Collaborating in Motion

Distributed and Stochastic Algorithms for Emergent Behavior in Programmable Matter

Joshua J. Daymude

Prof. Andréa W. Richa (Chair)

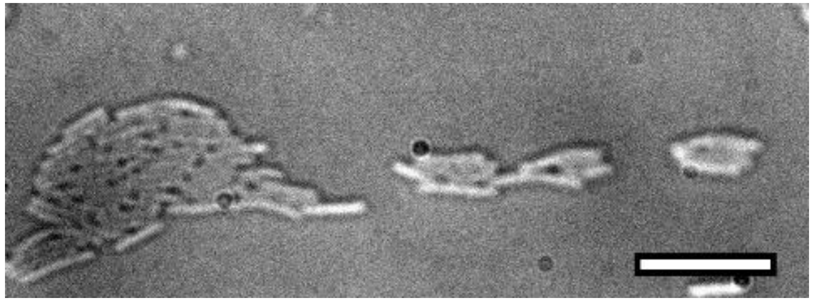
Prof. Christian Scheideler, Prof. Dana Randall, Prof. Theodore Pavlic, and Prof. Stephanie Gil

PhD Dissertation Defense — March 15, 2021

CIDSE, Arizona State University

Self-Organizing Systems

Cooperative decentralized systems are capable of surprising emergent behavior arising from relatively simple interactions of their members.



HMSKCLCA 2011

Microsoft Research 2016

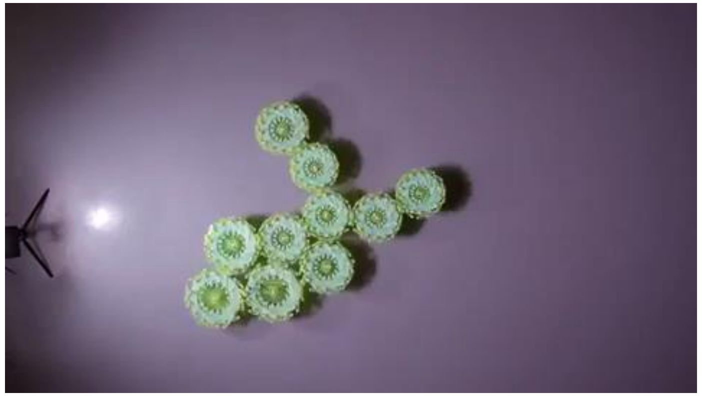
Programmable Matter

Programmable matter is a substance that can change its physical properties autonomously based on user input or environmental stimuli.

"Catoms"
PB 2018

"Kilobots"
RCN 2014

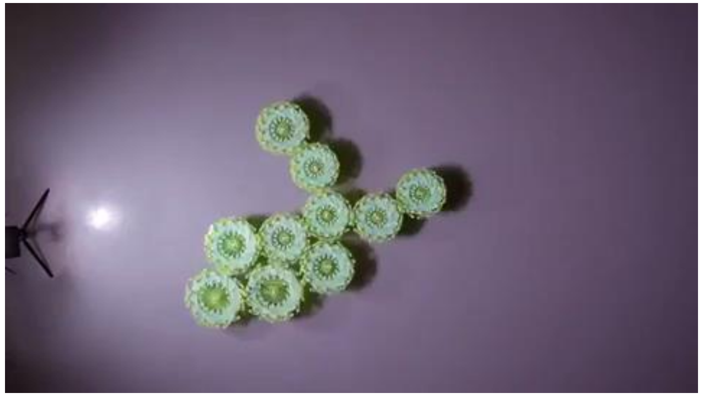
"M-Blocks"
RGR 2013

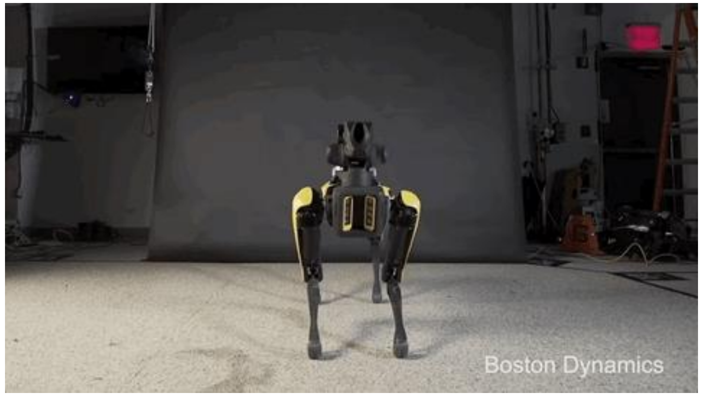


"Particle Robots"
LBBCRHRL 2019

Programmable Matter

Centimeter/millimeter-scale robots are more limited than, say, Spot from Boston Dynamics.



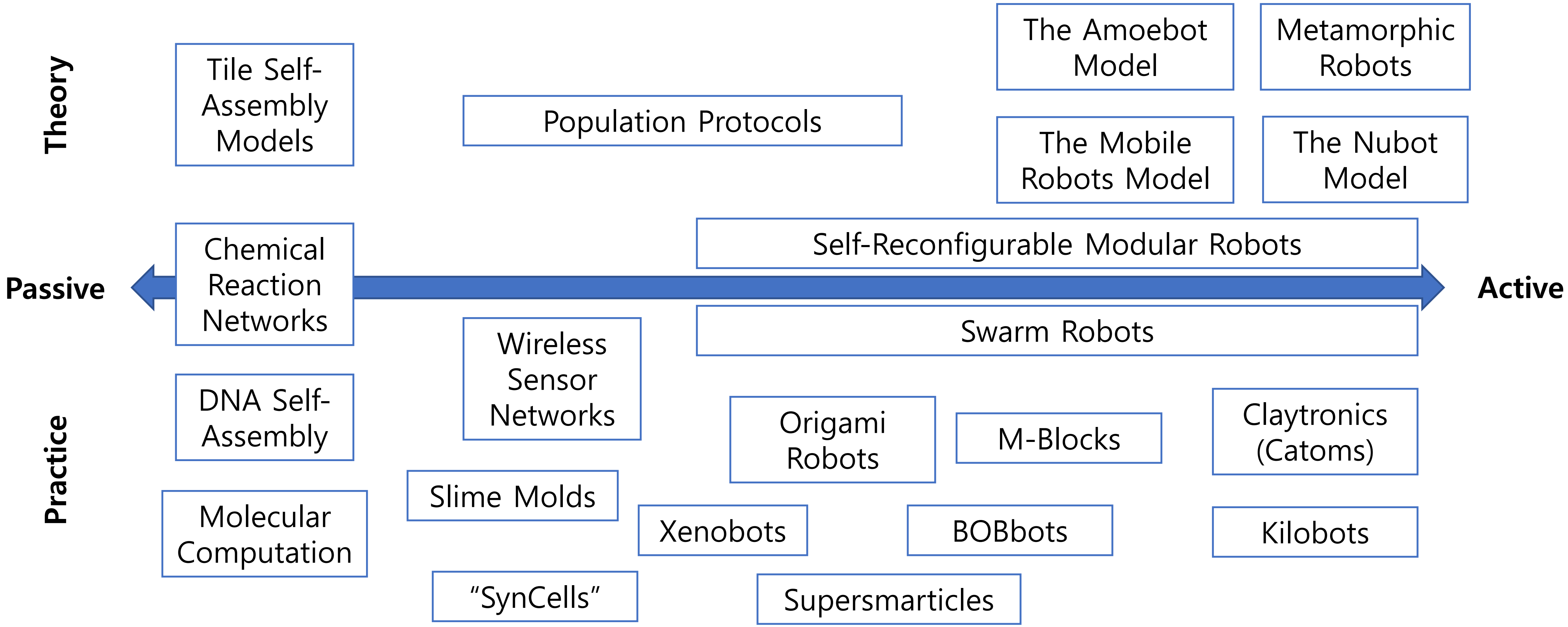


Most programmable matter and modular robotic systems assume:

- Modest compute resources.
- Strictly local sensing and communication (e.g., 1-neighborhood).
- Limited (e.g., constant-size) or no persistent memory.
- Local, rudimentary movement.

Existing Research on Programmable Matter

Programmable matter systems can be organized by their degree of self-determination in deciding and enacting local behaviors.



Dissertation: Outline

This dissertation focuses on the algorithmic foundations of active programmable matter.

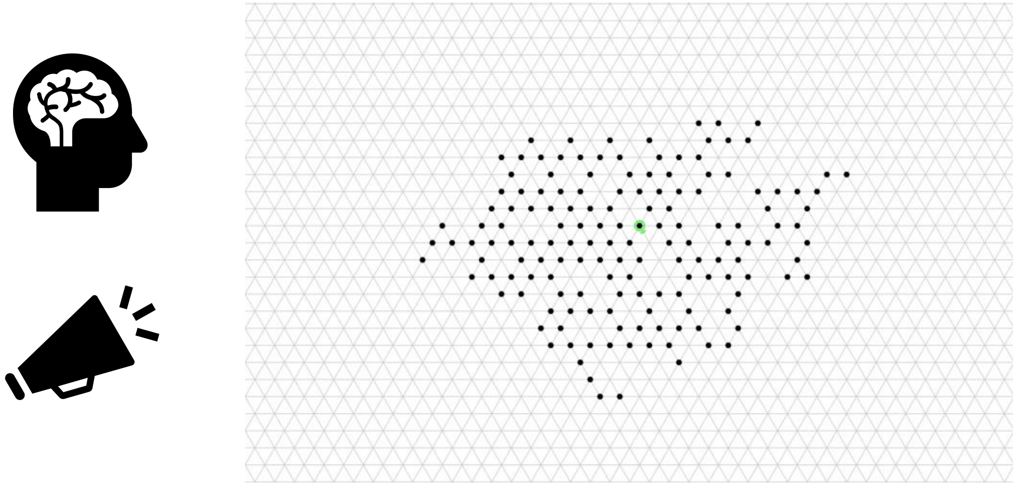
Three main research questions:

1. What are the minimum individual capabilities necessary to achieve system behavior X?
2. How can existing algorithms be enhanced to capture more realistic assumptions?
3. How can digital algorithms be translated for simple, analog (passive) systems?

