(Almost) Ten Years of the Amoebot Model of Programmable Matter

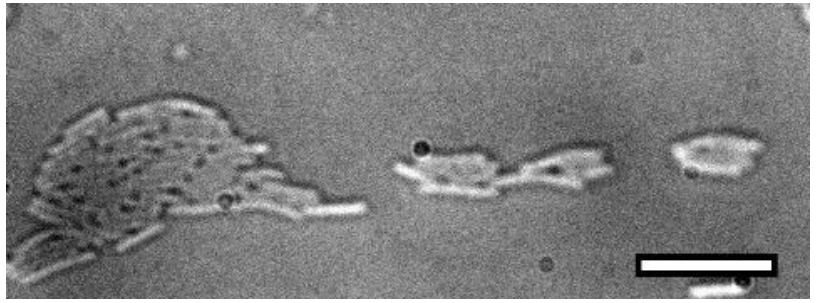
Joshua J. Daymude

SIROCCO 2023 — June 6, 2023

Universidad de Alcalá, Alcalá de Henares, Spain

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

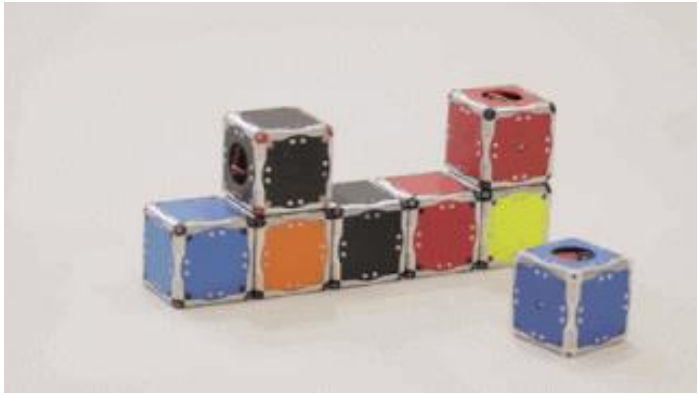
Programmable Matter

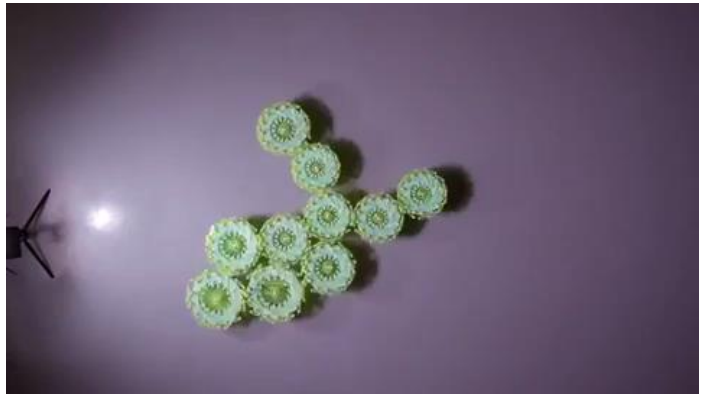
Programmable matter is a substance that can change its physical properties autonomously based on user input or environmental stimuli.

