{c} 0 \leq x_1 \leq 2 \\ 0 \leq x_2 \leq \infty \end{array}$$

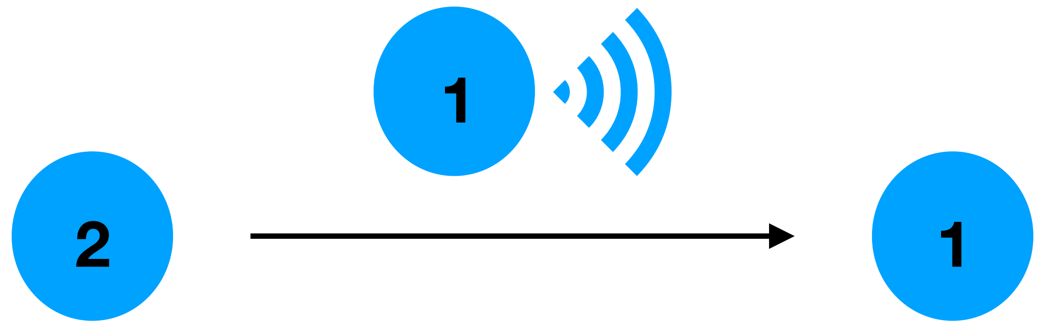
Counting Constraints

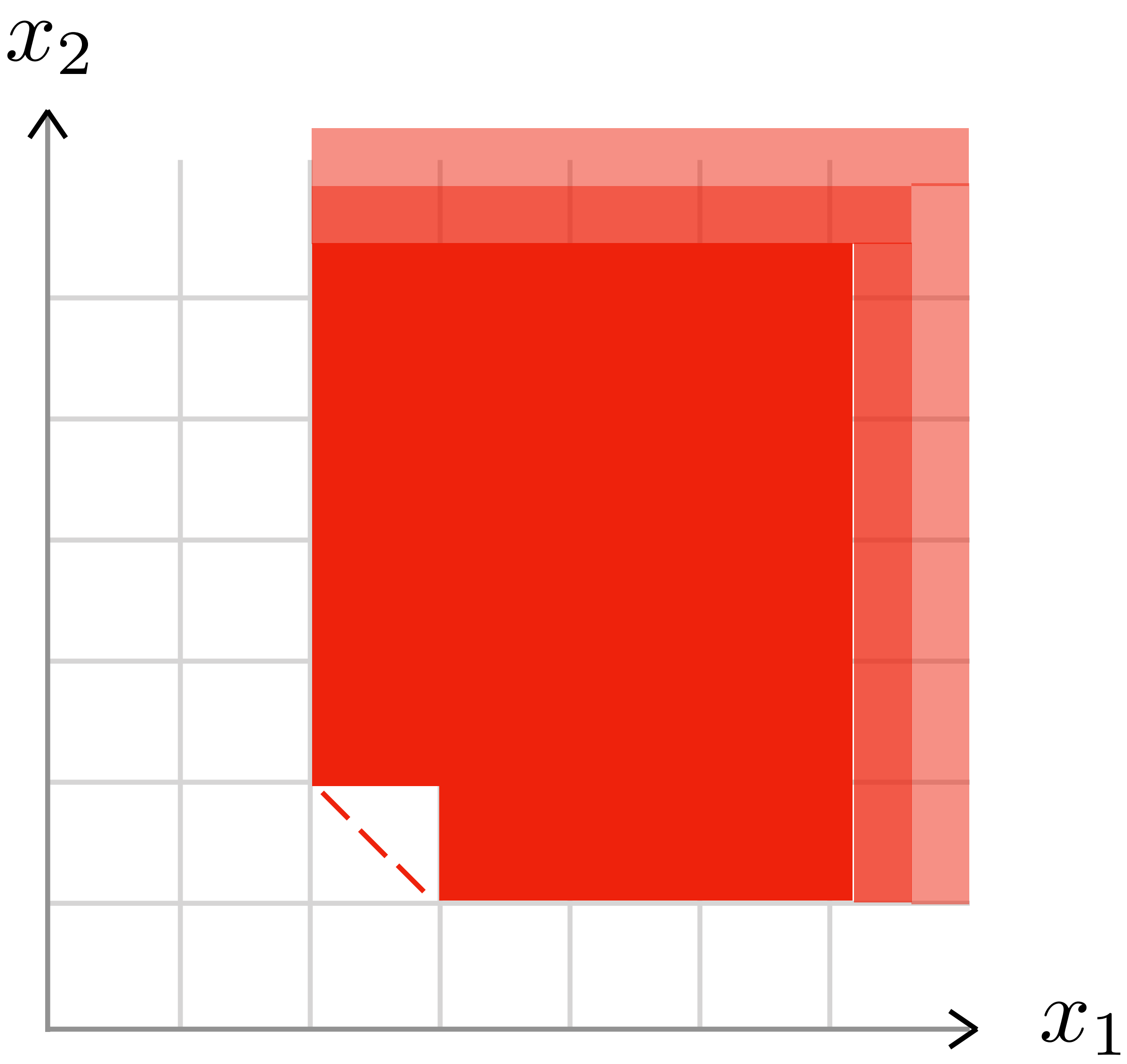




$$2 \leq x_1 \leq \infty$$
$$2 \leq x_2 \leq \infty$$

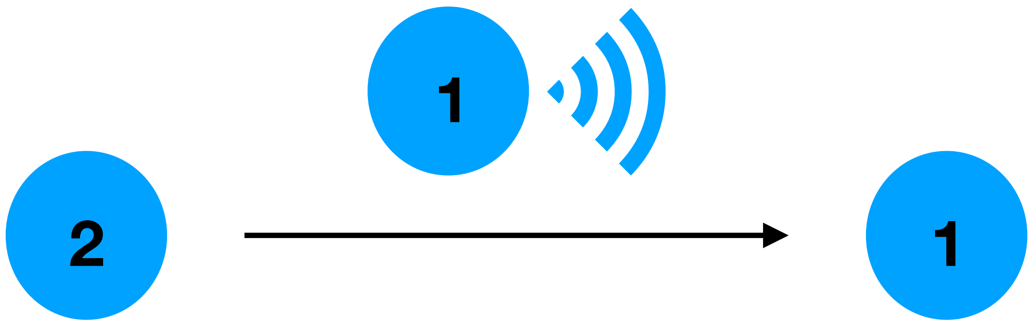
Counting Constraints

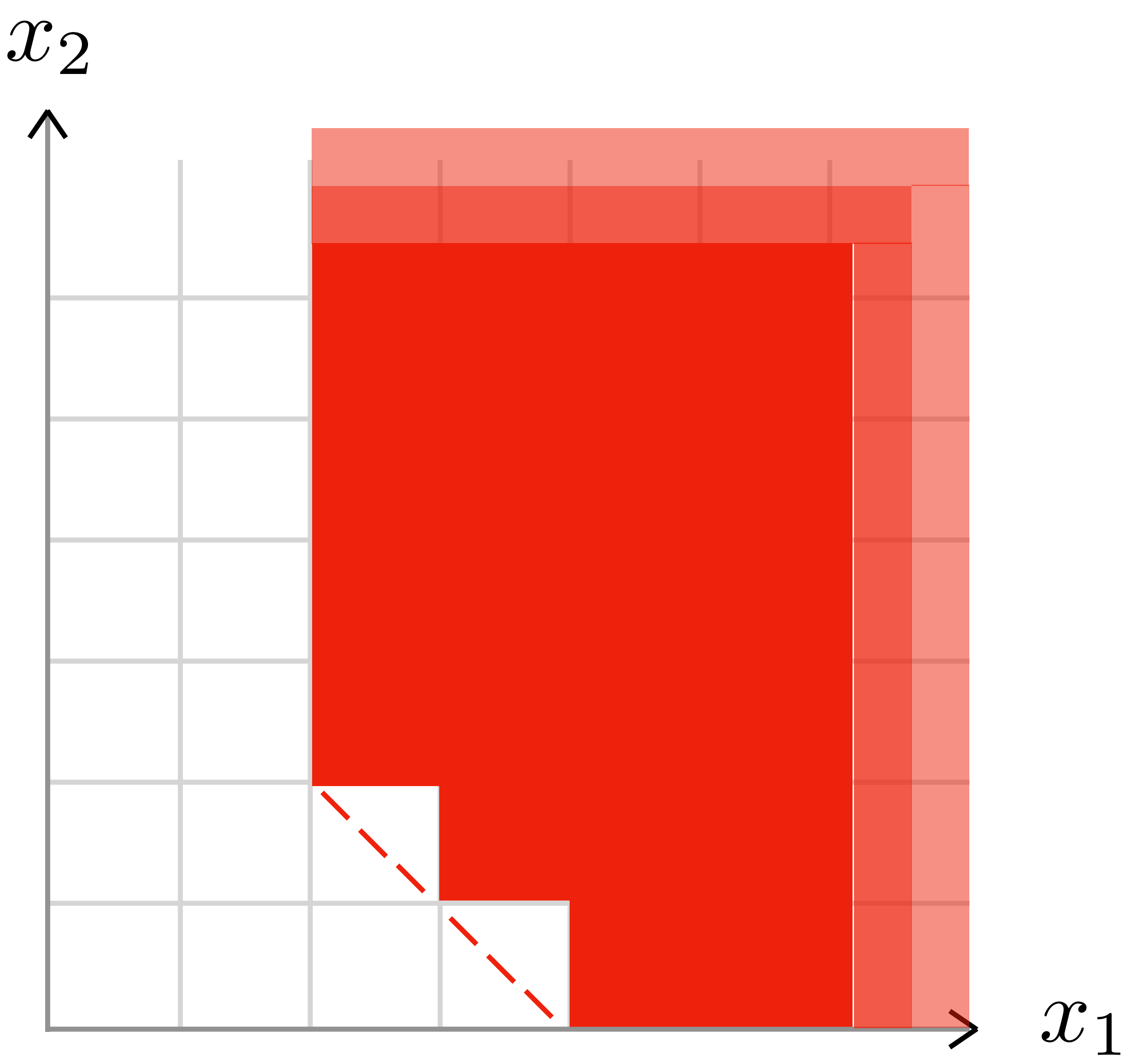




$$2 \leq x_1 \leq \infty$$
$$2 \leq x_2 \leq \infty$$
$$\Downarrow$$
$$3 \leq x_1 \leq \infty$$
$$1 \leq x_2 \leq \infty$$

Counting Constraints





$$2 \leq x_1 \leq \infty$$
$$2 \leq x_2 \leq \infty$$
