Population Protocols with Leaders

Guided Research

What are Population Protocols?

- Models of distributed systems of agents
- Agents are:
  - Small
  - Anonymous
  - Aimlessly wandering around
- Can compute all semi-linear predicates
- Some classical distributed problems are computable:
  - Leader Election
  - Majority Voting

What are Population Protocols?

- Each agent is in one of finitely many states
- Configuration: Map states to multiplicities in the population
- States have outputs – often Boolean (here: colors)

C(Y) = 1
C(N) = 2

What are Population Protocols?

- Each agent is in one of finitely many states
- Configuration: Map states to multiplicities in the population
- States have outputs – often Boolean (here: colors)
- In each step, pairwise interaction of two agents
- Transitions: give new states for agents, depending on their old states

C(Y) = 1
C(N) = 2

$t_1 : Y, N \to y, n$
$t_2 : Y, n \to Y, y$
$t_3 : N, y \to N, n$
$t_4 : n, y \to y, y$

What are Population Protocols?

- Each agent is in one of finitely many states
- Configuration: Map states to multiplicities in the population
- States have outputs – often Boolean (here: colors)
- In each step, pairwise interaction of two agents
- Transitions: give new states for agents, depending on their old states
- Implicitly assume *silent* transition when none is given

C(Y) = 1
C(N) = 2

$$t_1 : Y, N \to y, n$$
$$t_2 : Y, n \to Y, y$$
$$t_3 : N, y \to N, n$$
$$t_4 : n, y \to y, y$$

What are Population Protocols?

- Execution: Infinite sequence of configurations

C(Y) = 1
C(N) = 2

$t_1 : Y, N \rightarrow y, n$
$t_2 : Y, n \rightarrow Y, y$
$t_3 : N, y \rightarrow N, n$
$t_4 : n, y \rightarrow y, y$

What are Population Protocols?

- Execution: Infinite sequence of configurations

C(Y) = 1
C(N) = 2

$$t_1 : Y, N \rightarrow y, n$$
$$t_2 : Y, n \rightarrow Y, y$$
$$t_3 : N, y \rightarrow N, n$$
$$t_4 : n, y \rightarrow y, y$$

What are Population Protocols?

- Execution: Infinite sequence of configurations

C(y) = 1
C(N) = 1
C(n) = 1

$t_1 : Y, N \rightarrow y, n$
$t_2 : Y, n \rightarrow Y, y$
$t_3 : N, y \rightarrow N, n$
$t_4 : n, y \rightarrow y, y$

What are Population Protocols?

- Execution: Infinite sequence of configurations
- Convergence: Eventually, all agents will have same output forever

C(N) = 1
C(n) = 2

$t_1: Y, N \rightarrow y, n$
$t_2: Y, n \rightarrow Y, y$
$t_3: N, y \rightarrow N, n$
$t_4: n, y \rightarrow y, y$

What are Population Protocols?

- Computing a predicate: always converge to right output for given initial configuration eventually

- Assume fairness:

  If during the execution, C occurs infinitely often, and from C one can reach C', then C' must occur infinitely often.

Not all Protocols are created equal

- Multiple protocols for the same predicate: Which one is better?
- In terms of size of the state space?
- In terms of expected steps until convergence?
- For expected steps, need a probability distribution that governs interactions
  - Uniform pairs: In each step, two agents are chosen uniformly at random to interact
  - Uniform rules: In each step, a transition to execute is chosen uniformly at random from all enabled transitions
- Uniform pairs sufficient for most cases

Not all Protocols are created equal

Goal: Efficient (w.r.t. space and/or time) protocols for all semi-linear predicates

Focus on fast protocols here!

Not all Protocols are created equal

Relax assumptions:

- Instead of exact protocols, use approximate ones: converge to correct result with some (hopefully large) probability
- Instead of all agents identical, use a special leader agent
  - Always exactly one leader
  - Uses separate states
  - Can be elected before the actual protocol starts (we assume the protocol starts with a leader)
  - Call non-leaders followers

Microcode Protocol

- Simulate a virtual register machine in the protocol

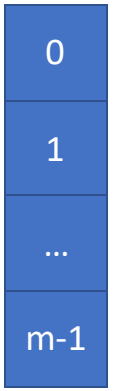
- What does a register machine need?
  - Registers: Stored among followers in unary
    - Can have values between 0 and the number of followers
  - Instruction list & Program counter: Stored in the leader

Microcode Protocol

- Leader issues an instruction, which is propagated and executed by followers
- Leader uses a phase clock to count interactions, and continue when all followers have participated in the instruction with high probability

Microcode Protocol: Phase clock

- Wait between issuing subsequent instructions by using the phase clock
- m Phases labelled $0, \ldots, m-1$