"Catoms"
PB 2018

"Kilobots"
RCN 2014

"M-Blocks"
RGR 2013

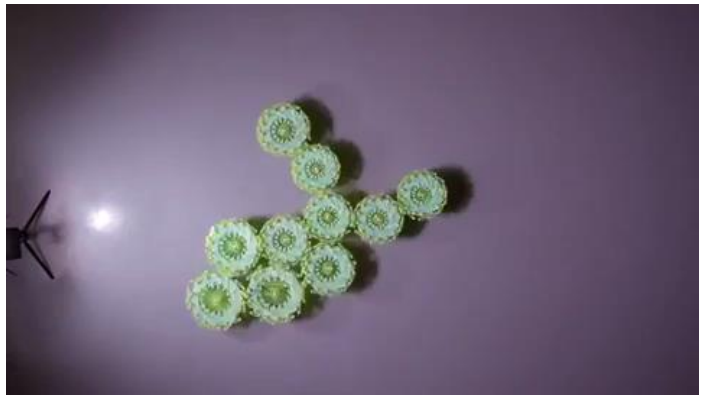




"Particle Robots"
LBBCRHRL 2019

Programmable Matter

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

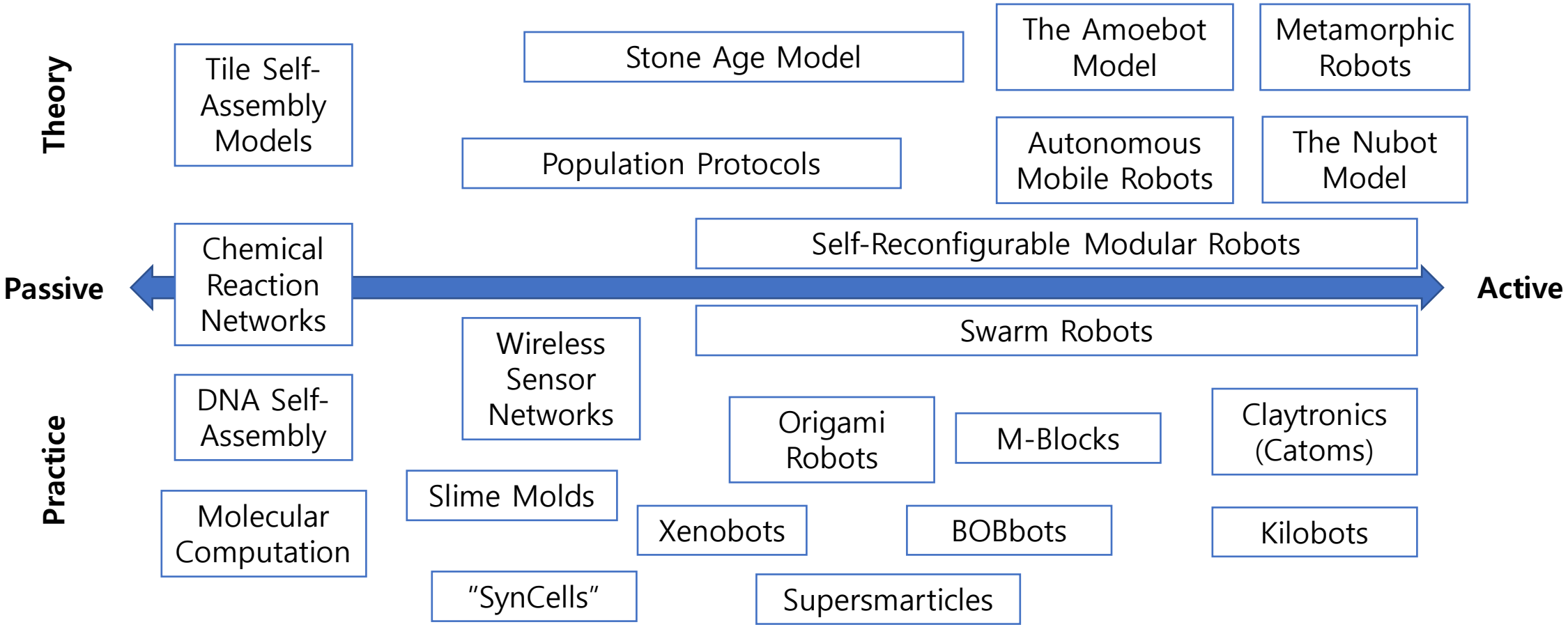


Most programmable matter and modular robotic systems assume:

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

Programmable Matter in Theory and Practice

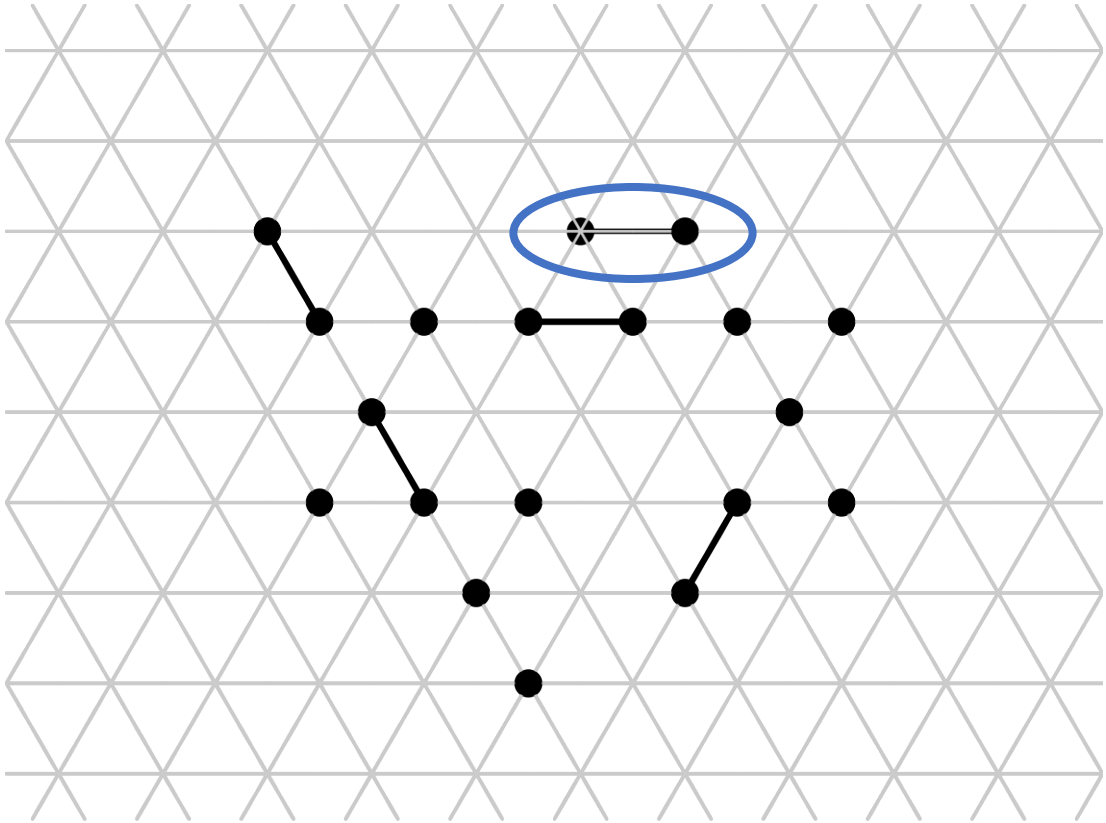
Programmable matter systems can be organized by their degree of self-determination in deciding and enacting local behaviors.



The Amoebot Model: A Typical Setup (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.



