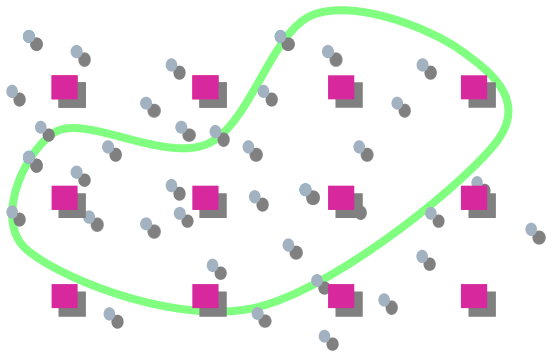
Virtual Infrastructure for Programming Mobile Robots

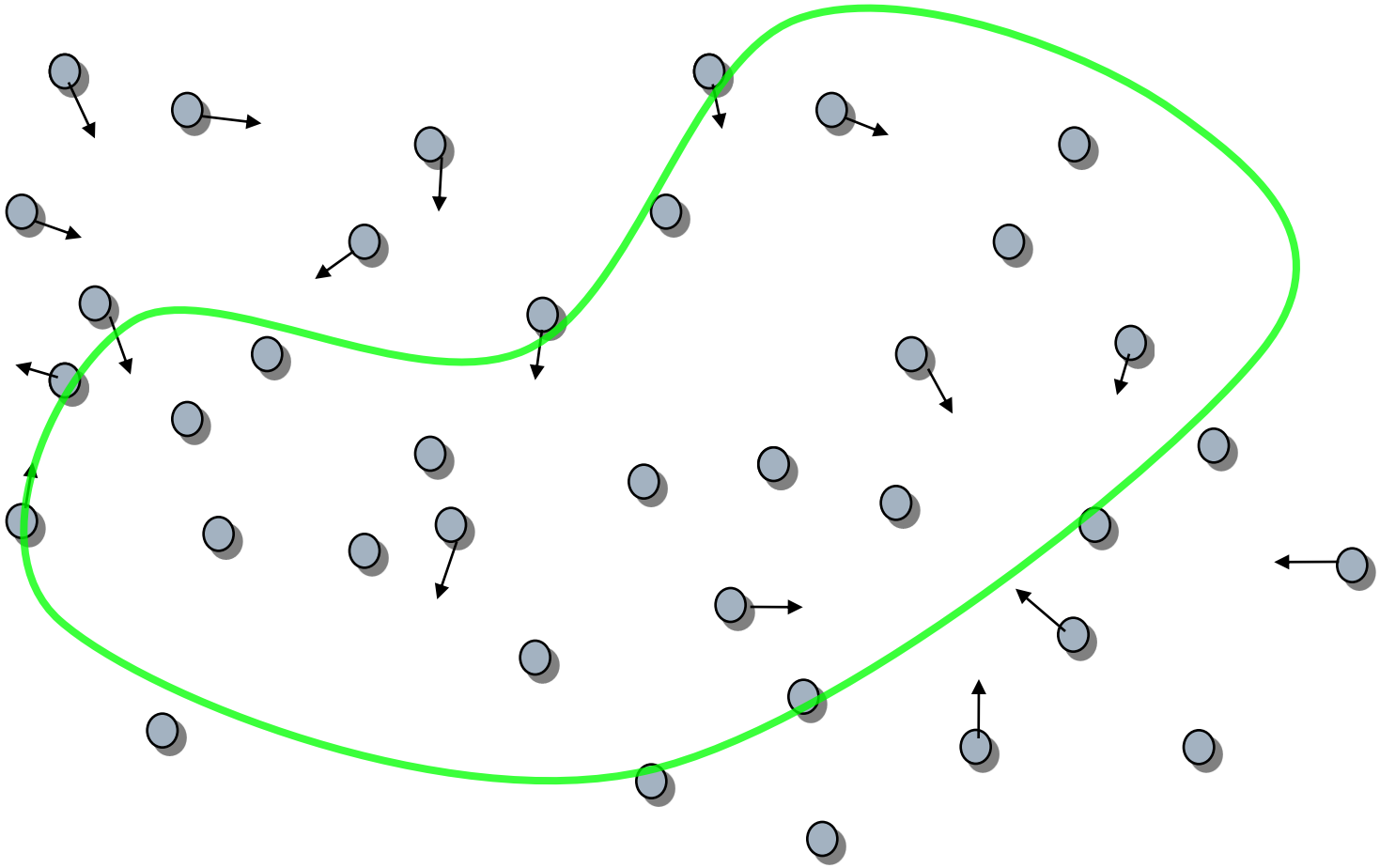
Sayan Mitra(UIUC)