Microcode Protocol: Phase clock

- Wait between issuing subsequent instructions by using the phase clock

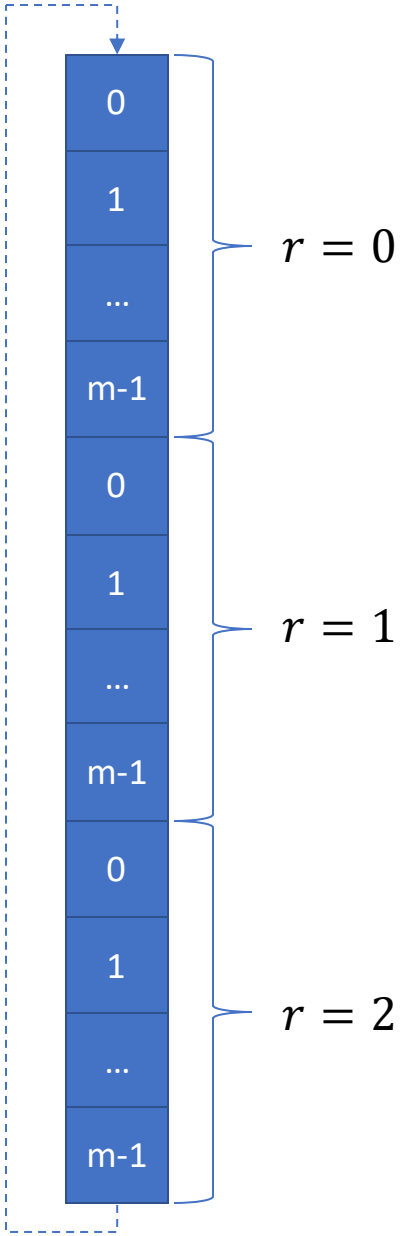
- m Phases labelled $0, \ldots, m-1$
- Each agent stores:
  - Current phase number $p \in \{0, \ldots, m-1\}$

Microcode Protocol: Phase clock

- Wait between issuing subsequent instructions by using the phase clock

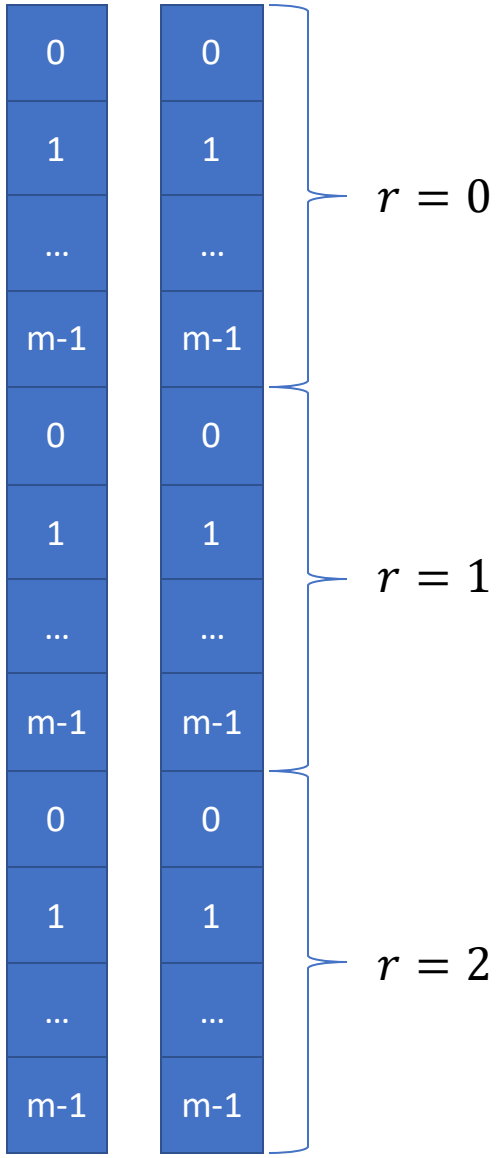
- m Phases labelled $0, \ldots, m-1$
- Each agent stores:
  - Current phase number $p \in \{0, \ldots, m-1\}$
  - Round number $r \in \{0,1,2\}$ (initially 0)