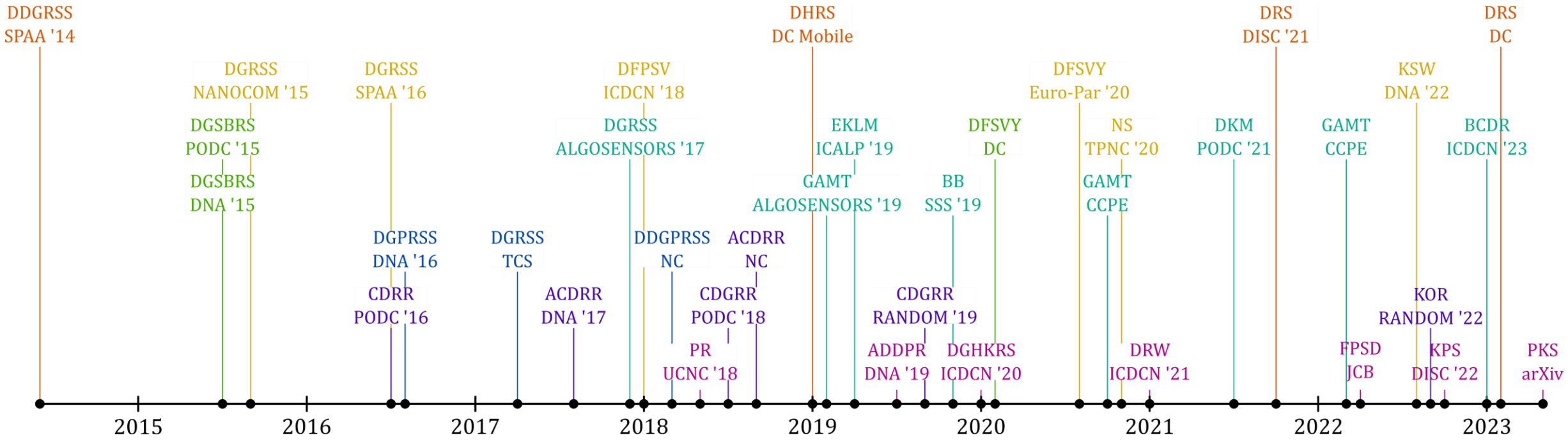
Connections to Other Models

The amoebot model has been studied in relation to:

- **Autonomous mobile robots**, incorporating their Look-Compute-Move cycles into amoebot actions [DFPSV 2018, FPS 2019, DFSVY 2020].
- **Hybrid programmable matter**, where amoebots are replaced by passive "tiles" that are moved around by relatively few robots walking on the structure's surface [GHRKKS 2018, GHKKRSS 2018/20, KLS 2023].
- Any amoebot algorithm can be simulated in the **tile automata (TA)** model (closely related to DNA tiling) [ADDPR 2019], which in turn can be simulated by the **signal-passing tile assembly model (STAM)** [CLSW 2020].