collaborators

- joint work with
- Seth Gilbert (EPFL)
- Nancy Lynch (MIT)
- Tina Nolte (MIT)

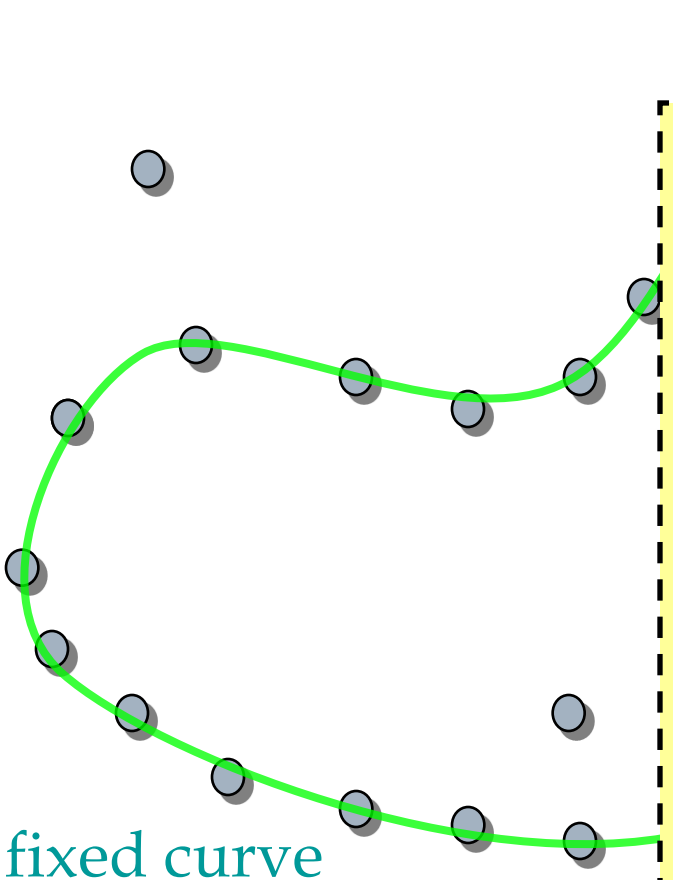
robust pattern formation



computing nodes with controlled mobility

failures, joins, unreliable communication

robust pattern formation



given fixed curve

goal: self-stabilizing formatio
unreliable communication

Oblivious, anonymous robots

Suzuki, Yamashita`99

Defago, Konagaya`02

Synchronous communication