- Phases are cyclic



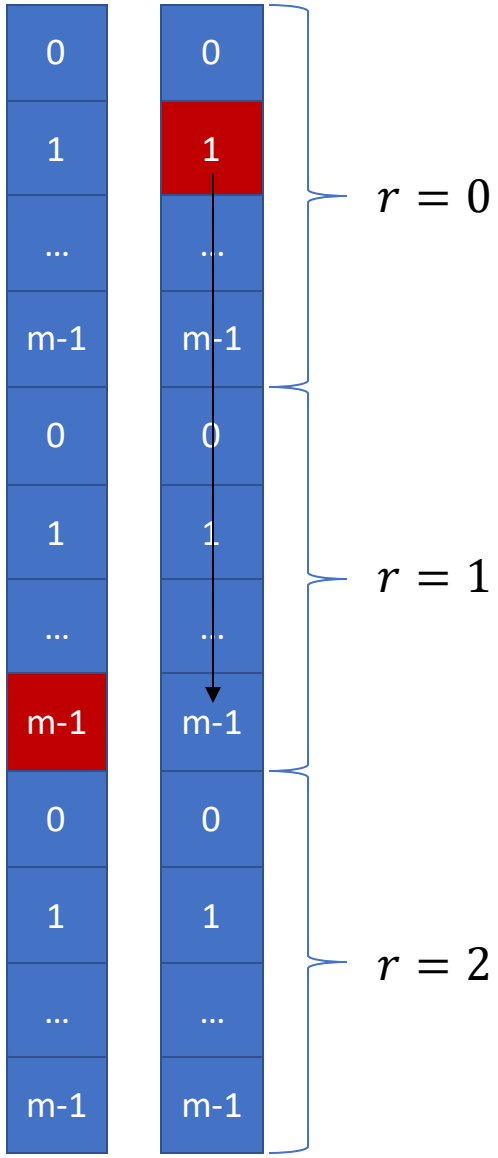
Microcode Protocol: Phase clock

- When two non-leaders meet:
  - Newer phase overwrites older one
  - Round number i is older than round number $i + 1 \; mod \; 3$



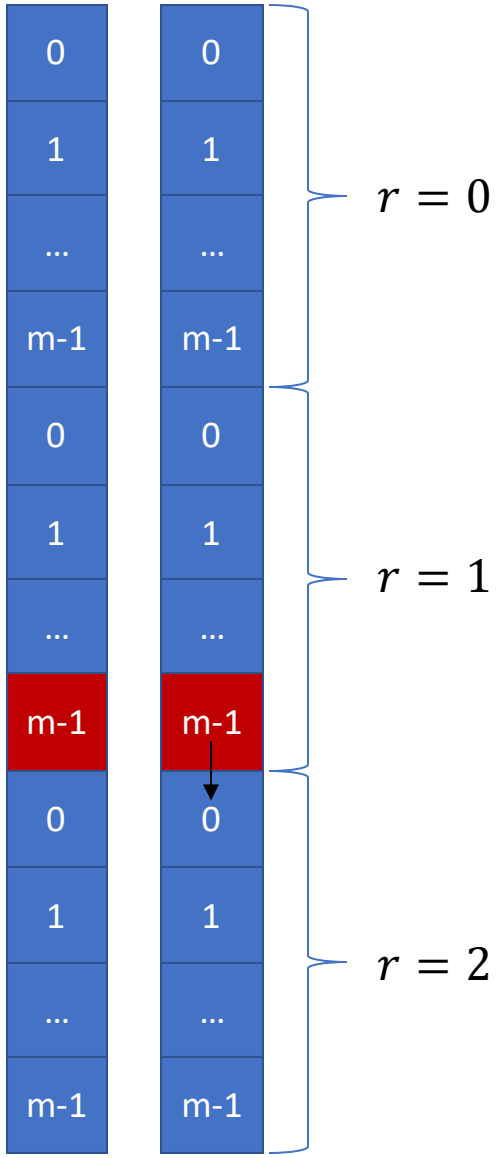
Microcode Protocol: Phase clock

- When two non-leaders meet:
  - Newer phase overwrites older one
  - Round number i is older than round number $i + 1 \; mod \; 3$



Microcode Protocol: Phase clock

- When two non-leaders meet:
  - Newer phase overwrites older one
  - Round number i is older than round number $i + 1 \; mod \; 3$
- When a leader meets a follower:
  - When both are in the same phase (and round):
    - Leader goes to next phase
    - Follower stays
  - Otherwise, leader convinces follower to its own phase



Microcode Protocol: Instructions

- Each round: execute one instruction

- Leader changes current instruction when it enters a new round
- Stores a list of instructions, together with a program counter (=current line)

- When agents meet, both execute instruction of agent with later round number

Microcode Protocol: Instructions

- Simple instructions:
  - SET(A): Set register A to 1 in all followers
  - COPY(A, B): Set register B to the value of register A in all followers
  - NOOP: No effect
  - INIT: Denotes a follower that has not yet started executing the first instruction
  - GOTOTRUE(linenumber): Sets the program counter to the indicated line
  - SETONE(A): Set register A to 1 in exactly one agent
- Registers can be negated!
- Simple instructions: when an agent starts executing the instruction, it is immediately finished
- Complex instructions: Given as protocols that agents participate in when they are executing the instruction