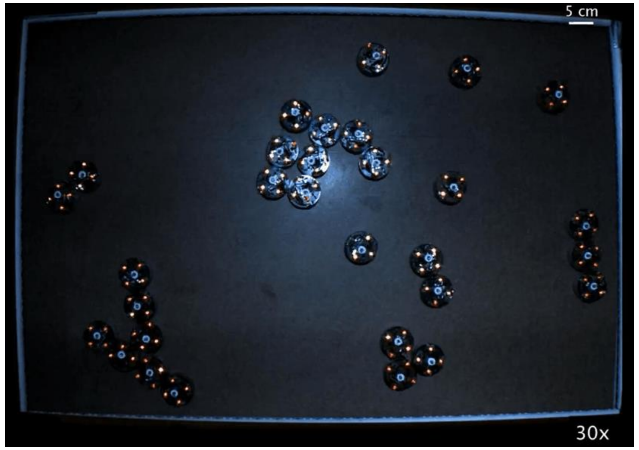
Dissertation: The Big Picture

1. The Amoebot Model and its Enhancements

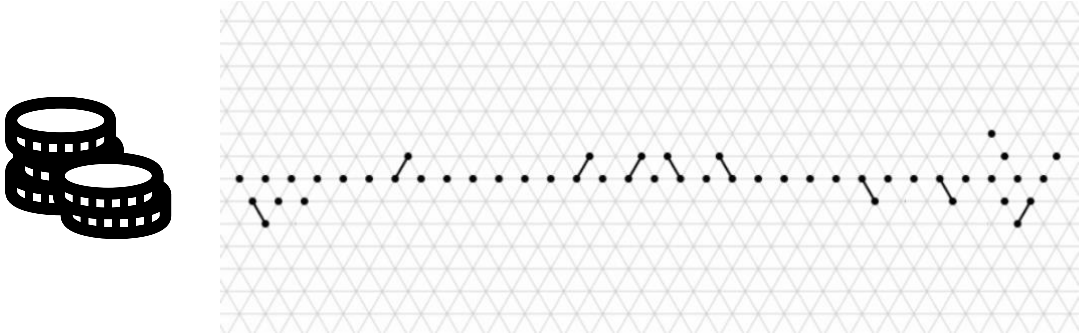
2. Stateful Distributed Algorithms



4. Swarm Robotics and Granular Active Matter



3. Stochastic Distributed Algorithms



Part I

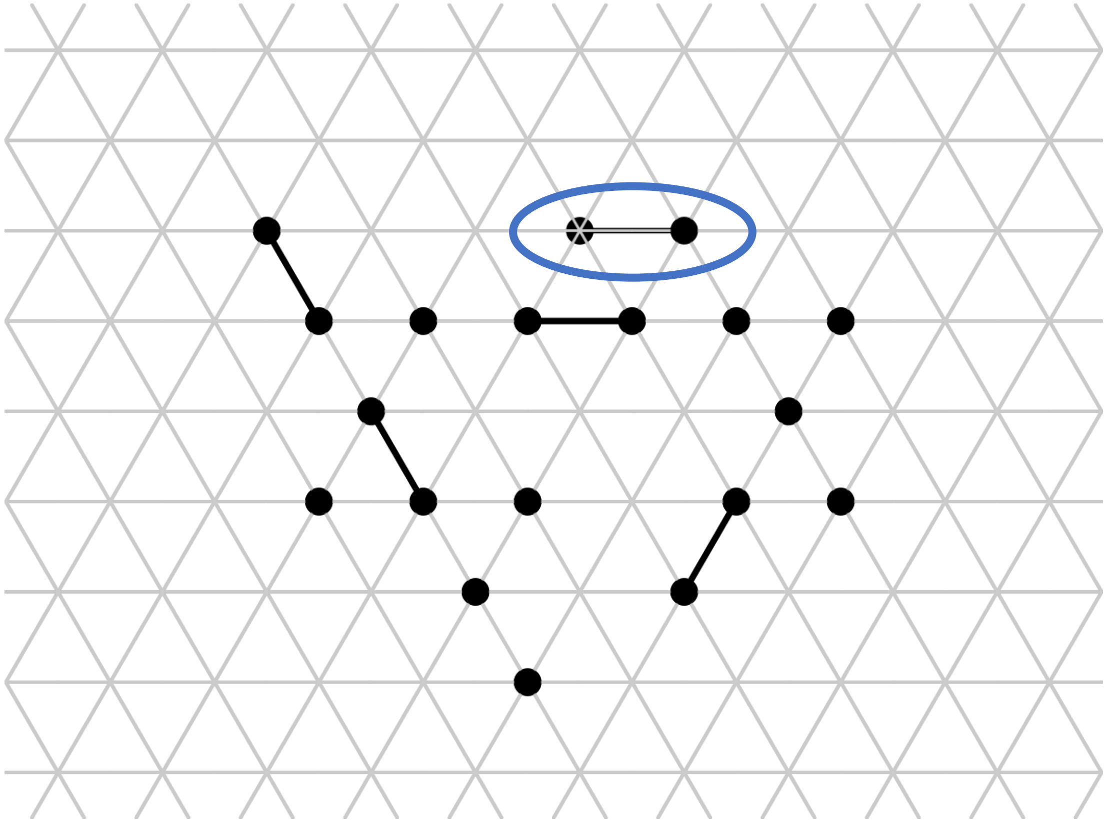
Stateful Distributed Algorithms for Programmable Matter

What are the minimum individual capabilities necessary to achieve system behavior X?

The Amoebot Model (2014–2021)

The **amoebot model** is an abstraction of programmable matter.

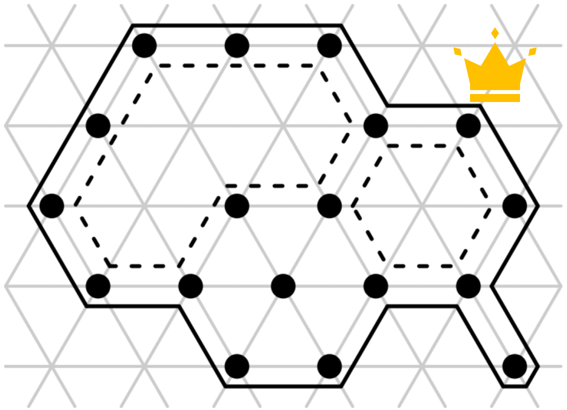
- Space: triangular lattice G_Δ.
- Amoebots can be contracted (one node) or expanded (two adjacent nodes).
- Amoebots are anonymous, have only constant-size memories, communicate with immediate neighbors, and have no global compass.
- Self-actuated movements via expansions, contractions, and handovers.
- Sequential, weakly fair adversary: one amoebot acts per time, every amoebot acts infinitely often.

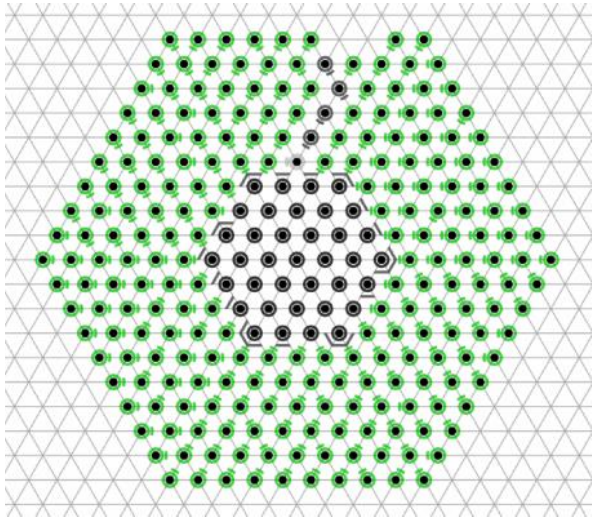


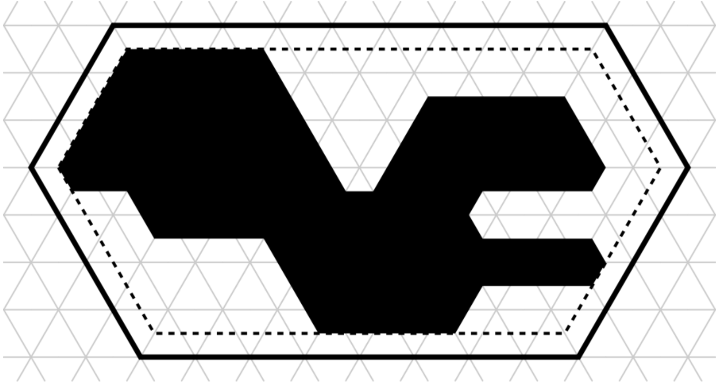
Stateful Distributed Algorithms

With constant-size memory, communication between neighbors, and local movements, amoebot systems can solve:

1. **Leader Election**. A unique amoebot must irreversibly declare itself the system's leader.
2. **Object Coating**. The system must reconfigure into even layers coating a given object.
3. **Convex Hull Formation**. The system must reconfigure as the convex hull of a given object, enclosing it with the minimum number of amoebots.





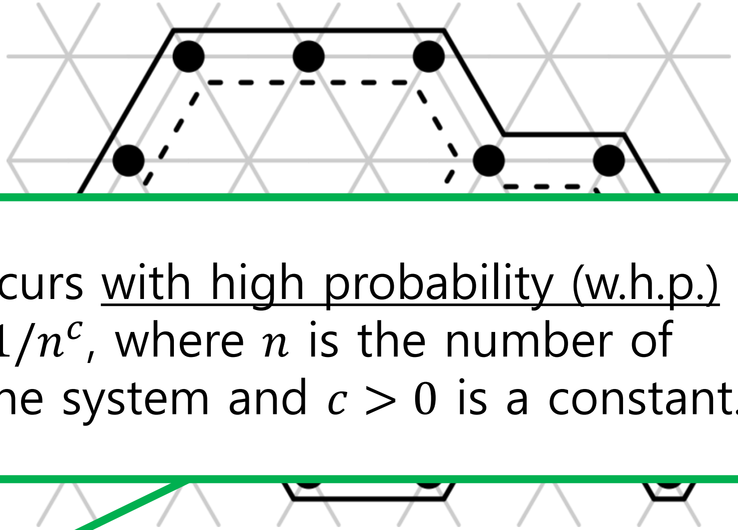


Leader Election

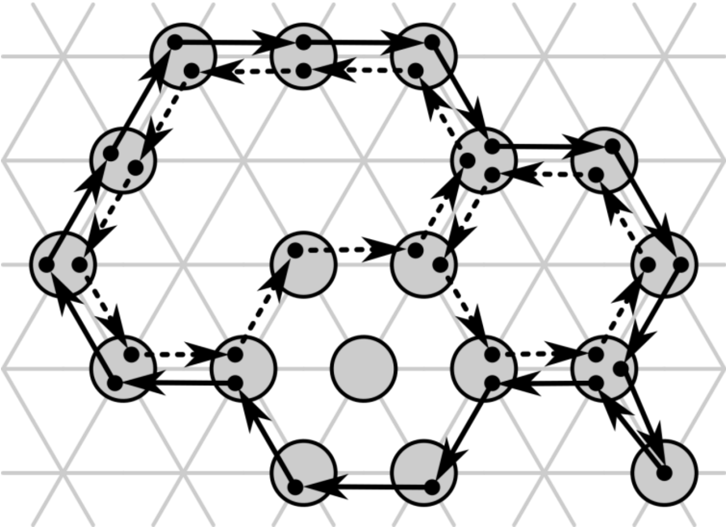
Problem: A unique amoebot must irreversibly declare itself the system's leader.

Motivation: Leader election is well-studied in distributed computing. Can help coordinate the system for more complex behaviors (e.g., shape formation, object coating, etc.).

Key Idea: Each amoebot gets a random value. The largest value on the outer boundary wins.