Blondel, Tsitsiklis

Murray, Olfati-Saber, Fax `95

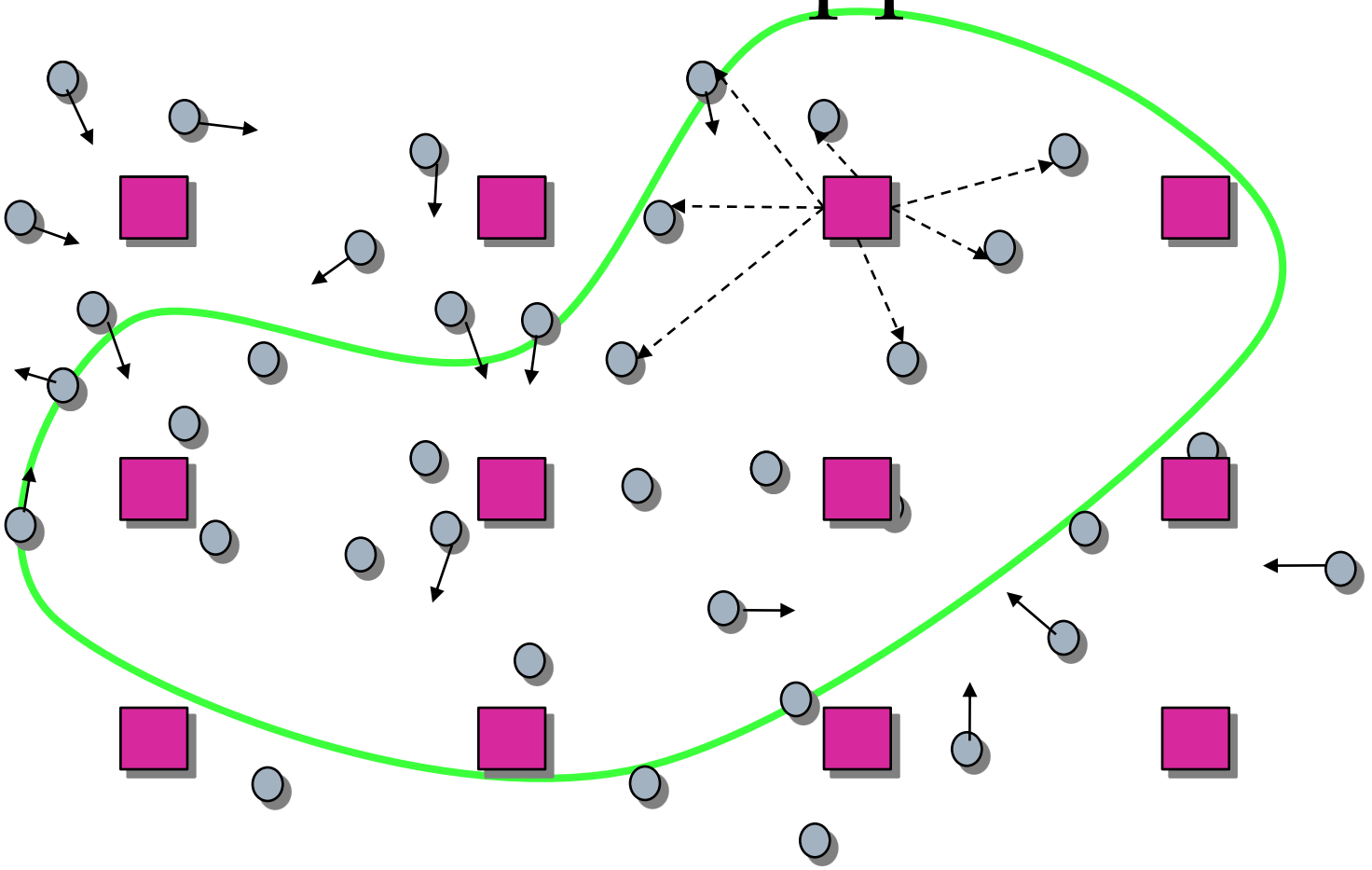
Cortez, Martinez, Bullo `04

Partially synchronous

Chandy, Mitra, Pilotto `08

No failures!

virtual node approach



Mobile robots emulate virtual nodes (VNE)

Virtual nodes execute local motion coordination (MC) algorithm

MC self-stabilizing + VNE self-stabilizing = robust pattern formation

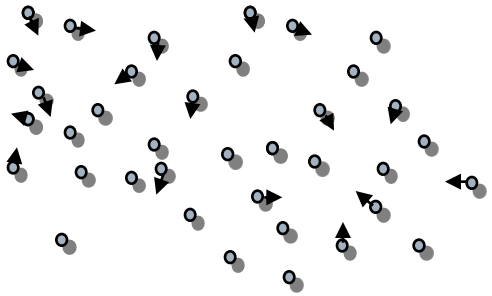
Its the programming abstraction stupid!