Microcode Protocol: Complex Instructions

- DUP(A,B): Agents participate in the duplication protocol with initial state (A,B)

- Duplication Protocol:
  - Transfer value from A to B until either A is empty or B is full

  - States: $\{(1,1), (0,1), (0,0)\}$
    - Agents can also start with $(1,0)$ which is immediately swapped to $(0,1)$
  - Transitions:
    - $(1,1), (0,0) \rightarrow (0,1), (0,1)$
    - $(0,0), (1,1) \rightarrow (0,1), (0,1)$

Microcode Protocol: Complex Instructions

- CANCEL(A,B): Agents participate in the cancellation protocol with initial state (A,B)

- Cancellation Protocol:
  - Decrease A and B simultaneously until one is empty

  - States: $\{(1,0), (0,1), (0,0)\}$
    - Agents can also start with $(1,1)$ which is immediately swapped to $(0,0)$
  - Transitions:
    - $(1,0), (0,1) \rightarrow (0,0), (0,0)$
    - $(0,1), (1,0) \rightarrow (0,0), (0,0)$

Microcode Protocol: Complex Instructions

- PROBE(p, linenumber): Agents participate in the probing protocol

- Probing Protocol:
  - If any agent satisfies the predicate p, the leader changes the prorgram counter to the given line number
  - Typically tests if a register is non-zero
  - Uses an additional component as state: probing state
  - States: $\{0,1,2\}$
  - Initial States:
    - Leader: State 1
    - Followers: State 0
  - Transitions:
    - $x, y \to 2,2$ if either agent satisfies the predicate
    - $x, y \to \max(x, y), \max(x, y)$ if neither agent satisfies the predicate
  - When run to convergence, all agents are either in state 1 (if no agent satisfies the predicate) or state 2 (if any agent satisfies the predicate)

Microcode Protocol: Example

State explosion

- Incrementing by one: 4 registers, 4 instructions: ~ 508 states, $\sim 248k$ transitions
- Comparison: 9 registers, 32 instructions: $\sim 16k$ states, $\sim 10^8$ transitions
- Explicitly giving the state space for simulations does not work!

Simulating Protocols

- Store current configuration in a data structure
- Two main operations in each step:
  - Choose two agents to interact uniformly at random
  - Update multiplicities
- How to best store configurations?

Counting Vectors

- Store configuration as a hash map H from states to multiplicities
- Random Choice:

Algorithm 1 Uniform Random Choice Algorithm for Counting Vectors

Given: A hash table H mapping states to multiplicities

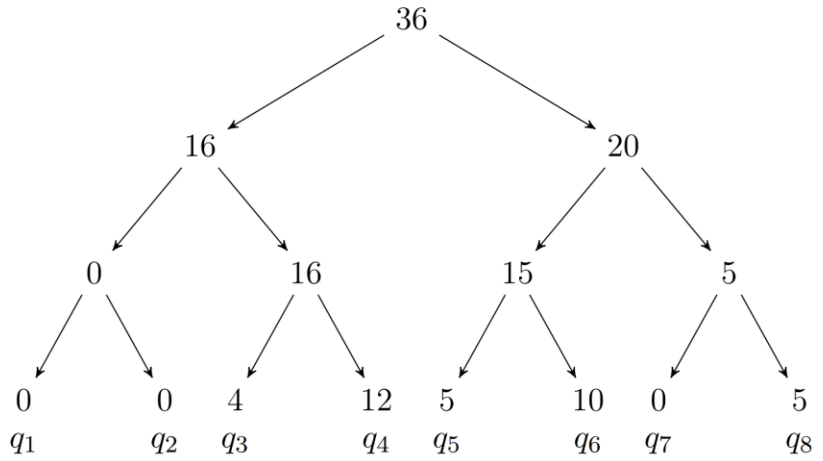
```
1: c ← UniformChoice(0, n − 1)
2: for s in S do
3:     if c < H[s] then
4:         return s
5:     else
6:         c ← c − H[s]
7:     end if
8: end for
```

  - $O(|S|)$ expected steps
- Updates:
  - Delete old value, insert new value
  - $O(1)$ steps