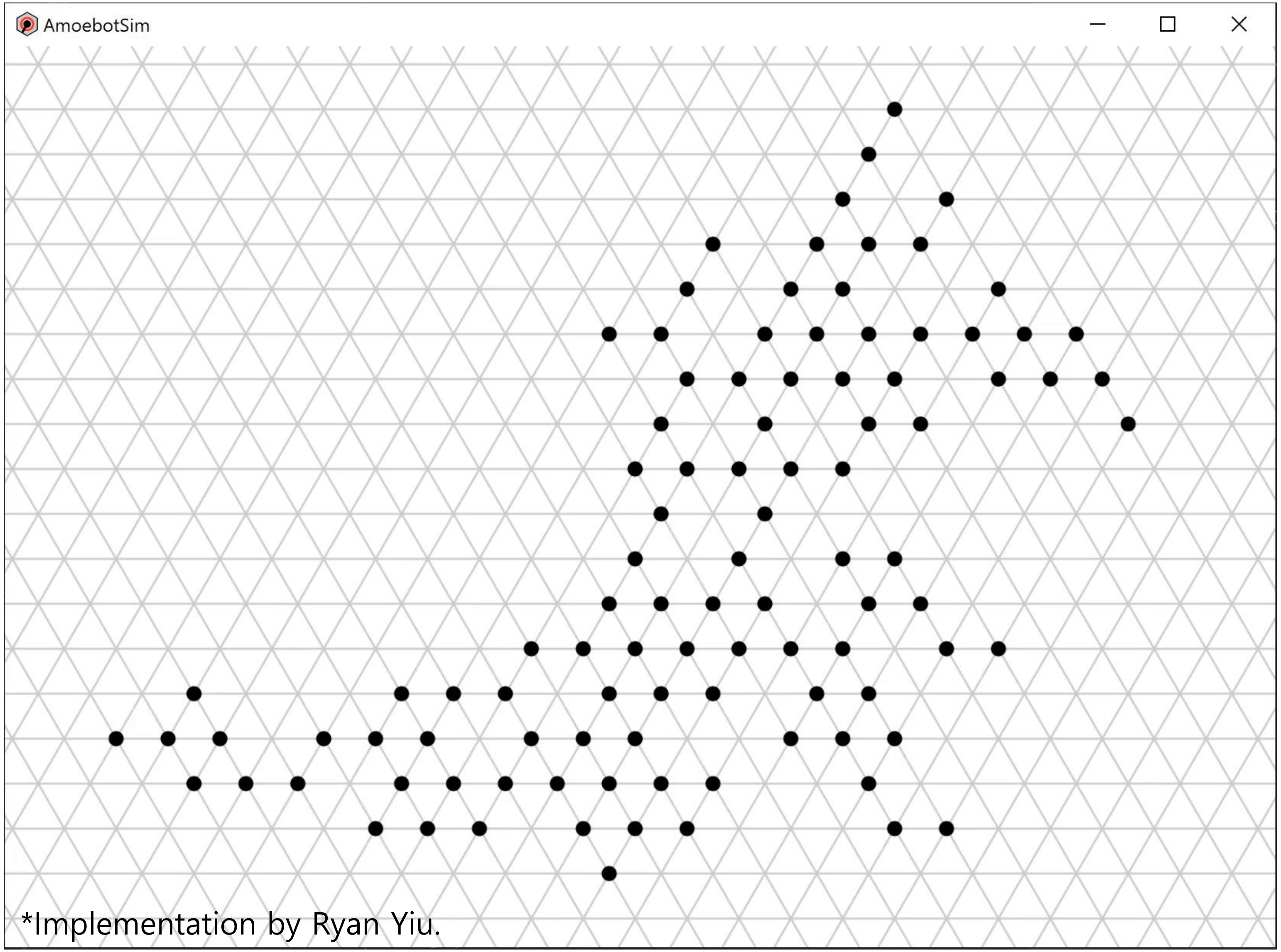


An event A occurs with high probability (w.h.p.) if $\Pr[A] \geq 1 - 1/n^c$, where n is the number of amoebots in the system and $c > 0$ is a constant.



Theorem. The Improved-Leader-Election algorithm solves the leader election problem in $\mathcal{O}(L)$ rounds w.h.p., where L is the length of the outer boundary.

Leader Election



*Implementation by Ryan Yiu.

Leader Election

Problem: A unique amoebot must irreversibly declare itself the system's leader.

Theorem. The Improved-Leader-Election algorithm solves the leader election problem in $\mathcal{O}(L)$ rounds w.h.p., where L is the length of the outer boundary.

Our algorithm inspired significant follow-up work:

Algorithm	**Det.**	**Weak Sched.**	**Allows Holes**	**Removes Chirality**	**Static**	**Leaders Elected**	**Runtime**
Leader-Election [59]	No	No	Yes	No	Yes	1	$\mathcal{O}(L^*)$ exp.
Improved-Leader-Election	No	No	Yes	No	Yes	1, whp.	$\mathcal{O}(L)$ whp.
Di Luna et al. [64, 65]	Yes	Yes	No	Yes	Yes	$k \leq 3$	$\mathcal{O}(n^2)$
Gastineau et al. [91]	Yes	No	No	No	Yes	1	$\mathcal{O}(n)$
Bazzi and Briones [21]	Yes	Yes	Yes	No	Yes	$k \leq 6$	$\mathcal{O}(n^2)$
Emek et al. [77]	Yes	No	Yes	Yes	No	1	$\mathcal{O}(Ln^2)$

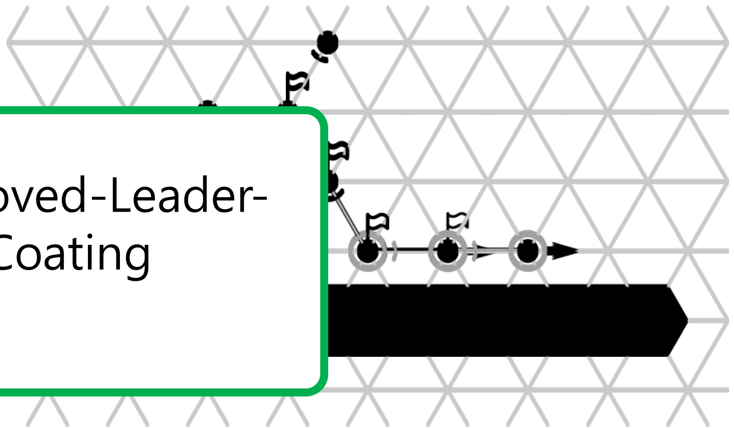
Interestingly, our Improved-Leader-Election algorithm remains state-of-the-art for settings with holes where amoebots cannot move and exactly one leader should be elected.

Object Coating

Problem: The system must reconfigure into even layers coating a given object.

Motivation: Smart paint, distributed sensor networks, and shape formation via reverse-molds.

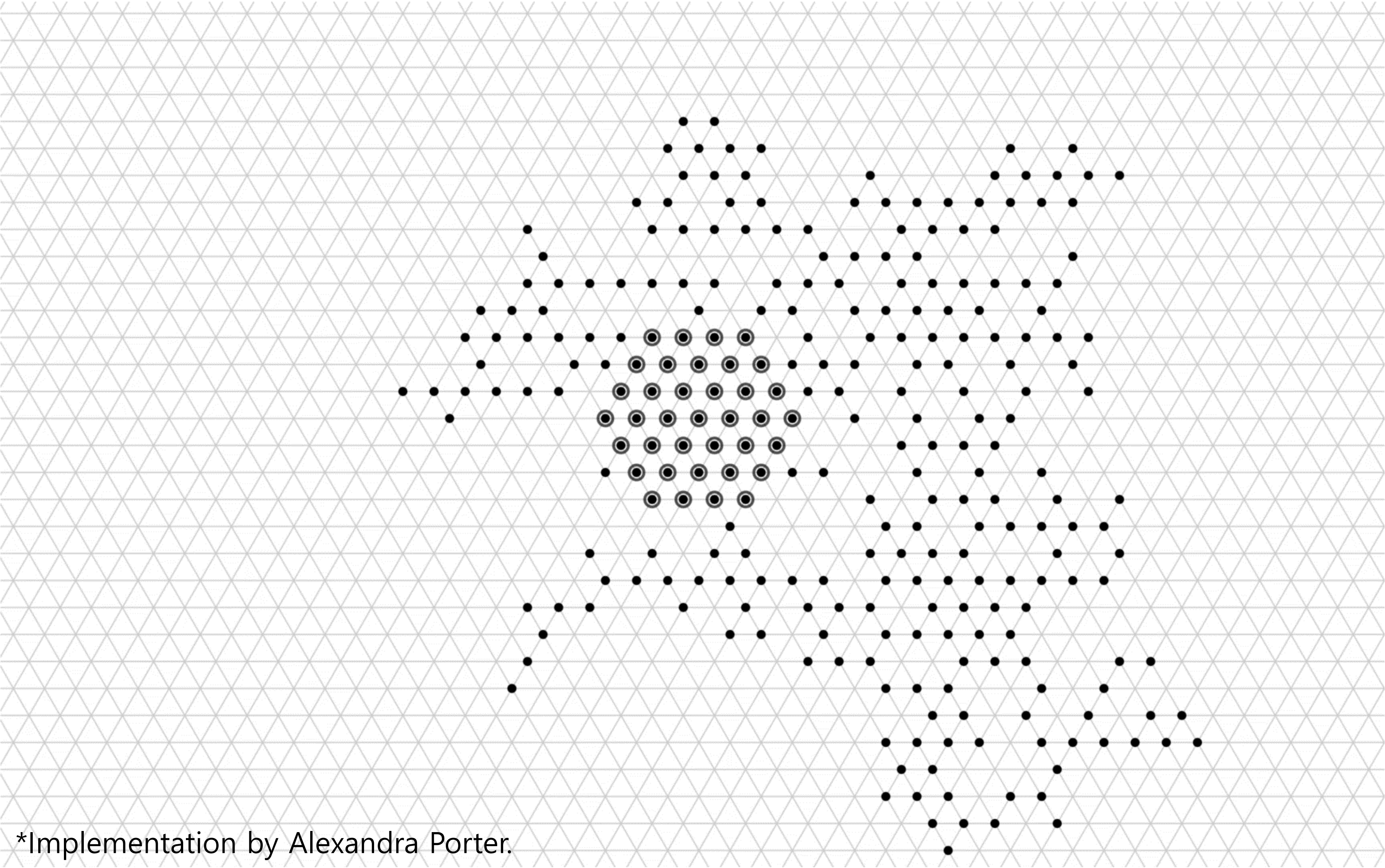
Key Idea: Coat the first layer by following the object's surface. Elect a leader to mark the start and end of higher layers. Then form higher layers. [DGRSS 2017].



This "w.h.p." is inherited from Improved-Leader-Election. The rest of the Universal-Coating algorithm is deterministic.

Theorem. The Universal-Coating algorithm solves the object coating problem in $\mathcal{O}(n)$ rounds w.h.p., where n is the number of amoebots in the system. This runtime is worst-case asymptotically optimal — no local-control algorithm can do any better, in the worst case.

Object Coating



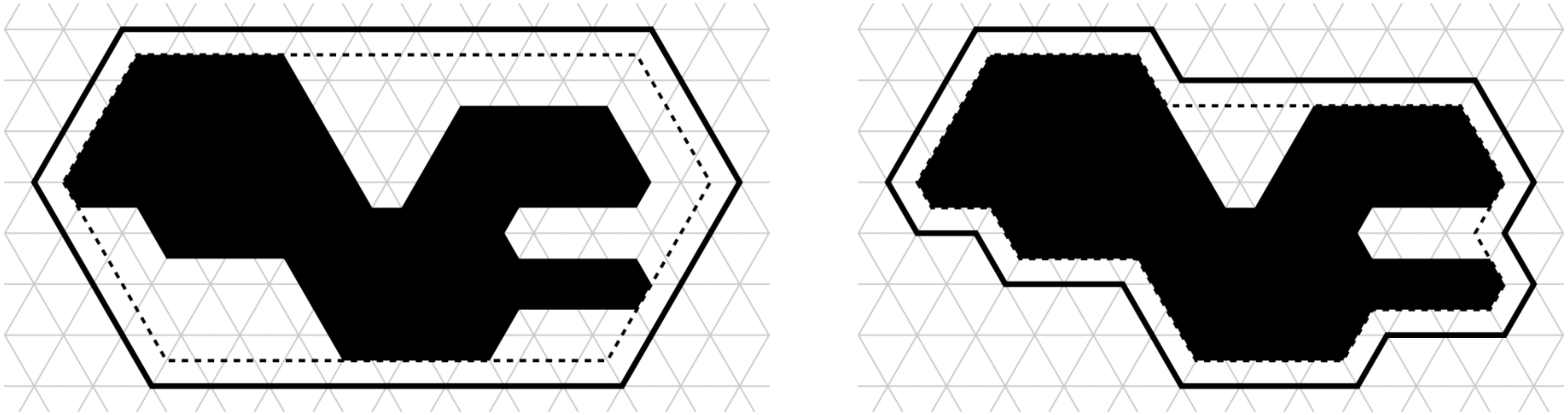
*Implementation by Alexandra Porter.