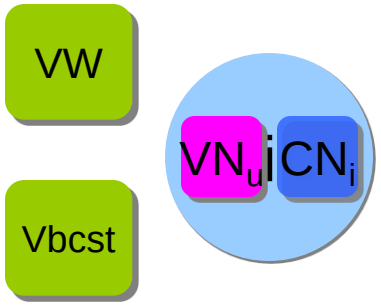
- autonomous robots in the real-world
  - exploration, rescue, cooperative control
  - curve may be changing, locally defined
- application development is hard
  - no infrastructure,
  - unpredictable reliability, communication, motion

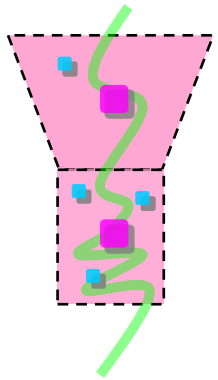
  how to simplify application development while guaranteeing correctness ?
- VN approach
  - Prove properties of VNE once and for all
  - write applications for VNs --- stationary, relatively persistent
  - analyze stability of VN algorithm --- relatively routine

outline

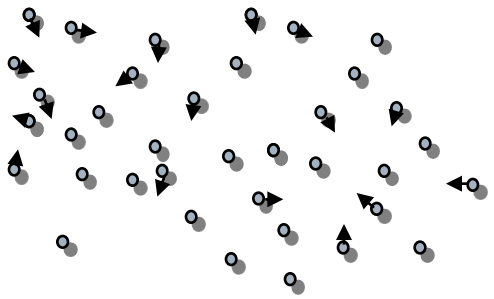
- physical model & VN layer
- VNE is self-stabilizing
- MC is self-stabilizing
- conclusion



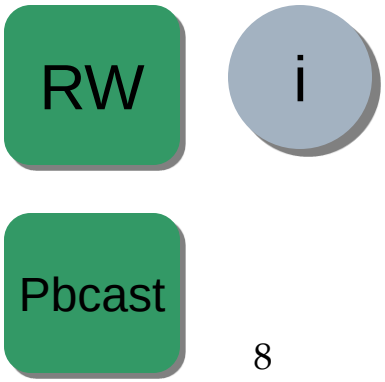




physical layer assumptions



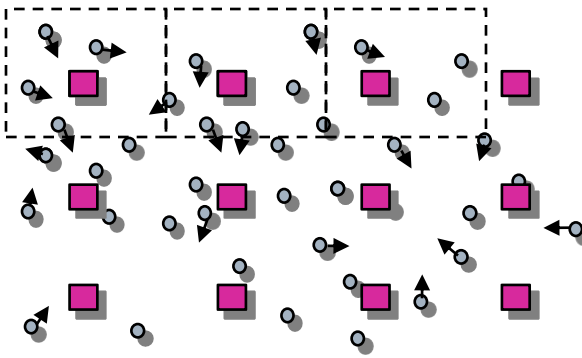
- Physical nodes may fail and restart
  - P: set of unique ids for nodes
  - v_{max} : max speed
  - local clocks progress at rate of real-time
- Nodes receive from *RW (GPS)*
  - location/ region information, real-time clock
  - refreshes each node at least every ε_{sample} time.
- Area tiled into regions with ids in U
  - regions are neighbors if they share points
  - r: max dist between 2 points in neighboring regions
- Pbcast local broadcast: bcast, brcv
  - non-duplicative delivery
  - bounded-time delivery: d_{phys}
  - reliable delivery within distance: $r + \varepsilon_{sample}\, v_{max}$