Segment Tree

- Leaf nodes: States with multiplicities
- Parent: Stores sum of children

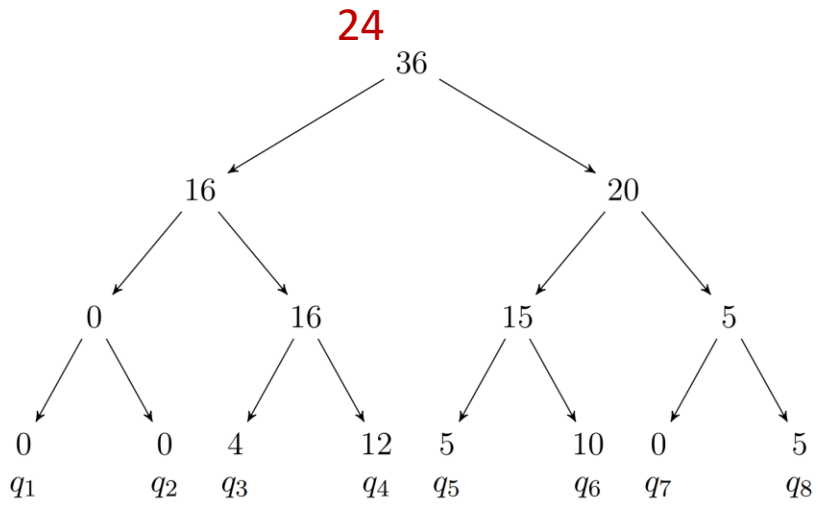
- Random Choice:
  - Choose a C between 0 and the number of agents
  - Start at root node, and compare C to the number of the left child l
    - If smaller, go to left child
    - If greater, go to right child and decrease C by l
    - Repeat until arriving at a leaf node
  - $O(\log|S|)$ steps



Segment Tree

- Leaf nodes: States with multiplicities
- Parent: Stores sum of children

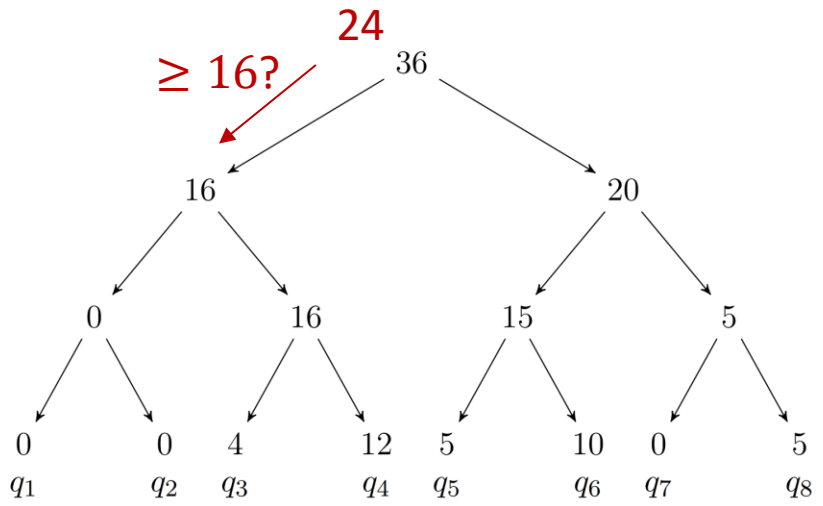
- Random Choice:
  - Choose a C between 0 and the number of agents
  - Start at root node, and compare C to the number of the left child l
    - If smaller, go to left child
    - If greater, go to right child and decrease C by l
    - Repeat until arriving at a leaf node
  - $O(\log|S|)$ steps



Segment Tree

- Leaf nodes: States with multiplicities
- Parent: Stores sum of children

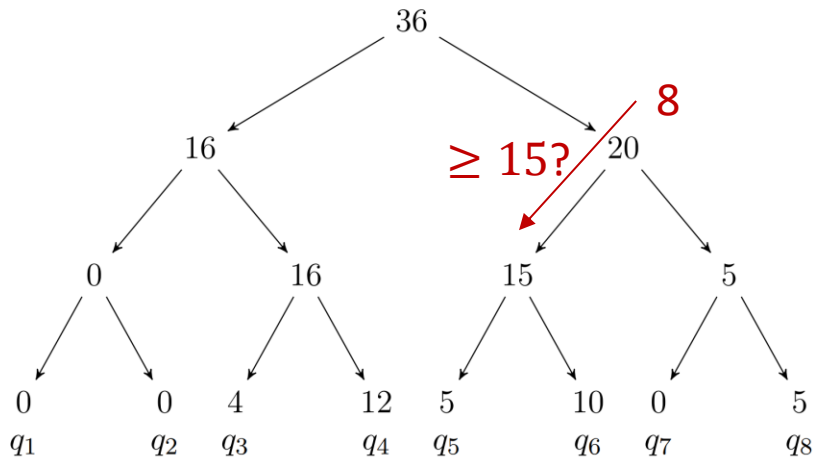
- Random Choice:
  - Choose a C between 0 and the number of agents
  - Start at root node, and compare C to the number of the left child l
    - If smaller, go to left child
    - If greater, go to right child and decrease C by l
    - Repeat until arriving at a leaf node
  - $O(\log |S|)$ steps