$$\Downarrow$$
$$3 \leq x_1 \leq \infty$$
$$1 \leq x_2 \leq \infty$$
$$\Downarrow$$
$$4 \leq x_1 \leq \infty$$
$$0 \leq x_2 \leq \infty$$

Counting Constraints

- Counting constraints represent possibly **infinite** sets of configurations

- Counting constraints are closed under Boolean combinations

- The counting constraint representation is closed under reachability i.e. ***post**** **of a counting set is a counting set**

Counting Constraints

- Counting constraints represent possibly **infinite** sets of configurations

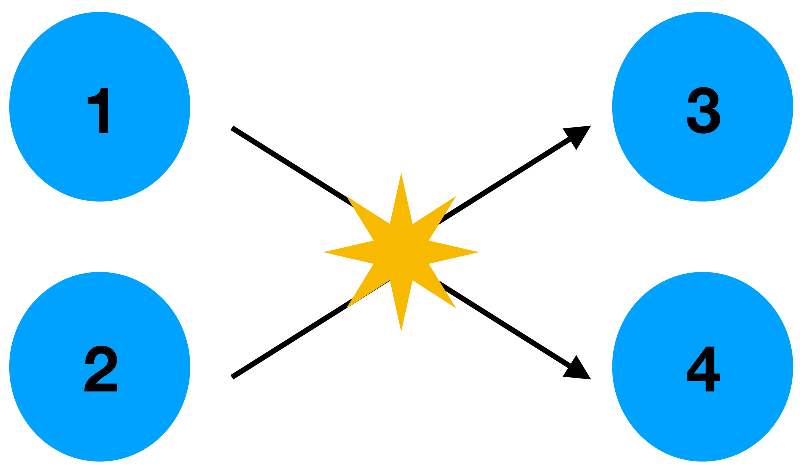
- Counting constraints are closed under Boolean combinations

- The counting constraint representation is closed under reachability i.e. ***post**** **of a counting set is a counting set**

This is the **fundamental** property for counting constraints
and it is **not true in the general population protocol framework**

Counting Constraints

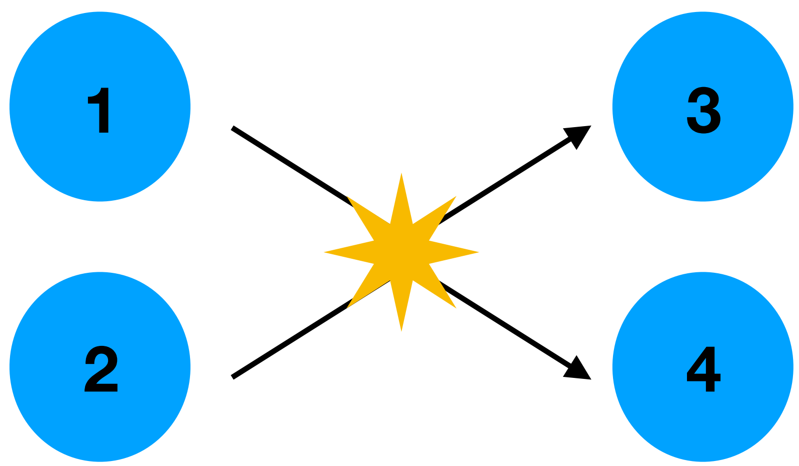
The counting constraint representation is **not** closed under reachability in the general population protocol framework