Convex Hull Formation

Problem: The system must reconfigure as the convex hull of a given object, enclosing it with the minimum number of amoebots.

Motivation: Collective transport, isolating hazardous materials, macrophage-like engulfing.

In our discrete setting of the triangular lattice, we consider restricted-orientation convex hulls.

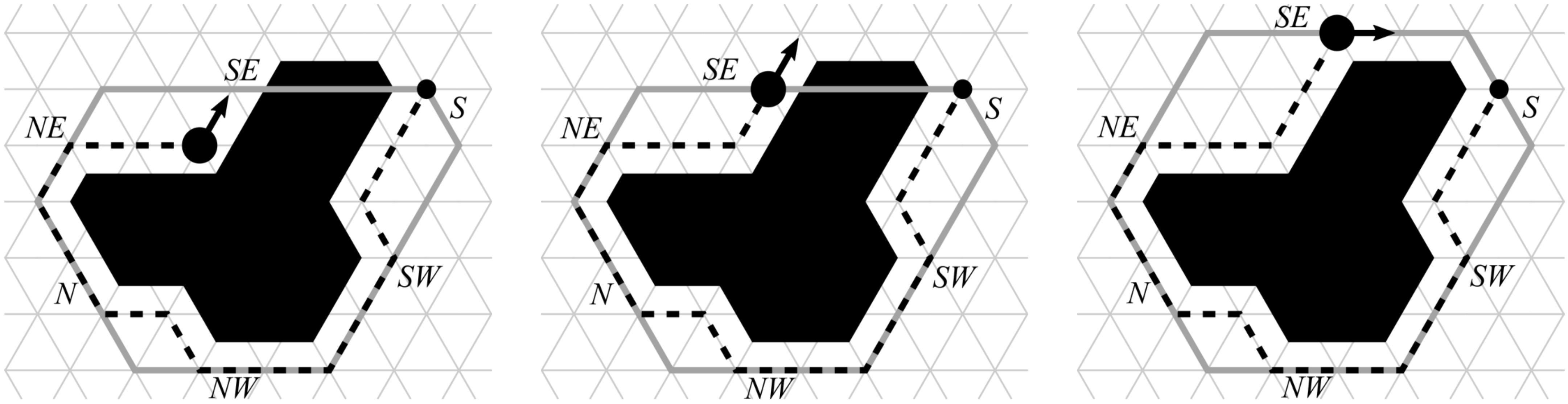


Ours is the first distributed algorithm to compute restricted-orientation convex hulls without global orientation or coordinates and when limited to constant-size memory.

Convex Hull Formation

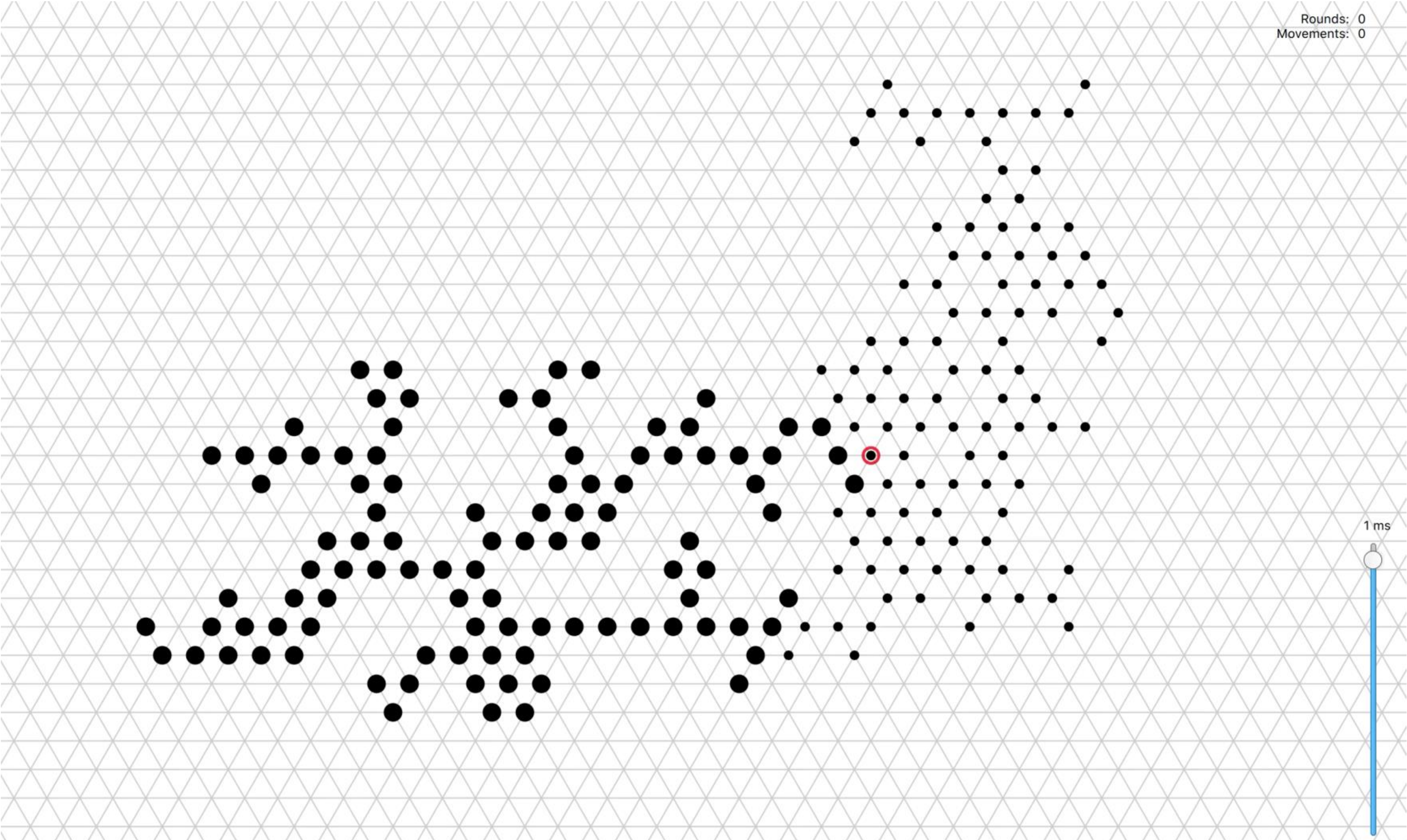
Problem: The system must reconfigure as the convex hull of a given object, enclosing it with the minimum number of amoebots.

Key Idea: Use a leader to explore the object, keeping track of its distances to each of the six half-planes forming the convex hull. Once determined, simply follow the convex hull.



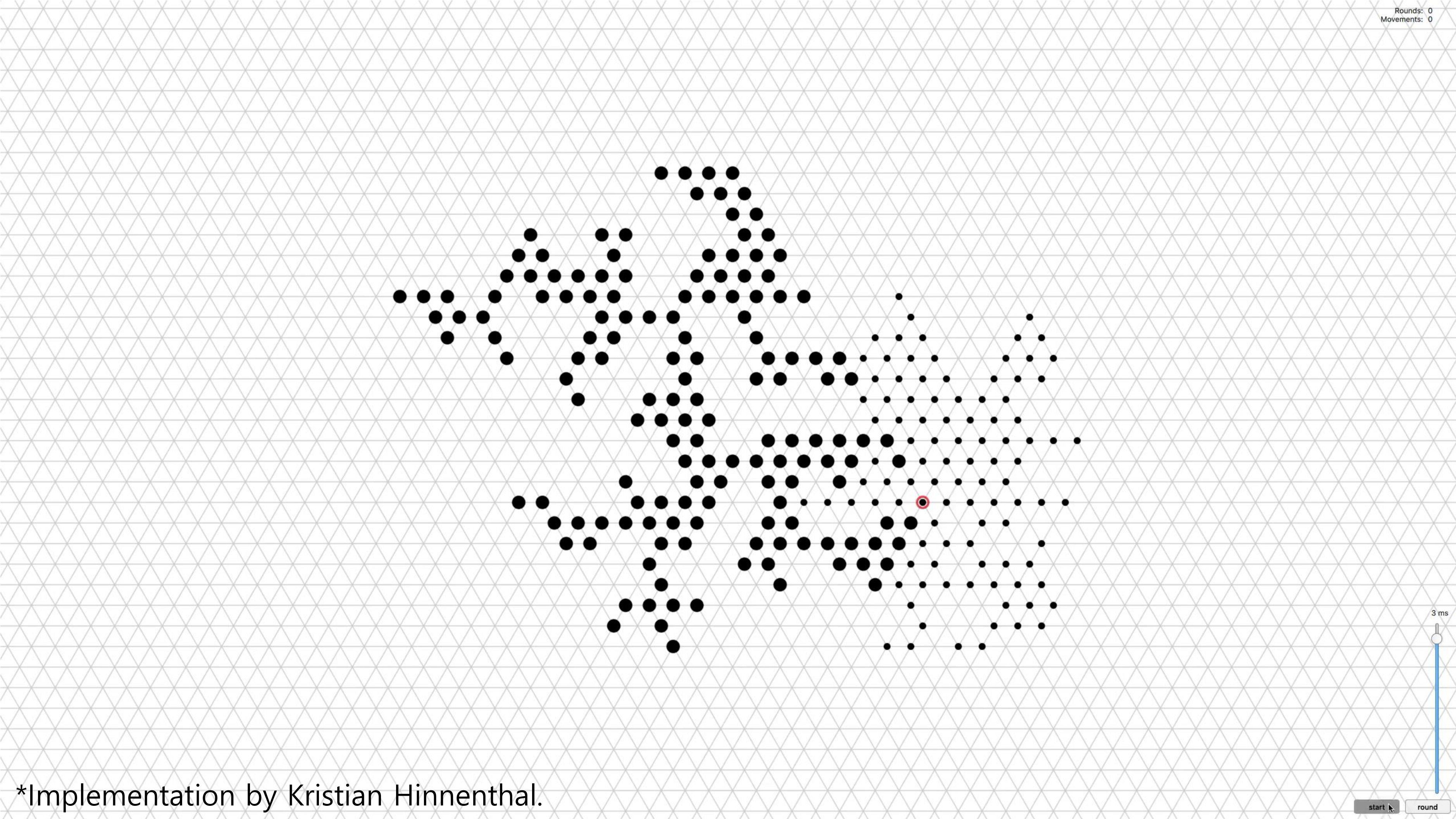
But distances are too big for constant-size memory! So we use the rest of the amoebot system as the leader's distributed memory.

Convex Hull Formation



*Implementation by Kristian Hinnenthal.