Segment Tree

- Leaf nodes: States with multiplicities
- Parent: Stores sum of children

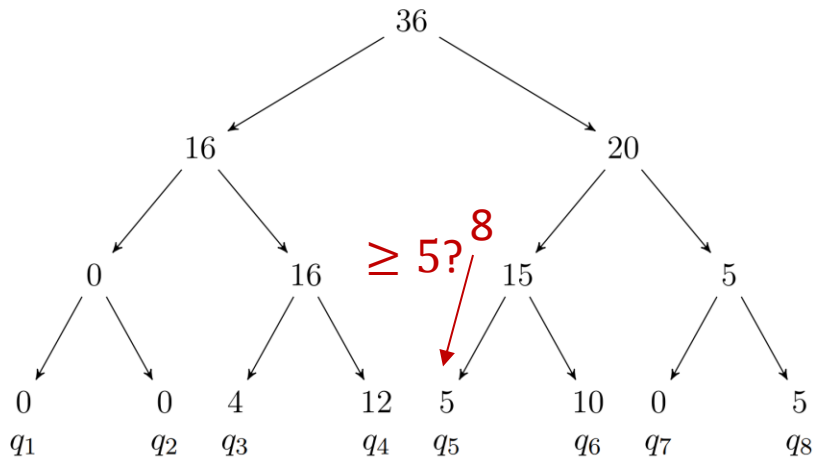
- Random Choice:
  - Choose a C between 0 and the number of agents
  - Start at root node, and compare C to the number of the left child l
    - If smaller, go to left child
    - If greater, go to right child and decrease C by l
    - Repeat until arriving at a leaf node
  - $O(\log|S|)$ steps



Segment Tree

- Leaf nodes: States with multiplicities
- Parent: Stores sum of children

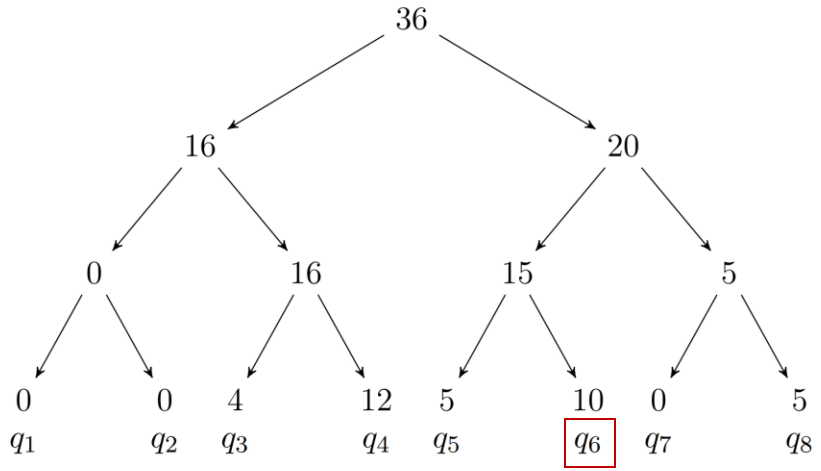
- Random Choice:
  - Choose a C between 0 and the number of agents
  - Start at root node, and compare C to the number of the left child l
    - If smaller, go to left child
    - If not, go to right child and decrease C by l
    - Repeat until arriving at a leaf node
  - $O(\log |S|)$ steps



Segment Tree

- Leaf nodes: States with multiplicities
- Parent: Stores sum of children

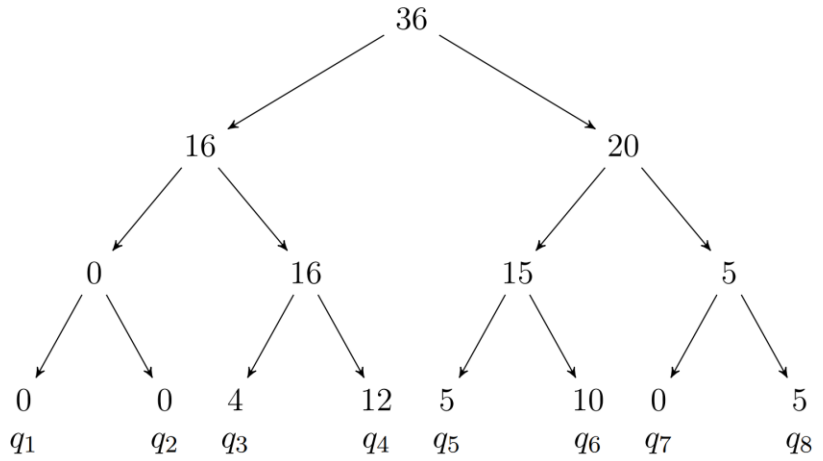
- Random Choice:
  - Choose a C between 0 and the number of agents
  - Start at root node, and compare C to the number of the left child l
    - If smaller, go to left child
    - If greater, go to right child and decrease C by l
    - Repeat until arriving at a leaf node
  - $O(\log|S|)$ steps