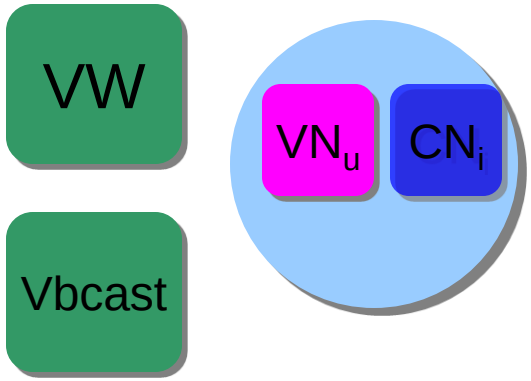


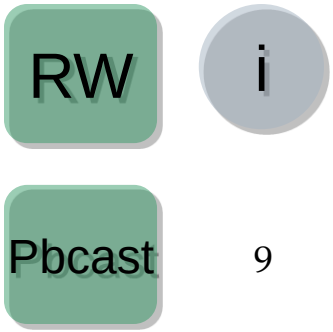
VSA Layer

Implemented by physical layer

- fixed tiling of deployment space
- VN algorithm
  - predetermined locations and programs
  - automaton with real-time clocks: Timed IOA
  - with external interface: bcast, receive, fail, restart
- Vbcast local broadcast:vcast, vrcv
  - integrity, non-duplicative delivery
  - bounded-time delivery $\leq d$ (larger than d_{phys})
  - Reliable delivery to all VNs & CNs in same or in neighboring region
- VW
  - provides correct real-time to VNs
  - at least every ε_{sample} time
  - can crash, restart







VN Emulation algorithm

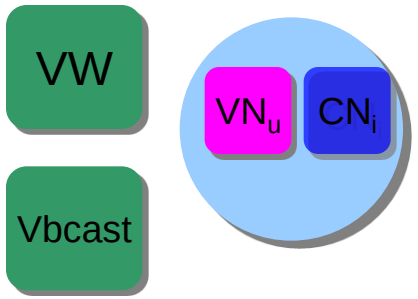
- Implements of TO broadcast
- Implements of VN algorithm
  - Round-based
    - Emulates deterministic timed state machine
    - Messages sent using totally ordered broadcast
    - Each mobile node maintains state and processes messages as if it was the VN
  - Leader-based
    - Only leader broadcasts on behalf of the VN
    - Leader broadcasts state after performing VN broadcasts to maintain consistency and help joiners

Implementation VN

- TIOA *A* implements TIOA *B* *(written $A \leq B$)* if *A*'s externally visible behaviors are externally visible behaviors of *B*.

- *amap* maps virtual layer algorithm to a physical layer algorithm

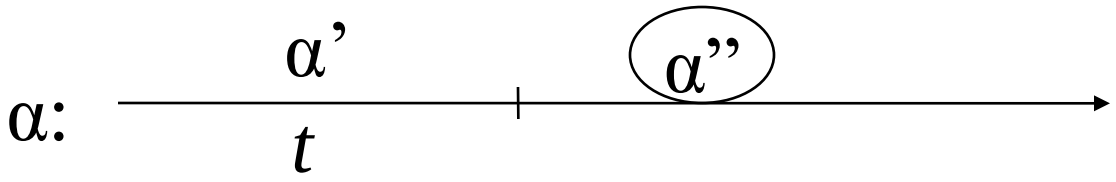
  PLNodes[amap[MC]] || RW || Pbcast
  implements
  VLNodes[MC] || VW || Vbcast



- For any *virtual layer algorithm A* and any execution *E* of the emulation of *A*, there exists a corresponding execution *E'* of *VLayer[alg]*