Convex Hull Formation



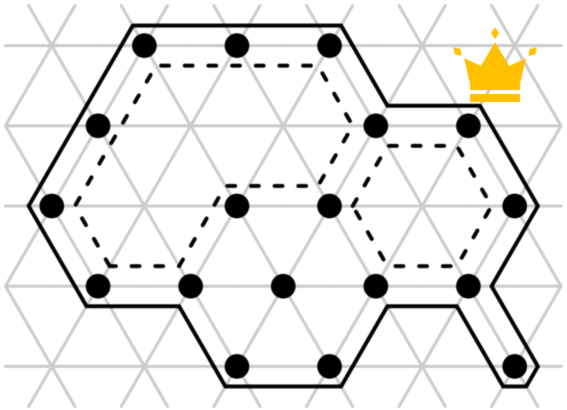
*Implementation by Kristian Hinnenthal.

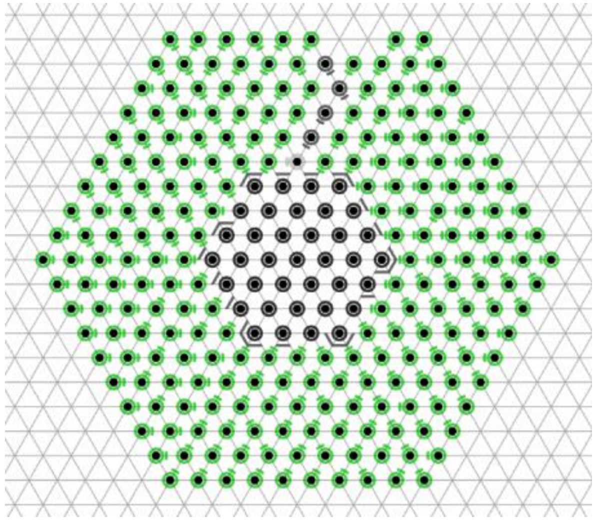
Part I

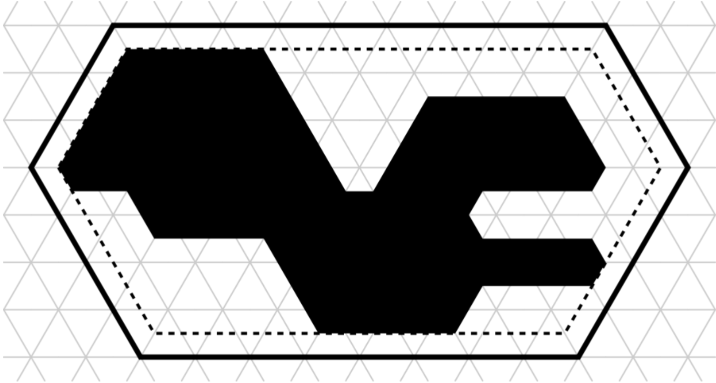
Stateful Distributed Algorithms for Programmable Matter

What are the minimum individual capabilities necessary to achieve system behavior X?

With constant-size memory, neighbor-to-neighbor communication, and local movements, a system can collectively achieve leader election, object coating, and convex hull formation.







Part II

The Amoebot Model and its Enhancements

How can existing algorithms be enhanced to capture more realistic assumptions?

Enhancing the Amoebot Model

The amoebot model and its algorithms do not account for energy costs of the amoebots' actions (**energy-agnostic**) and assume only one amoebot is active at a time (**sequential**).

Real programmable matter systems are **energy-constrained** and **concurrent**.

At a high level, what we'd like is the following:

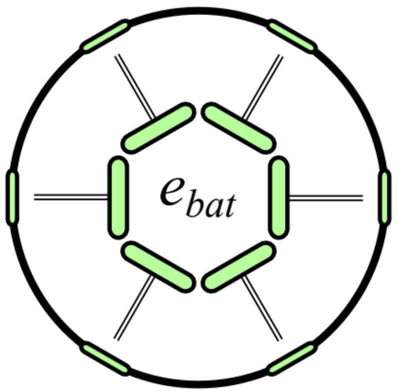
This is too optimistic and may be impossible to guarantee in general, so instead we only consider algorithms $\mathcal{A}$ that obey certain conventions.

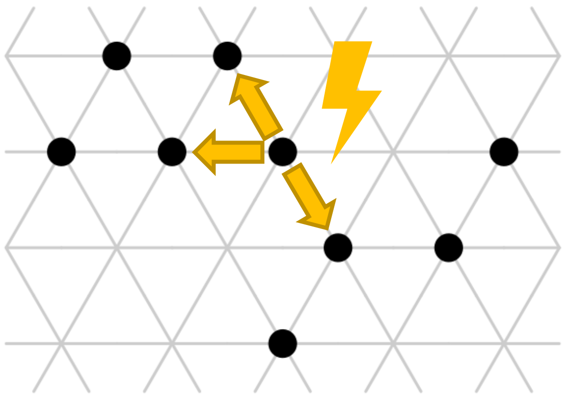
Energy Distribution

Goal: Model energy harvesting, distribution, and usage. Ensure all amoebots eventually get the energy they need to run some algorithm $\mathcal{A}$.

Model Extensions

- Each amoebot A has a constant-size battery $A.e_{bat}$.
- Amoebots with access to an external energy source can directly harvest energy.
- Amoebots can transfer a fixed amount of energy per time to their neighbors without loss.





Energy Distribution

Goal: Model energy harvesting, distribution, and usage. Ensure all amoebots eventually get the energy they need to run some algorithm $\mathcal{A}$.

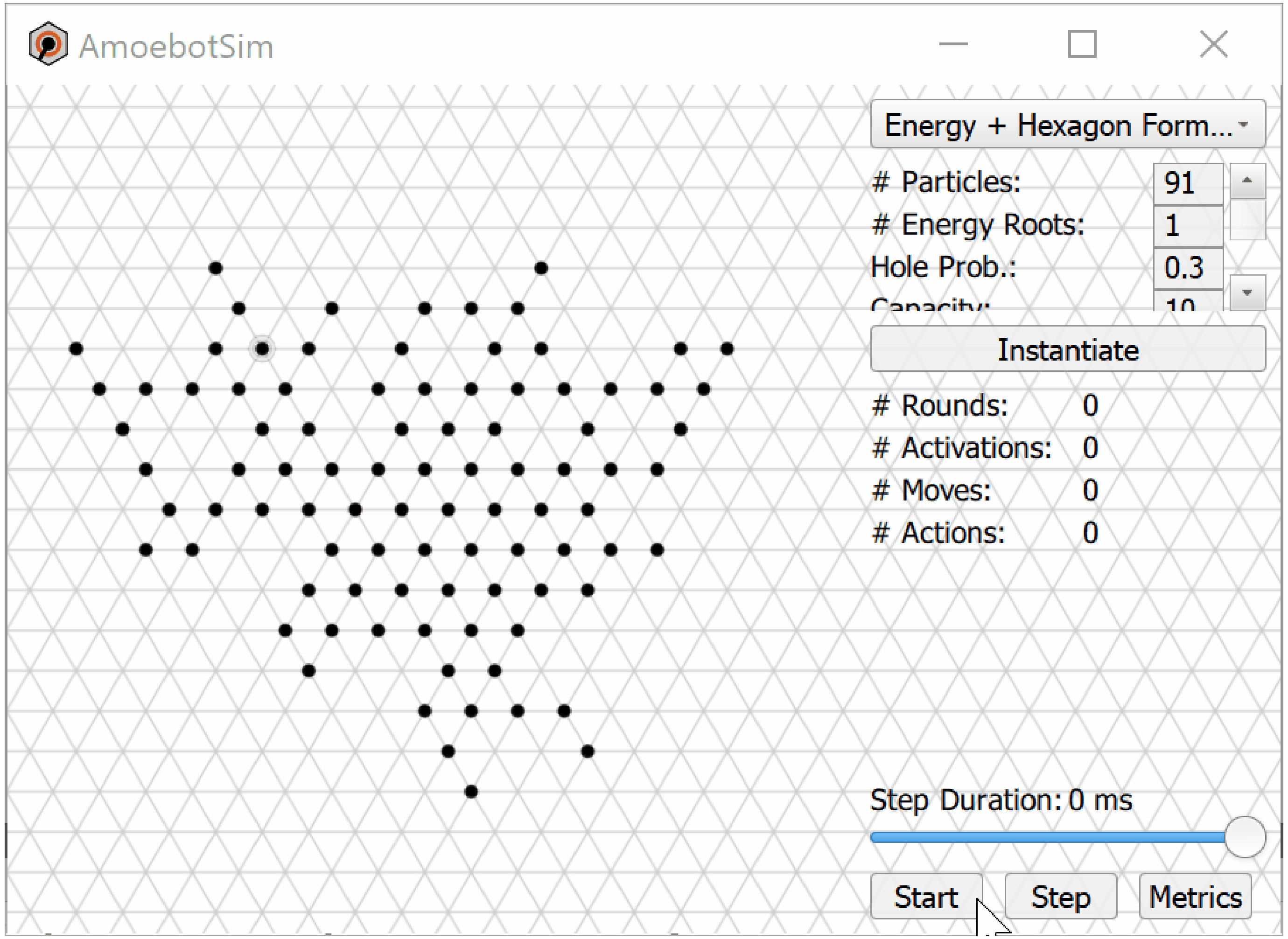
We developed the **Energy-Sharing** algorithm as an asymptotically optimal mechanism for distributing energy to all amoebots in a system.

We then developed the **Forest-Prune-Repair** algorithm as a mechanism for maintaining an underlying spanning forest structure as amoebots move.

*Joint work with Jamison Weber.

Energy-Constrained Shape Formation

Energy-Sharing + Forest-Prune-Repair composed with **Hexagon-Formation**:



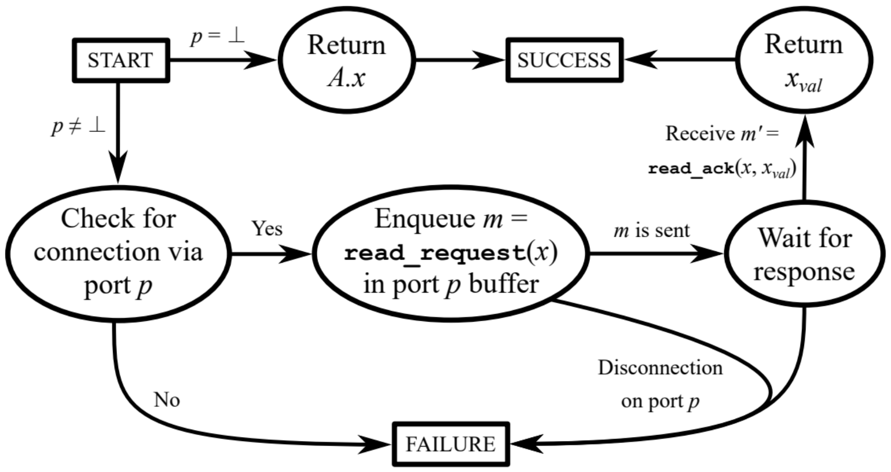
The Canonical Amoebot Model

Goal: Study amoebot algorithms where many amoebots are simultaneously active.

Generalizes the amoebot model by partitioning amoebot functionality into:

- A higher-level **application layer** where algorithms are defined in terms of operations.
- A lower-level **system layer** that executes an amoebot's operations via message passing.