Segment Tree

- Leaf nodes: States with multiplicities
- Parent: Stores sum of children

- Random Choice:
  - Choose a C between 0 and the number of agents
  - Start at root node, and compare C to the number of the left child l
    - If smaller, go to left child
    - If greater, go to right child and decrease C by l
    - Repeat until arriving at a leaf node
  - $O(\log|S|)$ steps

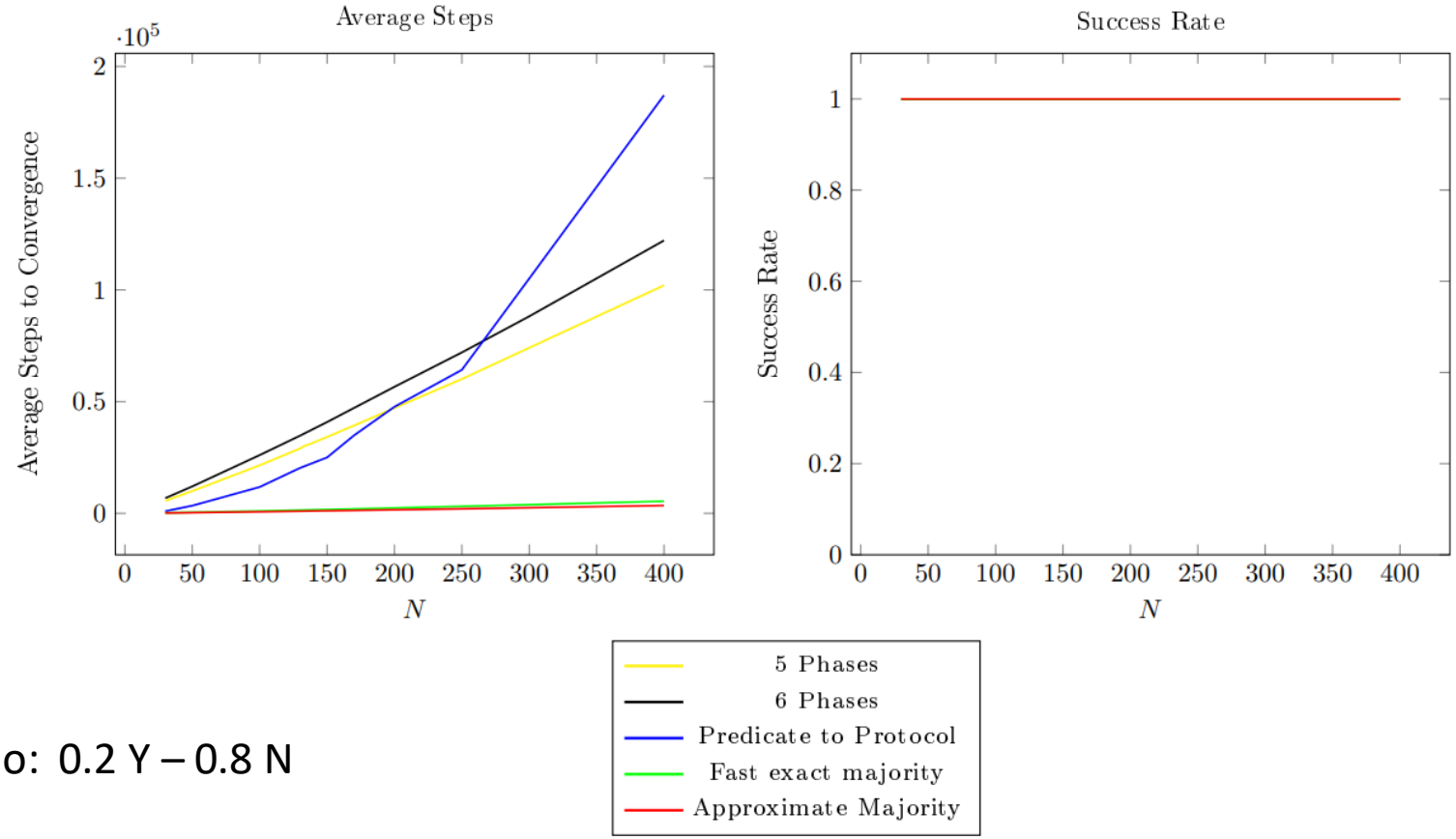


- Updates:
  - Adjust number at leaf node
  - Propagate change up to root node
  - $O(\log|S|)$ steps