Consider a protocol with a unique transition and with no agents in state 3 and 4 in the initial configurations

Counting Constraints

The counting constraint representation is **not** closed under reachability in the general population protocol framework



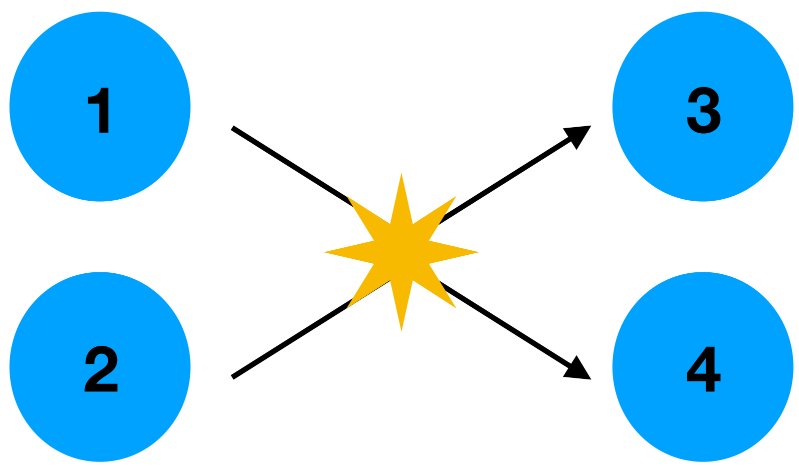
Consider a protocol with a unique transition and with no agents in state 3 and 4 in the initial configurations

$\mathcal{I} =$ configurations of the form $(n_1, n_2, 0, 0)$

$post^*(\mathcal{I}) =$ configurations of the form (n'_1, n'_2, n, n)

Counting Constraints

The counting constraint representation is **not** closed under reachability in the general population protocol framework



Consider a protocol with a unique transition and with no agents in state 3 and 4 in the initial configurations

$\mathcal{I} =$ configurations of the form $(n_1, n_2, 0, 0)$

$post^*(\mathcal{I}) =$ configurations of the form (n'_1, n'_2, n, n)

This cannot be expressed with counting constraints

Key Idea

We will :

- **Reformulate** the problem of “computing a predicate” ☑
- Find a **good representation** for sets of configurations ☑
- **Bound the number of iterations** needed to calculate *pre** and *post** ☐

Key Idea

We will :

- **Reformulate** the problem of "computing a predicate" ☑
- Find a **good representation** for sets of configurations ☑
- **Bound the number of iterations** needed to calculate *pre** and *post** ☐

$$post^*(\mathcal{I}) \subseteq pre^*(\mathcal{ST}_0 \cup \mathcal{ST}_1)$$

$$\wedge$$

$$pre^*(\mathcal{ST}_0) \cap pre^*(\mathcal{ST}_1) \cap \mathcal{I} = \emptyset$$

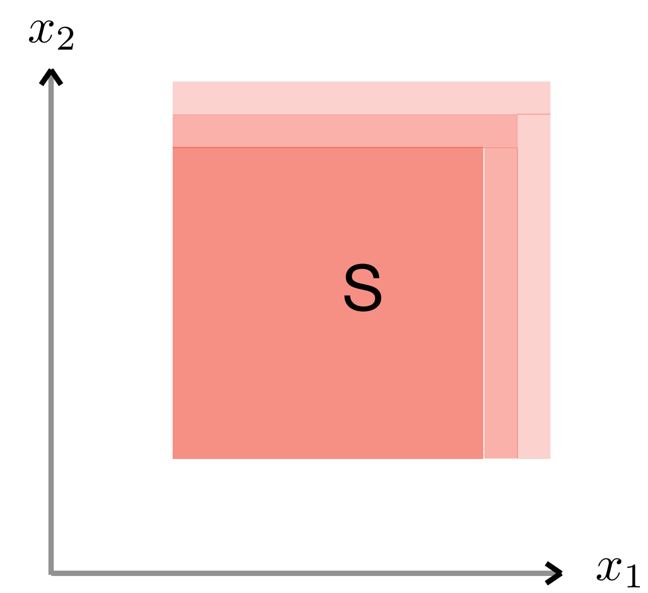
Idea

[C. Rackoff, '78]