Operation	**Return Value on Success**
CONNECTED(p)	TRUE iff a neighboring amoebot is connected via port p
READ(p, x)	The value of x in the public memory of this amoebot if $p = \perp$ or of the neighbor incident to port p otherwise
WRITE(p, x, x_{val})	Confirmation that the value of x was updated to x_{val} in the public memory of this amoebot if $p = \perp$ or of the neighbor incident to port p otherwise
CONTRACT(v)	Confirmation of the contraction out of node $v \in \{\text{HEAD}, \text{TAIL}\}$
EXPAND(p)	Confirmation of the expansion into the node incident to port p
PULL(p)	Confirmation of the pull handover with the neighbor incident to port p
PUSH(p)	Confirmation of the push handover with the neighbor incident to port p
LOCK()	Local identifiers of the amoebots that were successfully locked
UNLOCK($\mathcal{L}$)	Confirmation that the amoebots of $\mathcal{L}$ were unlocked



(a) READ(p, x)

The Canonical Amoebot Model

Algorithms in the canonical amoebot model are specified in terms of **actions**:

$$\langle \mathit{label} \rangle : \langle \mathit{guard} \rangle \rightarrow \langle \mathit{operations} \rangle$$

- *label* specifies the action's name.
- *guard* is a Boolean predicate determining whether this action is currently **enabled**.
- *operations* specifies the computation and sequence of operations to perform if enacted.

Example from Hexagon-Formation:

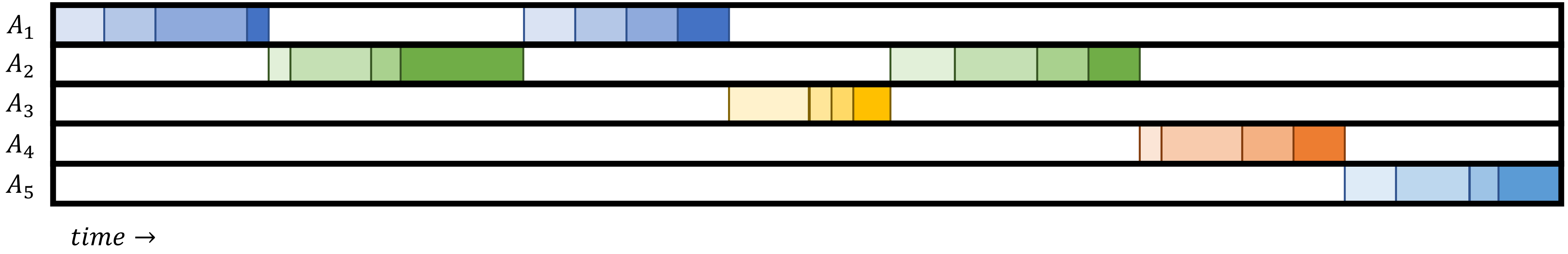
$$\underline{\alpha_2} : \underline{(A.\text{state} = \text{IDLE}) \wedge (\exists B \in N(A) : B.\text{state} \in \{\text{FOLLOWER}, \text{ROOT}\})} \rightarrow$$

Find a port p for which $\textsc{Connected}(p) = \text{TRUE}$ and $\textsc{Read}(p, \text{state}) \in \{\text{FOLLOWER}, \text{ROOT}\}$.
$\textsc{Write}(\bot, \text{parent}, p)$.
$\textsc{Write}(\bot, \text{state}, \text{FOLLOWER})$.

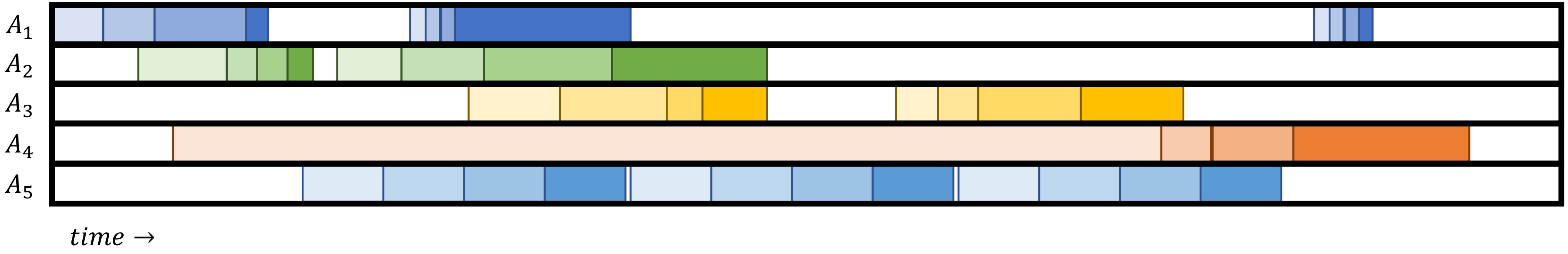
The Canonical Amoebot Model

We use an **adversary** to model timing and progress. Two primary levels of concurrency:

Sequential. At most one active amoebot per time.



Asynchronous. Arbitrary sets of amoebots can be simultaneously active.

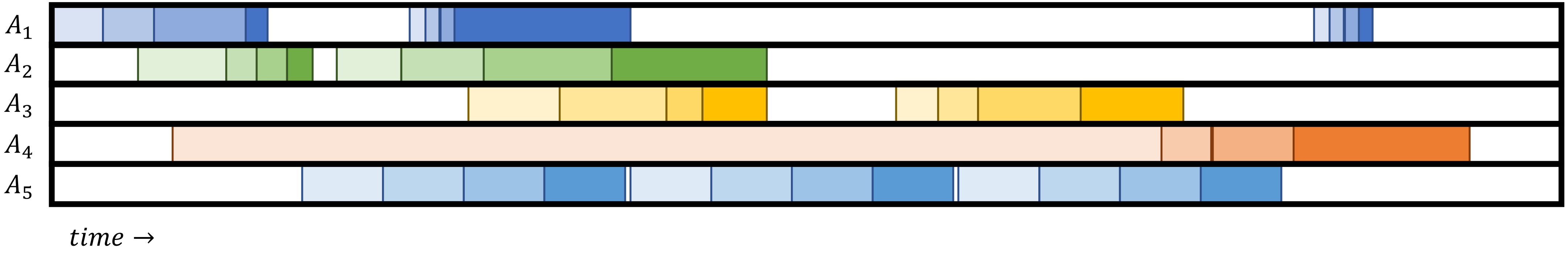


The Canonical Amoebot Model

An **unfair** adversary can activate any amoebot with an enabled action.

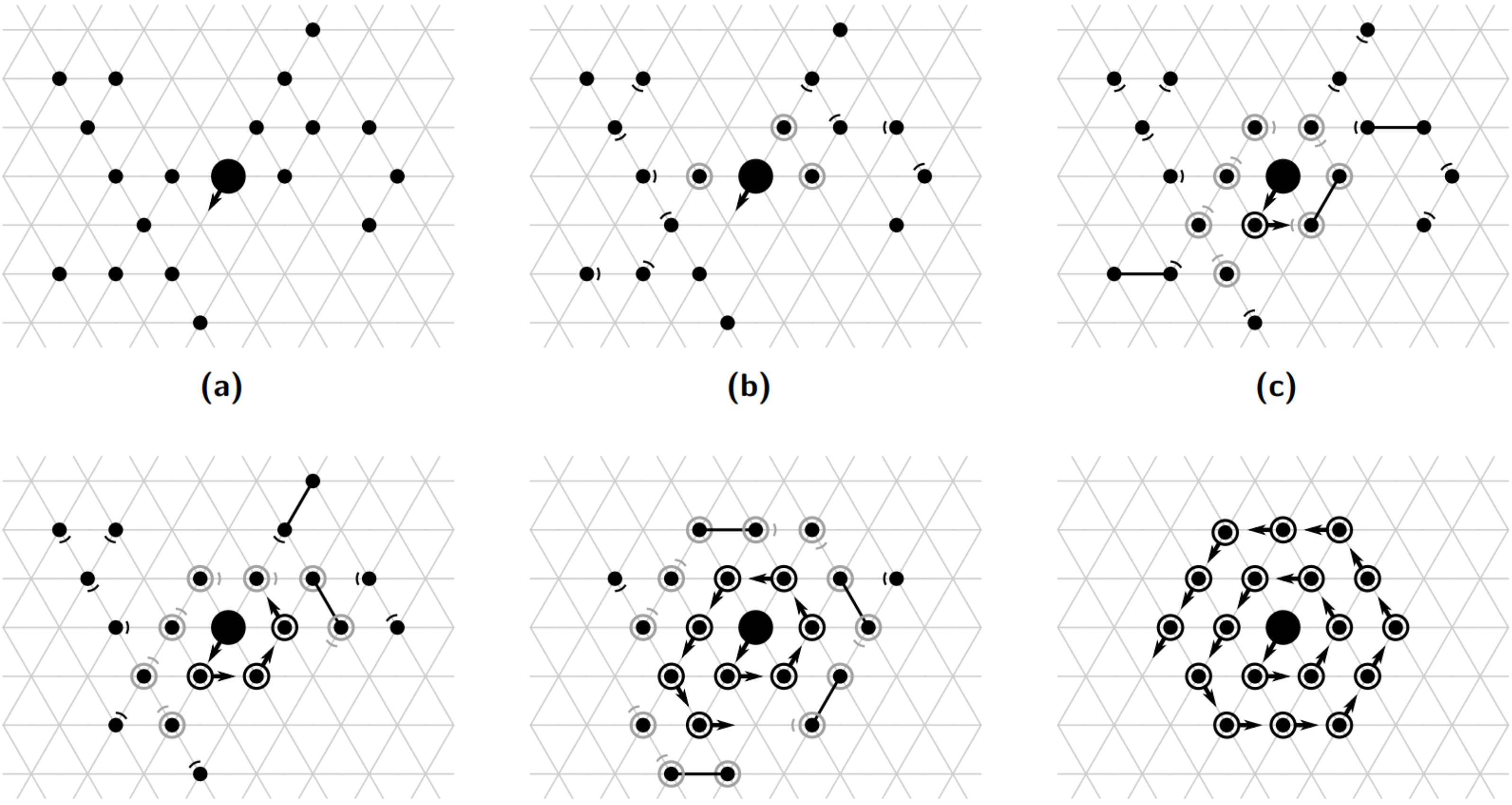
The rest of this talk will primarily focus on unfair asynchronous adversaries, the most general of all adversarial activation models.

Informally: <u>the adversary can activate any amoebot with something to do whenever it wants to.</u>



Asynchronous Hexagon Formation

Problem: Reconfigure any connected amoebot system as a regular hexagon, assuming there is a unique seed amoebot initially in the system.



(a) (b) (c)

(d) (e) (f)

Asynchronous Hexagon Formation

We formulate a Hexagon-Formation algorithm in terms of actions based on [DGRSS 2015].