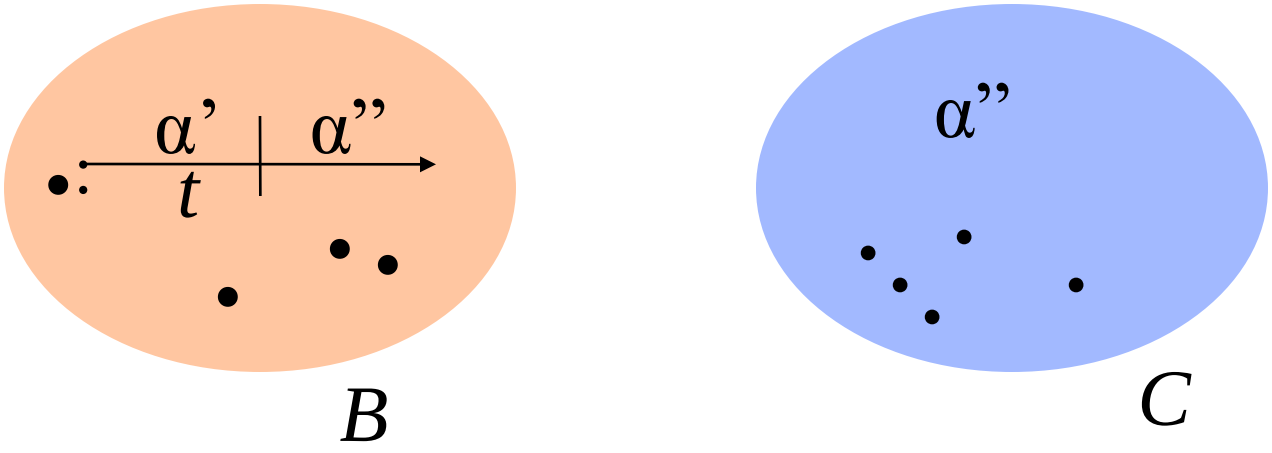
Stabilization preliminaries

- Stability: starting from arbitrary state s system reaches some state in a desirable set G
- Environment includes RW, Pbcast, VW, Vbcast
- New stability: any behavior (execution fragment) from an arbitrary set B, has a suffix in a desirable set of behaviors C
- α'' is a *state-matched t-suffix* of execution fragment α



Stabilization

- Let *B*, *C* be a sets of *execution fragments*, *t* be a non-negative real.
- *B stabilizes* in time *t* to *C* if each state-matched *t*-suffix of each sequence in *B* is a sequence in *C*.



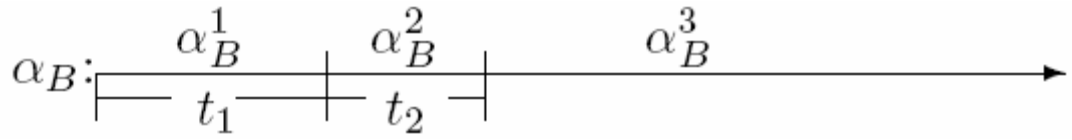
Stabilization results

- Lemma (Restriction). Let A be a set of actions, V be a set of variables, and let B stabilize to C in time t.

  Then, $\{\alpha \restriction (A,V) \mid \alpha \text{ in } B\}$ stabilizes to $\{\alpha \restriction (A,V) \mid \alpha \text{ in } C\}$ in time t.

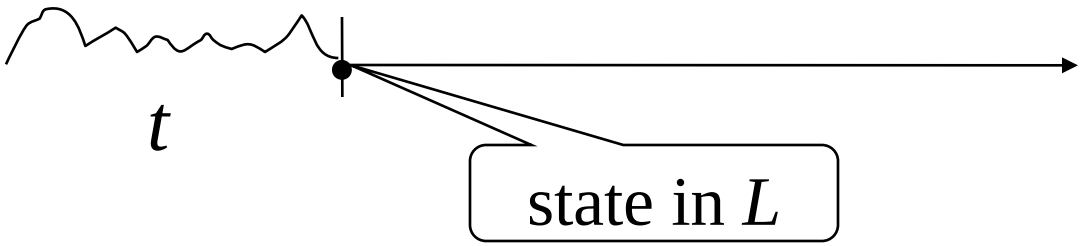
- Lemma (Transitivity). Let B stabilize to C in time t_1, and C stabilize to set D in time t_2.

  Then, B stabilizes to D in time t_1+t_2.



self-stabilization

- L is a *stable set* for $\boldsymbol{A}$ if it is closed under transitions
- *Any(**A**)*: A started in any arbitrary state
- *Reach(**A**)*: A started in any reachable state
- Let L be a legal set for $\boldsymbol{O} \,||\, \boldsymbol{A}$
- $\boldsymbol{A}$ *self-stabilizes* in time t to L relative to environment $\boldsymbol{O}$ if the set of executions of *Any(**A**)* $||\, \boldsymbol{O}$ stabilizes in time t to set of execution fragments of $\boldsymbol{A} \,||\, \boldsymbol{O}$ started in L



Stabilizing emulation

- (✌, O_A) emulation stabilizes in time t to (👌, O_B) under *emu* if:
  - For each B in 👌, the set of traces of *Any*(*emu*(B)) || *Reach*(O_B) stabilizes in time t to the set of traces of *Any*(B) || *Reach*(O_B)