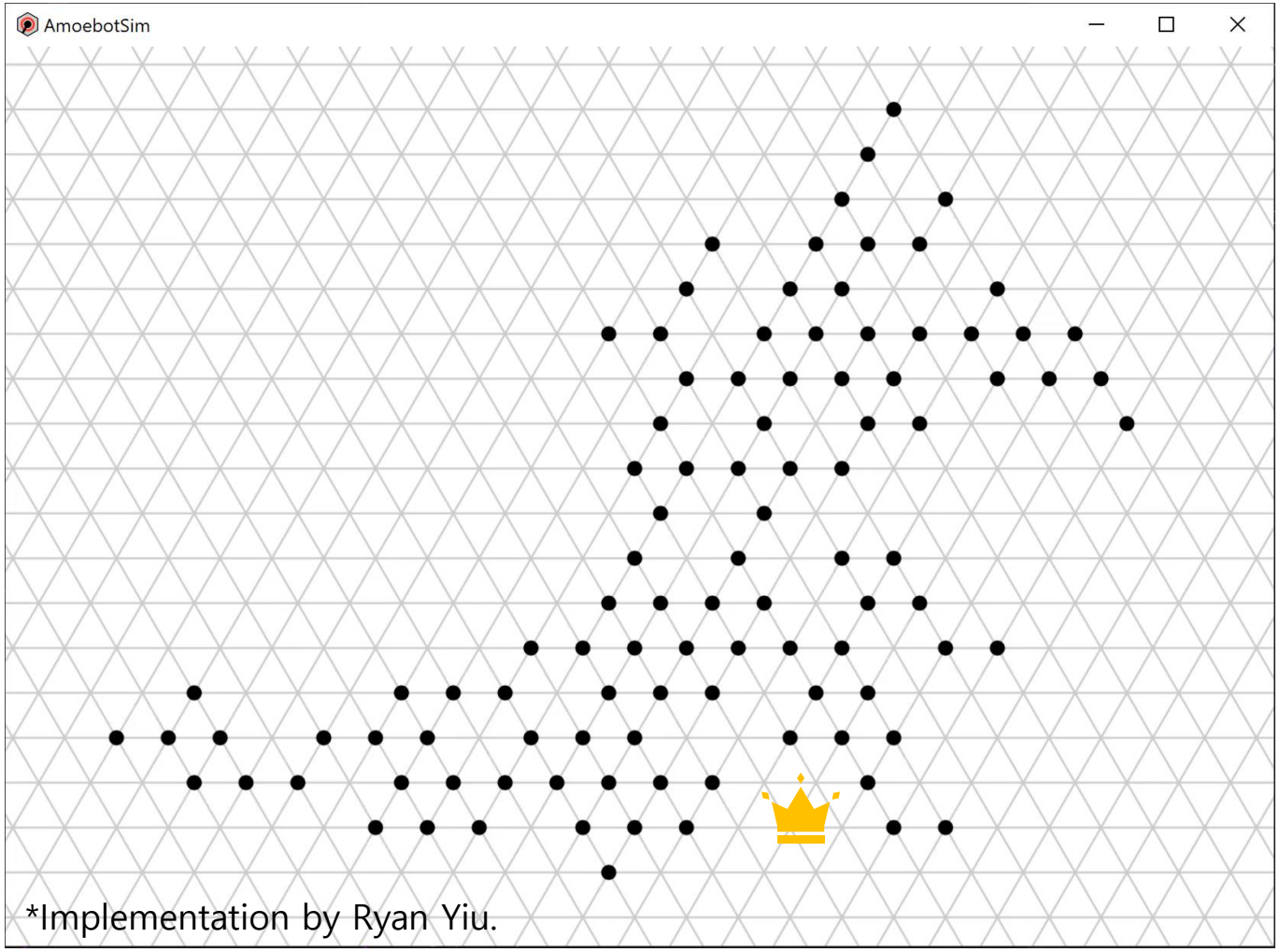
A Visual History of Amoebot Results

40+ papers: Models, shape formation/recognition, leader election, object coating, phase transitions, and more!



Leader Election in the Amoebot Model

Goal. Some amoebot must eventually, irreversibly declare itself the system's unique leader.



Leader Election in the Amoebot Model

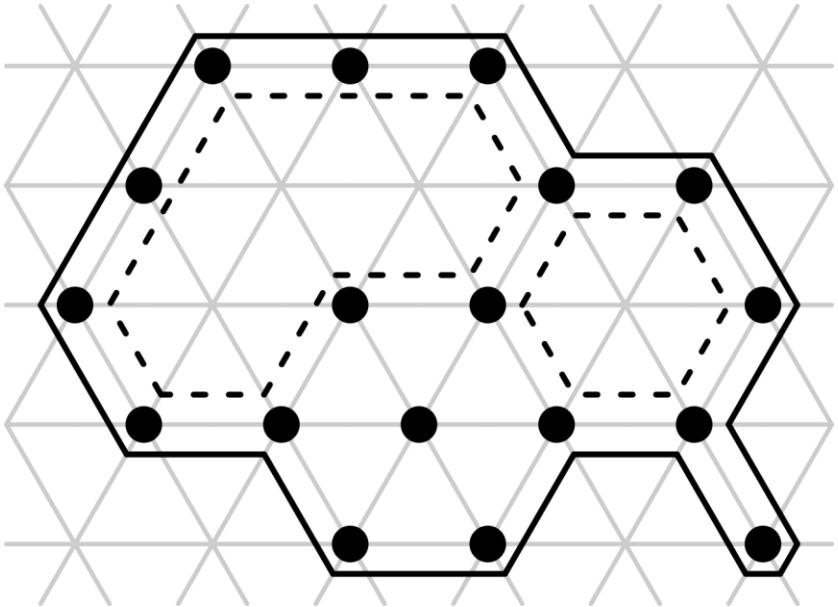
Many algorithms have been developed for different scenarios:

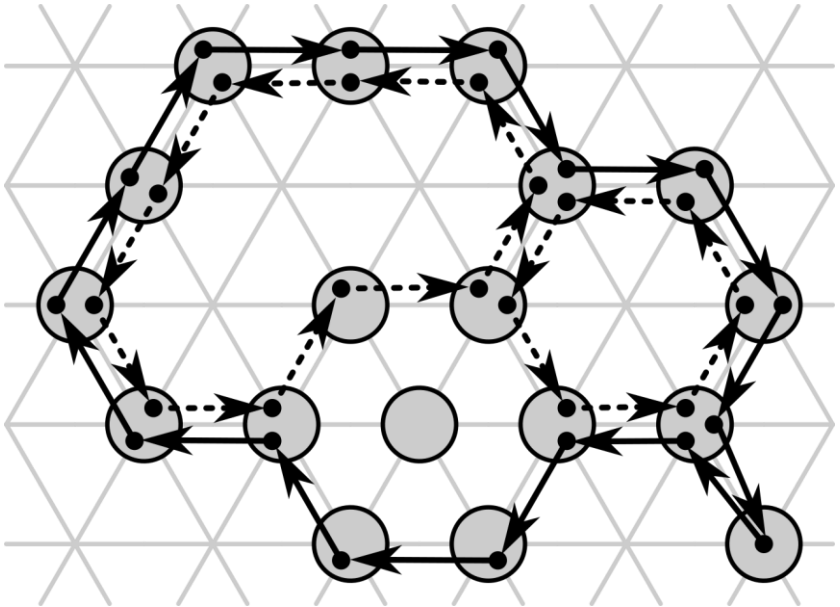
Algorithm	Space	Common Ori.	Adversary	Det.	Holes	Stationary	# Leaders	Runtime
[DGSBRS 2015]	2D	Chirality	Strong Seq.	✗	☑	☑	1	$\mathcal{O}(L^*)$, exp.
[DGRSS 2017]	2D	Chirality	Strong Seq.	✗	☑	☑	1, w.h.p.	$\mathcal{O}(L)$, w.h.p.
[DFSVY 2020]	2D	None	Sync.	☑	✗	☑	$k \leq 3$	$\mathcal{O}(n^2)$
[GAMT 2019]	2D	Chirality	Strong Seq.	☑	✗	☑	1	$\mathcal{O}(r + mtree)$
[EKLM 2019]	2D	None	Strong Seq.	☑	☑	✗	1	$\mathcal{O}(Ln^2)$
[BB 2019]	2D	Chirality	Weak Seq.	☑	☑	☑	$k \leq 6$	$\mathcal{O}(n^2)$
[DKM 2021]	2D	Chirality	Strong Seq.	☑	☑	✗	1	$\mathcal{O}(L + D)$
[GAMT 2022]	3D	View	Strong Seq.	☑	✗	☑	1	$\mathcal{O}(n)$
[BCDR 2023]	2&3D	None	Strong Seq.	☑	✗	☑	1	$\mathcal{O}(n)$

Leader Election in the Amoebot Model: Algorithm Ideas

Approach 1. Compete on the system's unique outer boundary, either with random identifiers (for 1 leader w.h.p.) or deterministically (for k leaders) [DGSBRS 2015, DGSRS 2017, BB 2019].

- Pros: Handles holes, stationary, and achieves the best known runtime (when randomized).
- Cons: Requires common chirality, and the deterministic algorithm is slow.