Algorithm 1 Hexagon-Formation for Amoebot A

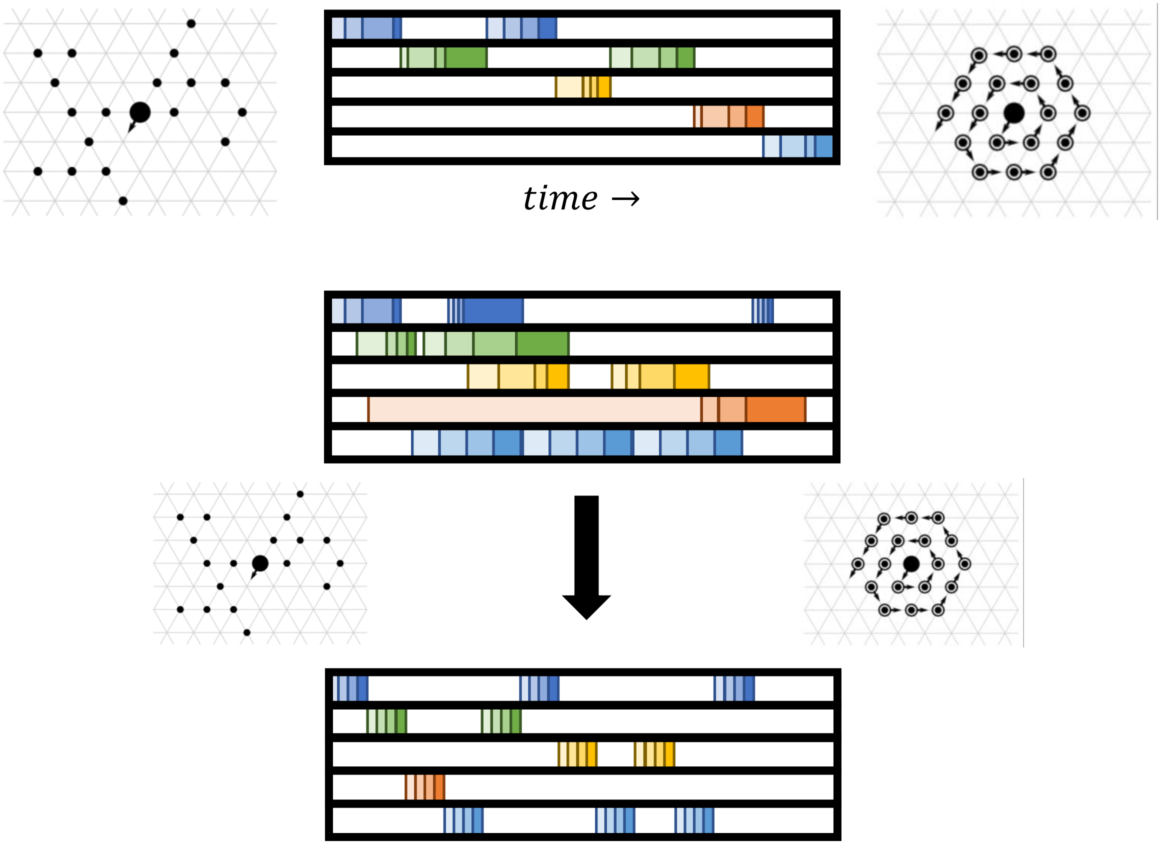
$\alpha_1 : (A.\text{state} \in \{\text{IDLE}, \text{FOLLOWER}\}) \wedge (\exists B \in N(A) : B.\text{state} \in \{\text{SEED}, \text{RETIRED}\}) \to$
  $\textsc{Write}(\perp, \text{parent}, \text{NULL})$.
  $\textsc{Write}(\perp, \text{state}, \text{ROOT})$.
  $\textsc{Write}(\perp, \text{dir}, \textsc{GetNextDir}(\text{counter-clockwise}))$. ▷ See Algorithm 2.
$\alpha_2 : (A.\text{state} = \text{IDLE}) \wedge (\exists B \in N(A) : B.\text{state} \in \{\text{FOLLOWER}, \text{ROOT}\}) \to$
  Find a port p for which $\textsc{Connected}(p) = \text{TRUE}$ and $\textsc{Read}(p, \text{state}) \in \{\text{FOLLOWER}, \text{ROOT}\}$.
  $\textsc{Write}(\perp, \text{parent}, p)$.
  $\textsc{Write}(\perp, \text{state}, \text{FOLLOWER})$.
$\alpha_3 : (A.\text{shape} = \text{CONTRACTED}) \wedge (A.\text{state} = \text{ROOT}) \wedge (\forall B \in N(A) : B.\text{state} \neq \text{IDLE})$
  $\wedge\ (\exists B \in N(A) : (B.\text{state} \in \{\text{SEED}, \text{RETIRED}\}) \wedge (B.\text{dir is connected to } A)) \to$
  $\textsc{Write}(\perp, \text{dir}, \textsc{GetNextDir}(\text{clockwise}))$.
  $\textsc{Write}(\perp, \text{state}, \text{RETIRED})$.
$\alpha_4 : (A.\text{shape} = \text{CONTRACTED}) \wedge (A.\text{state} = \text{ROOT}) \wedge (\text{the node adjacent to } A.\text{dir is empty}) \to$
  $\textsc{Expand}(A.\text{dir})$.
$\alpha_5 : (A.\text{shape} = \text{EXPANDED}) \wedge (A.\text{state} \in \{\text{FOLLOWER}, \text{ROOT}\}) \wedge (\forall B \in N(A) : B.\text{state} \neq \text{IDLE})$
  $\wedge\ (A \text{ has a tail-child } B : B.\text{shape} = \text{CONTRACTED}) \to$
  if $\textsc{Read}(\perp, \text{state}) = \text{ROOT}$ **then** $\textsc{Write}(\perp, \text{dir}, \textsc{GetNextDir}(\text{counter-clockwise}))$.
  Find a port $p \in \textsc{TailChildren}()$ s.t. $\textsc{Read}(p, \text{shape}) = \text{CONTRACTED}$. ▷ See Algorithm 2.
  Let p' be the label of the tail-child's port that will be connected to p after the pull handover.
  $\textsc{Write}(p, \text{parent}, p')$.
  $\textsc{Pull}(p)$.
$\alpha_6 : (A.\text{shape} = \text{EXPANDED}) \wedge (A.\text{state} \in \{\text{FOLLOWER}, \text{ROOT}\}) \wedge (\forall B \in N(A) : B.\text{state} \neq \text{IDLE})$
  $\wedge\ (A \text{ has no tail-children}) \to$
  if $\textsc{Read}(\perp, \text{state}) = \text{ROOT}$ **then** $\textsc{Write}(\perp, \text{dir}, \textsc{GetNextDir}(\text{counter-clockwise}))$.
  $\textsc{Contract}(\text{TAIL})$.

Asynchronous Hexagon Formation

Theorem. Hexagon-Formation (HF) is correct under an unfair asynchronous adversary.

Outline of analysis:

- HF is correct under an unfair sequential adversary.
- Enabled actions of HF remain enabled despite concurrent executions.
- Enabled actions of HF are executed identically in sequential and asynchronous settings.
- Any asynchronous execution of HF can be serialized.
- Any asynchronous execution of HF terminates.



Asynchronous Hexagon Formation

The combination of:

- Correctness under an unfair sequential adversary,
- Enabled actions remaining enabled despite concurrency, and
- Enabled actions executing identically in sequential and asynchronous settings

immediately yields serializability and asynchronous termination, which in turn yield asynchronous correctness.

A General Framework for Concurrency Control

Another approach to concurrency control: use **locks** to mitigate changes to an amoebot's neighborhood while it is active.

We developed a novel algorithm for **mutual exclusion** (locking) in asynchronous, anonymous, dynamic (moving), constant-size memory message passing systems.

Key Idea:

- On activation, an amoebot A first attempts to lock its neighborhood.
- If successful, its locked neighbors cannot move or change their memory contents.
- So A can evaluate its guards and perform its actions as if things were sequential (sort of).
- Failed locking attempts and expansions have no effect on the rest of the system.

Key Issue: Locks can't stop amoebots from expanding into an acting amoebot's neighborhood!

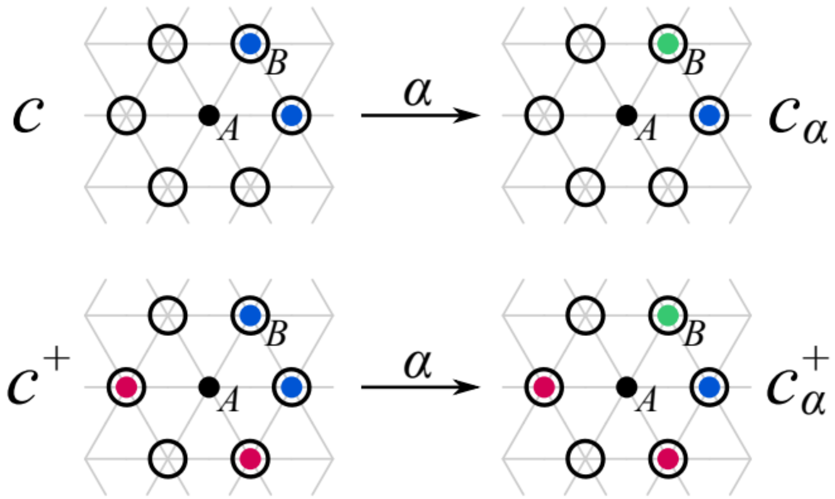
A General Framework for Concurrency Control

We introduce a set of conventions that must be satisfied for the protocol to apply.

Convention 1: Any execution of an enabled action must succeed in the sequential setting.

Convention 2: All compute operations must precede at most one movement operation.

Convention 3: **Monotonicity**. Action executions cannot be affected by (unlocked) amoebots that concurrently enter the acting amoebot's neighborhood.



Static algorithms (i.e., those that don't involve movement) are trivially monotonic. This includes most of the leader election algorithms.

Open Question: What amoebot algorithms satisfy monotonicity?

A General Framework for Concurrency Control

1. Validity. Any execution of an enabled action succeed in the sequential setting.
2. Computing Before Moving. Compute operations precede movement operations.
3. Monotonicity. Action executions are not affected by (unlocked) amoebots that concurrently enter the acting amoebot's neighborhood.

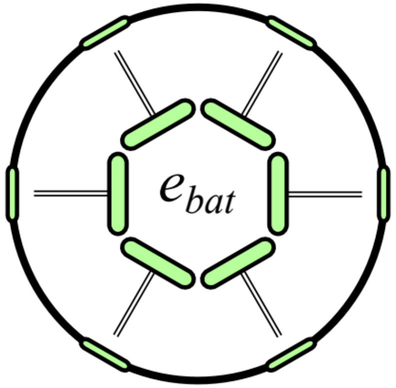
Theorem. Consider any algorithm $\mathcal{A}$ satisfying Conventions 1-3 and let $\mathcal{A}'$ be the algorithm obtained by the concurrency control protocol. If $\mathcal{A}$ terminates under any sequential execution, then every asynchronous execution of $\mathcal{A}'$ terminates in some sequential outcome of $\mathcal{A}$.

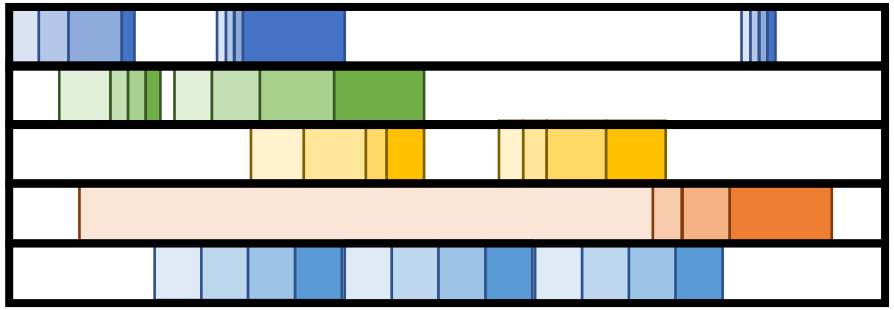
Part II

The Amoebot Model and its Enhancements

How can existing algorithms be enhanced to capture more realistic assumptions?

By satisfying certain conventions, energy-agnostic, sequential algorithms can be made energy-constrained and asynchronous.