Stabilizing emulation algorithm

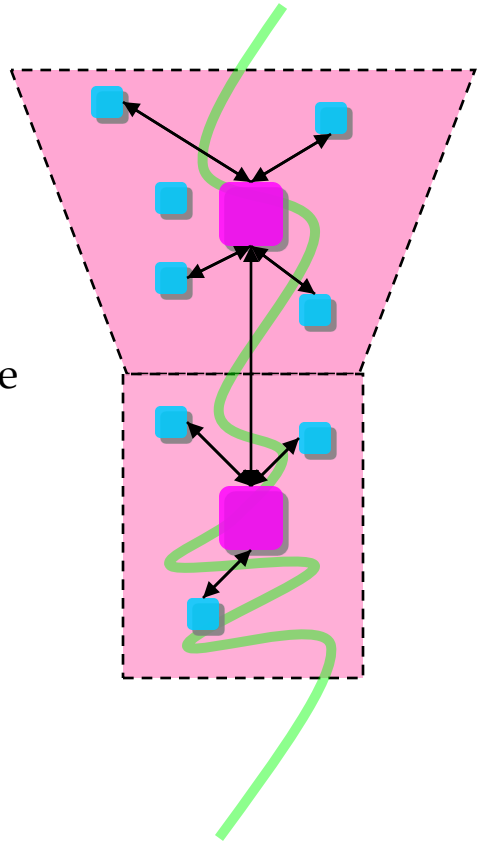
- VNE is a stabilizing emulation
  - if mobile nodes suffer corruption or the network unexpectedly changes, the implementation recovers to correctly emulate a VN from possibly arbitrary state.
    - Consistent with the abstract VN having suffered corruption
  - achieved with local checking and leader-broadcast state refresh in the main emulation algorithm
  - achieved with local checking and timestamps in implementations of the totally ordered bcast and leader election services.
    - uses RW oracle as source of consistency.

Self-stabilizing emulation proof

- Proof that algorithm is stabilizing emulation:
  1. Show totally ordered bcast implementation stabilizes in time t_1 to look like totally ordered bcast service in a legal state.
  2. Show leader election implementation stabilizes in time t_2 to look like leader election service in a legal state.
  3. Show that the main emulation algorithm, using totally ordered bcast and leader election services in legal states, stabilizes in time t_3 to look like the VSA layer.
  4. Conclude main emulation algorithm running with implementations of TObcast, leader election stabilizes in time $\max(t_1, t_2)+ t_3$ to look like the VSA layer.

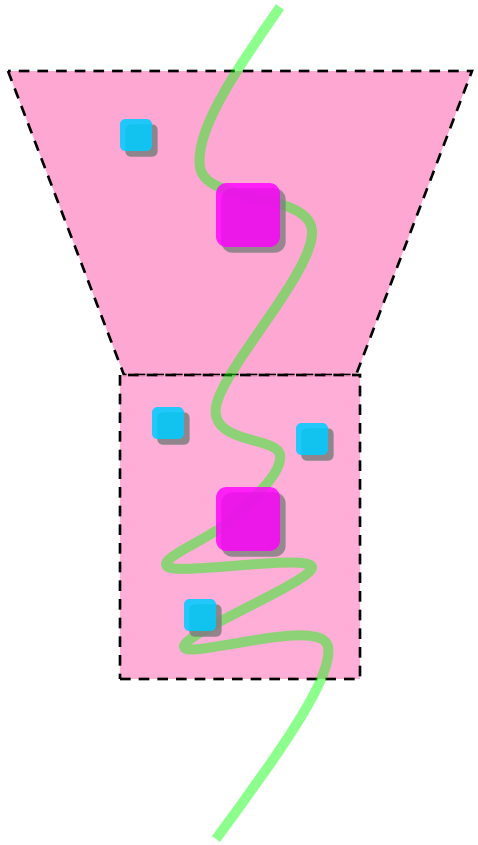
Motion Coordination Algorithm MC

- Algorithm for VN & algorithm for client nodes
- Each round:
  1. Each CN sends a message to its local VN, letting it know it is in the VN's zone
  2. Each VN exchanges messages with neighboring zone VNs, letting them know how many CNs it has
  3. Each VN calculates
     a) which of its local CNs should be assigned to other zones
     b) what its local CNs' new target points should be
  4. Each VN broadcasts the new target points
  5. Each CN reads VN target point and moves to it.