Identity Map

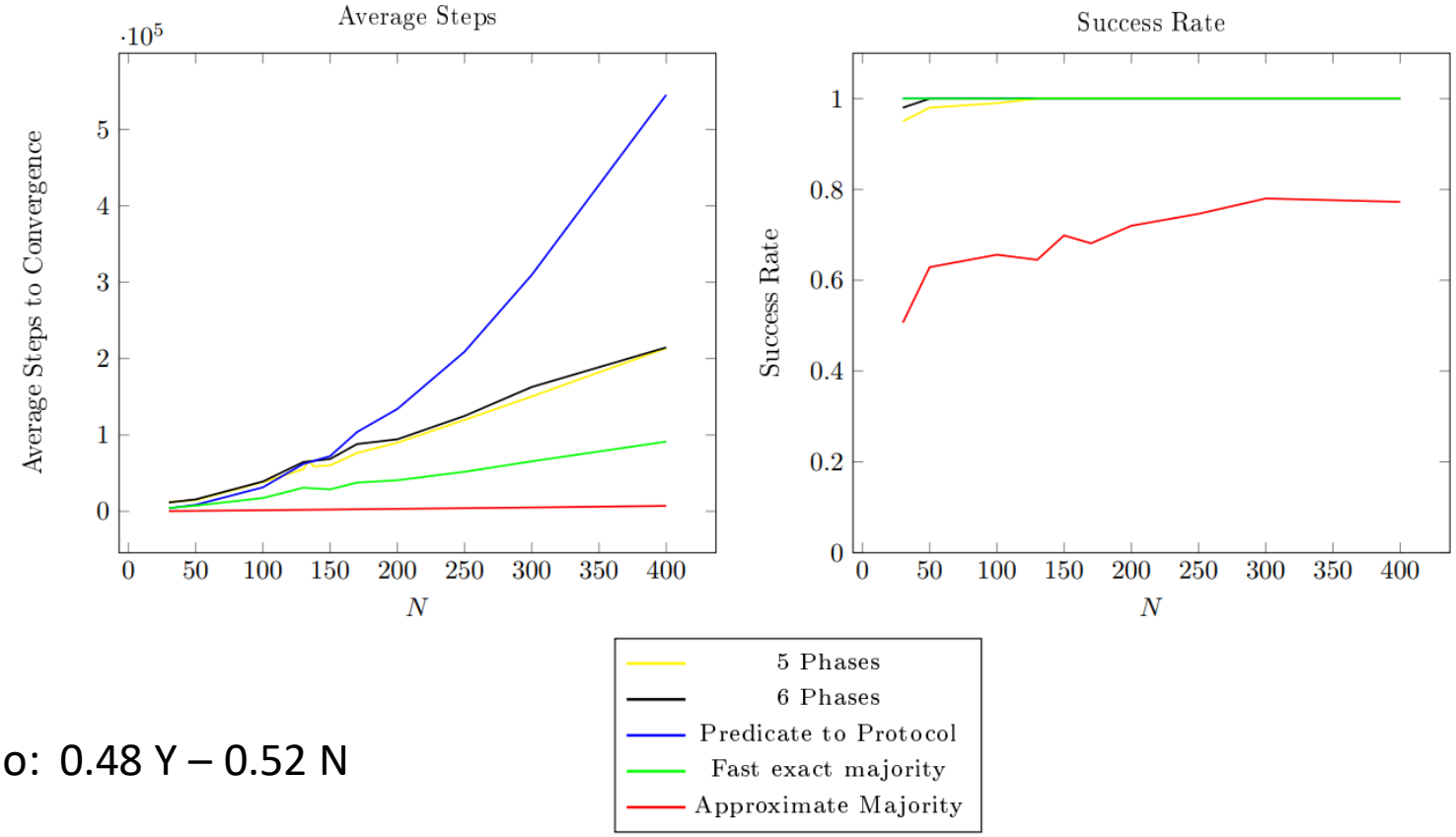
- Counting Vectors and Segment Tree depend on the number of states
- What if we have many states but few agents?
- Identity Map: Store the current state of each agent
- Random Choice:
  - Choose a number between 0 and the number of agents and get the state for that agent
  - $O(1)$ steps
- Update:
  - Replace an entry in the list
  - $O(1)$ steps
- No need to store all states
  - Need to dynamically generate transitions in order to take advantage of this

Comparing Protocols for Majority



Ratio: 0.2 Y – 0.8 N

Comparing Protocols for Majority



Ratio: 0.48 Y – 0.52 N

Conclusion

- Microcode Protcols can compute all semi-linear predicates
- Technically approximate, but very small error rate
- Can not compete with dedicated protocols for tasks