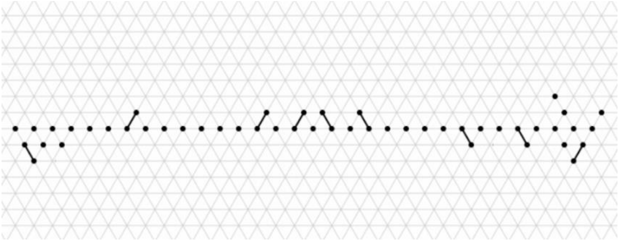


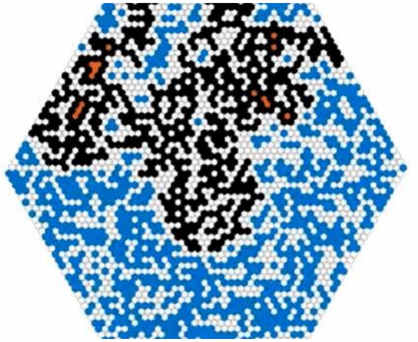
Part III

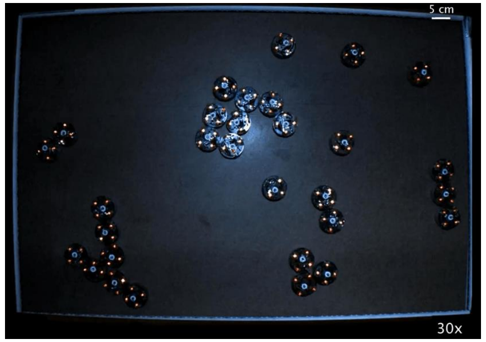
Stochastic Distributed Algorithms & Their Applications to Swarm Robotics and Granular Active Matter

How can digital algorithms be translated for simple, analog (passive) systems?

Connect biased random decisions to the physics of local interactions.

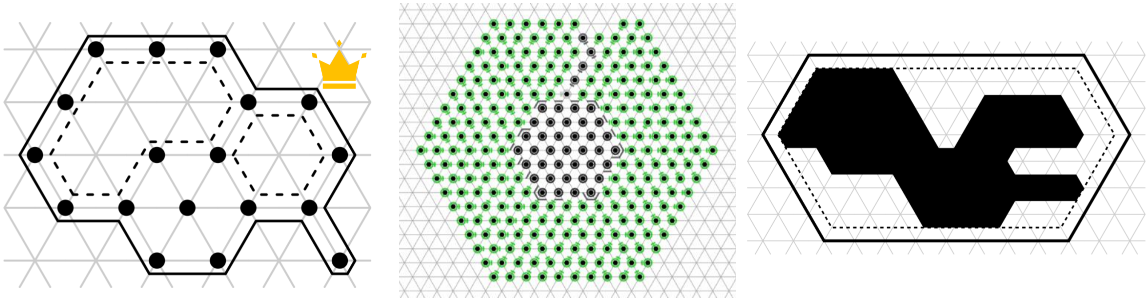






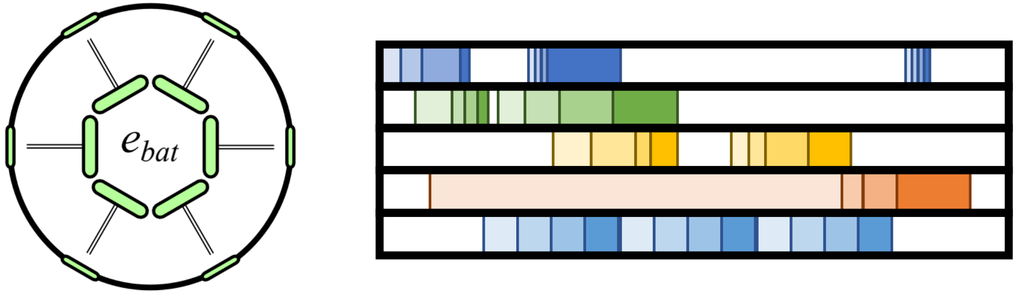
Conclusion: Algorithmic Foundations of Programmable Matter

1. What are the minimum individual capabilities necessary to achieve system behavior X?



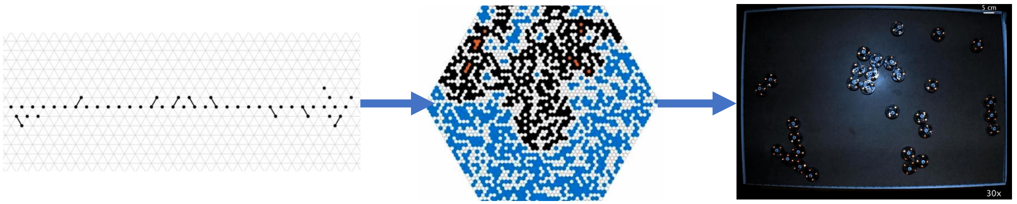
Constant-size memory, communication, and local movements suffice for complex behaviors.

2. How can existing algorithms be enhanced to capture more realistic assumptions?



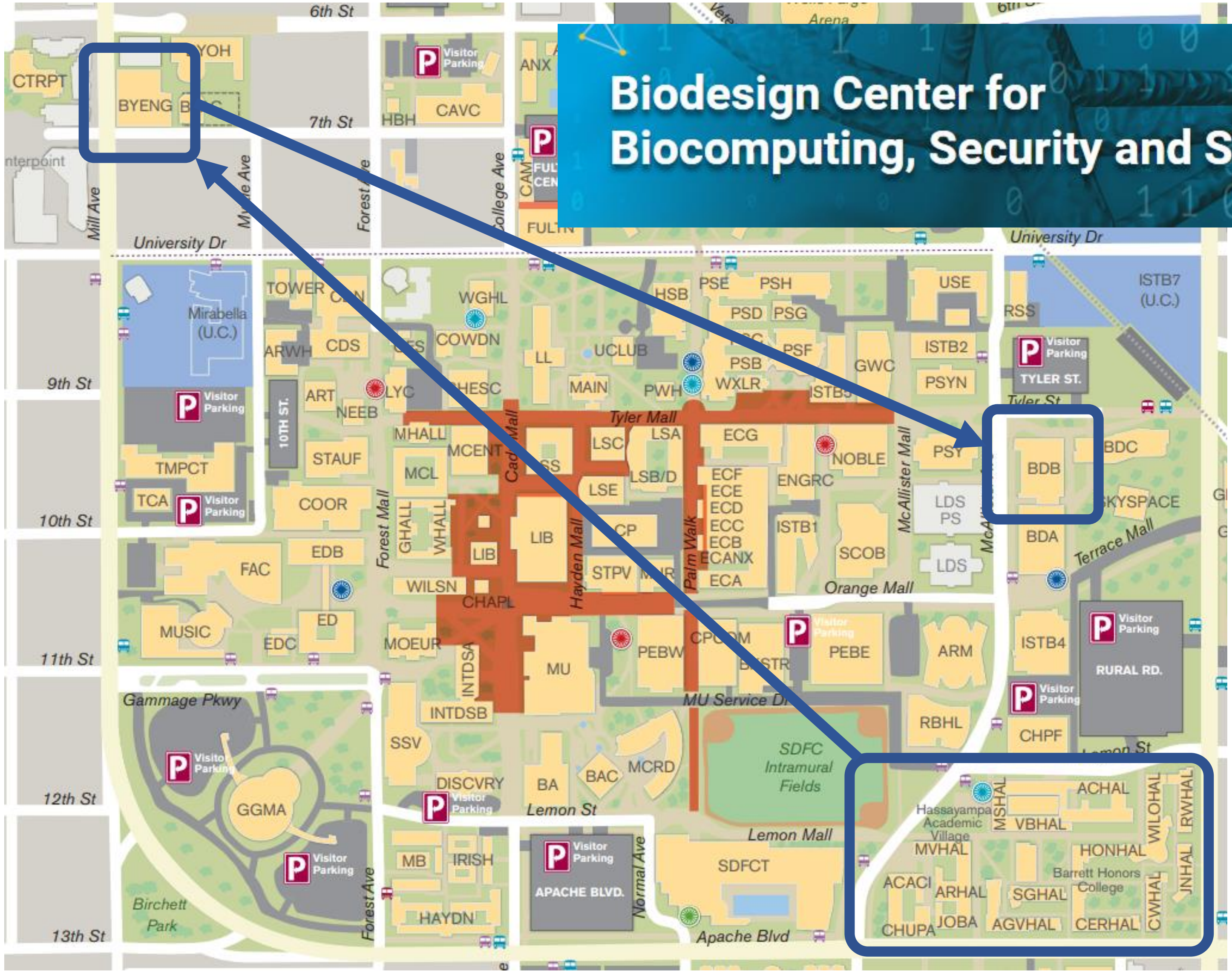
By satisfying certain conventions, energy-agnostic, sequential algorithms can be made energy-constrained and asynchronous.

3. How can digital algorithms be translated for simple, analog (passive) systems?



Connect biased random decisions to the physics of local interactions.

What's Next?



A postdoc with Stephanie Forrest at the ASU Biodesign Institute!

I’ve Got Some People Who Carry Me

Thank you!

sops.engineering.asu.edu

jdaymude.github.io

Arizona State University

ARCS

List of Publications: Dissertation (Chronologically)

Paper	Conference	Journal
"A Markov Chain Algorithm for Compression in Self-Organizing Particle Systems." Cannon, D., Randall, Richa.	PODC 2016	In Preparation
"A Stochastic Approach to Shortcut Bridging in Programmable Matter." Andrés Arroyo, Cannon, D., Randall, Richa.	DNA 2017	Natural Computing
"On the Runtime of Universal Coating for Programmable Matter." D., Derakhshandeh, Gmyr, Porter, Richa, Sheideler, Strothmann.		Natural Computing
"Improved Leader Election for Self-Organizing Programmable Matter." D., Gmyr, Richa, Scheideler, Strothmann.	ALGOSENSORS 2017	
"Phototactic Supersmarticles." Savoie, Cannon, D., Warkentin, Li, Richa, Randall, Goldman.		Artificial Life and Robotics
"A Local Stochastic Algorithm for Separation in Heterogeneous Self-Organizing Particle Systems." Cannon, D., Gökmen, Randall, Richa.	PODC 2018 (BA) RANDOM 2019	In Preparation
"Convex Hull Formation for Programmable Matter." D., Gmyr, Hinnenthal, Kostitsyna, Scheideler, Richa.	ICDCN 2020	
"Bio-Inspired Energy Distribution for Programmable Matter." D., Richa, Weber.	ICDCN 2021	

List of Publications: Dissertation (Chronologically)

Paper	Conference	Journal
"Programming Active Granular Matter with Mechanically Included Phase Changes." Li, Dutta, Cannon, D., Avinery, Aydin, Richa, Goldman, Randall.		Science Advances
"The Canonical Amoebot Model: Algorithms and Concurrency Control." D., Richa, Scheideler.*	In Preparation	

List of Publications: Non-Dissertation (Chronologically)

Paper	Conference	Journal
"Computing by Programmable Particles." D., Hinnenthal, Richa, Scheideler. Book Chapter in Distributed Computing by Mobile Entities.		2018
"Simulation of Programmable Matter Systems Using Active Tile-Based Self-Assembly." Alumbaugh, D., Demaine, Patitz, Richa.	DNA 2019	Natural Computing*
"Preventing Extreme Polarization of Political Attitudes." Axelrod, D., Forrest.		PNAS*
"Mutual Exclusion for Asynchronous, Anonymous, Dynamic, Constant-Size Memory Message Passing Systems." D., Scheideler, Richa.	DISC 2021**	
"AmoebotSim: A Visual Simulator for the Amoebot Model of Programmable Matter." D., Gmyr, Hinnenthal.**		
"Aggregation Without Computation: Negative Results and a Noisy, Discrete Adaptation." D., Harasha, Richa, Yiu.**		

*Under review.

**Manuscript in preparation.