- A theorem by Rackoff gives K such that

$$post^*(S) = \bigcup_{i \geq 0}^{K} post^i(S)$$

but only for S an **upward closed set**



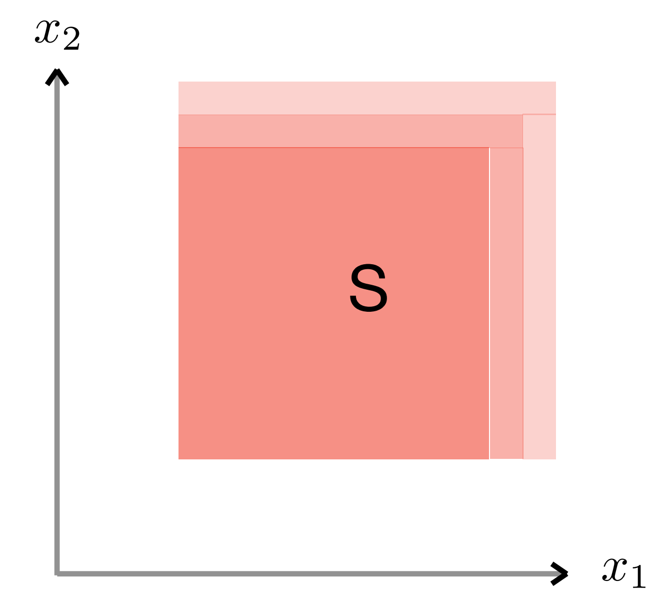
Idea

[C. Rackoff, '78]

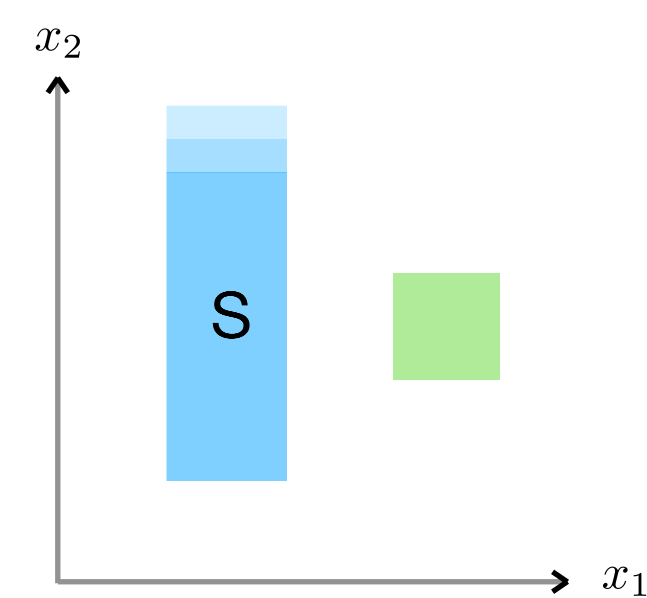
- A theorem by Rackoff gives K such that

$$post^*(S) = \bigcup_{i \geq 0}^{K} post^i(S)$$

but only for S an **upward closed set**



- We **generalize** this result by applying Rackoff a finite number of times — we split runs starting in S and apply Rackoff to each segment



Summary

We have :

- **Reformulated** the problem of “computing a predicate” ☑
- Found a **good representation** for sets of configurations ☑
- **Bounded the number of iterations** needed to calculate *pre** and *post** ☑

Summary

We have :

- **Reformulated** the problem of “computing a predicate” ✓
- Found a **good representation** for sets of configurations ✓
- **Bounded the number of iterations** needed to calculate *pre** and *post** ✓

$$post^*(\mathcal{I}) \subseteq pre^*(\mathcal{ST}_0 \cup \mathcal{ST}_1)$$

$$\wedge$$

$$pre^*(\mathcal{ST}_0) \cap pre^*(\mathcal{ST}_1) \cap \mathcal{I} = \emptyset$$

Conclusion

- We evaluate the formula in **EXPSPACE**
- Proof for **PSPACE-**hardness reduces from the acceptance problem of Turing machines running in linear space
- Future work : close the complexity gap (we are aiming for PSPACE), implement this in Peregrine

Conclusion

- We evaluate the formula $\begin{array}{c} post^*(\mathcal{I}) \subseteq pre^*(\mathcal{ST}_0 \cup \mathcal{ST}_1) \\ \wedge \\ pre^*(\mathcal{ST}_0) \cap pre^*(\mathcal{ST}_1) \cap \mathcal{I} = \emptyset \end{array}$ in **EXPSPACE**
- Proof for **PSPACE-**hardness reduces from the acceptance problem of Turing machines running in linear space
- Future work : close the complexity gap (we are aiming for PSPACE), implement this in Peregrine

Thank you !