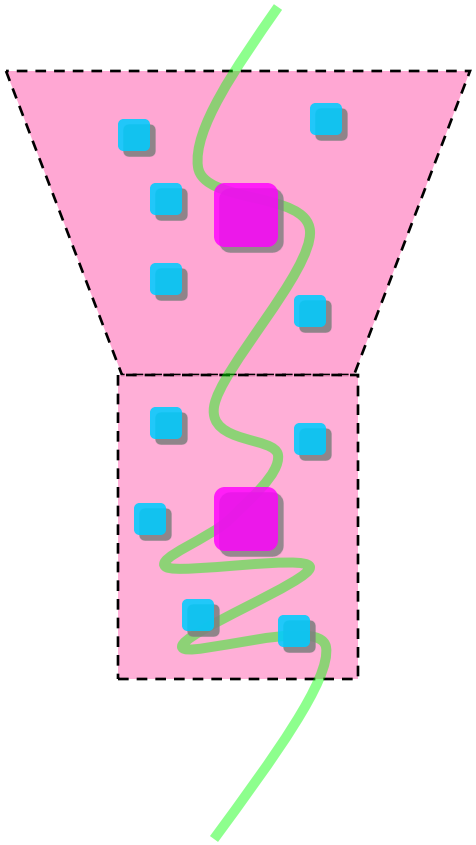
Stabilization of MC

- Uses notion of reset state:
  - MC starts fresh in each round
  - Stabilization proof: eventually correct local information about region and time, and then reach a fresh (reset) state, from which MC behaves normally
- If a client arrives in a region with a failed VN, then the VSA in the region is guaranteed to be alive by the next round



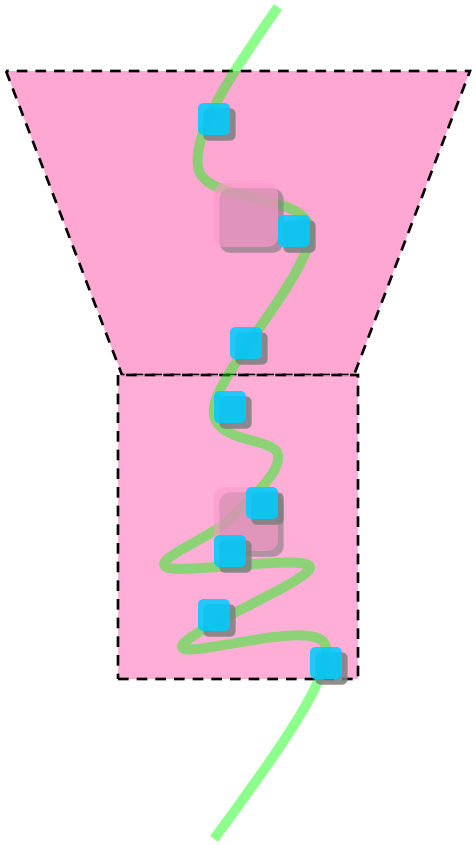
Stabilization of MC

- **Theorem:** If there are no failures or recoveries of client nodes at or after some round t_0, then:
  - within a finite number of rounds after t_0, the set of CNs assigned to each region u becomes fixed
  - the size of the set is proportional to the percentage of the curve in the region within a small tolerance



Stabilization of MC

- **Theorem:** If there are no failures or recoveries of client nodes at or after some round t_0, then:
  - within a finite number of rounds after t_0, the set of CNs assigned to each region u becomes fixed
  - the size of the set is proportional to the percentage of the curve in the region within a small tolerance , and
  - all client nodes in a region the curve passes through are located on the curve and are evenly spaced in the limit



conclusion

- virtual layer provides a useful programming abstraction for developing distributed coordination protocols
- robust for free

  VN emulation is self stabilizing

  \+ self-stabilizing VN application (e.g. MC)

  = self-stabilizing system (pattern formation)

future direction

- implementation, experiments
- more involved tasks
  - distributed path planning
  - coordinated operations