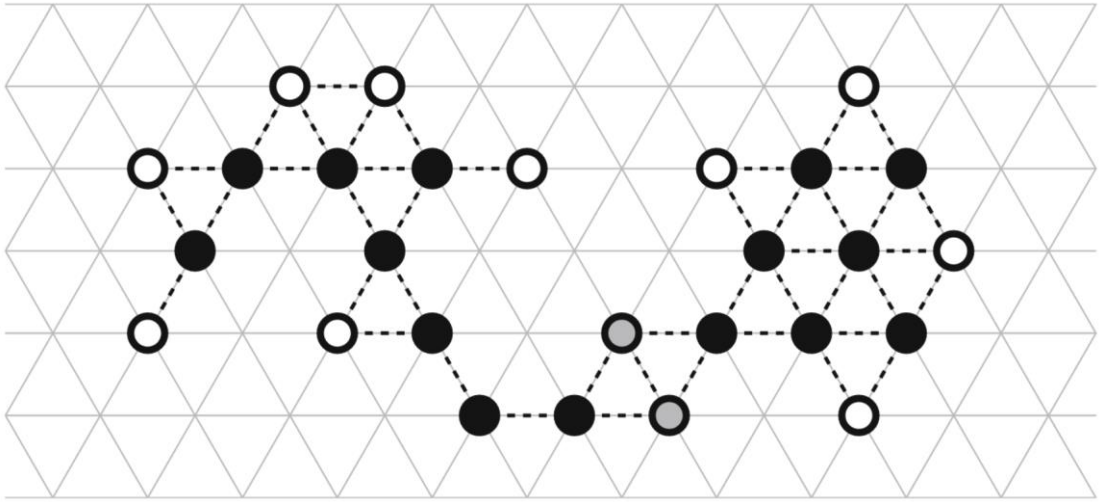


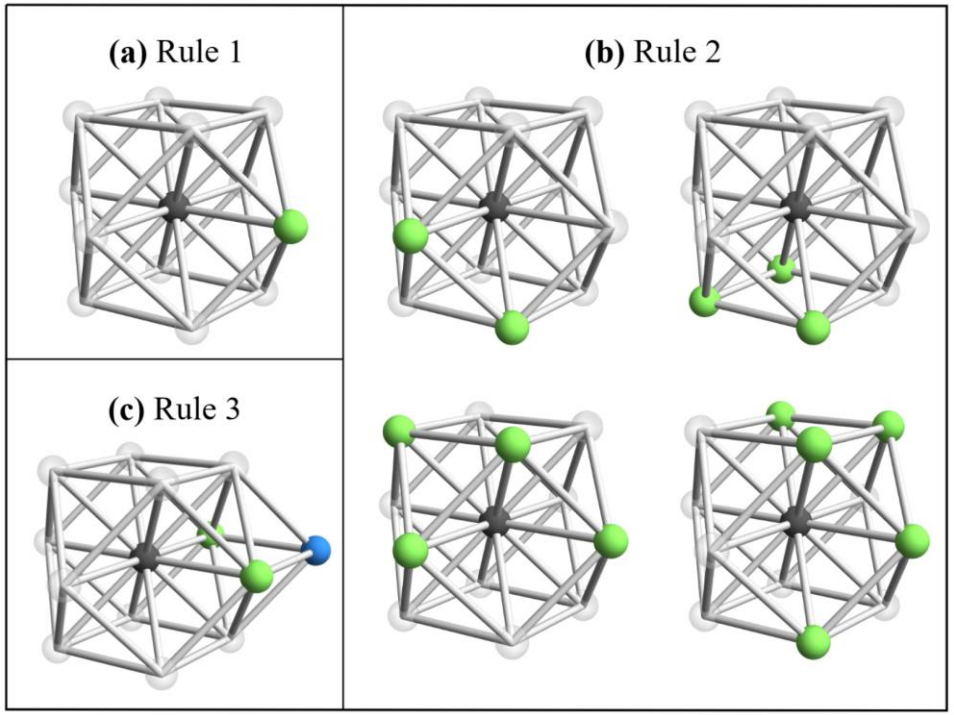


Leader Election in the Amoebot Model: Algorithm Ideas

Approach 2. "Erode" candidate amoebots whose removal leaves a connected structure of candidates until k remain (for k-symmetry). [DFSVY 2020, GAMT 2022, BCDR 2023].

- Pros: Simple, deterministic, handles assorted orientations, and flexible across 2D and 3D.
- Cons: Can't handle holes.





Leader Election in the Amoebot Model: Algorithm Ideas

Approach 3. Use amoebot movements in a clever way (under a sequential adversary) to break otherwise unbreakable symmetries. [EKLM 2019, DKM 2021].

- Pros: Deterministic, handles holes, and always elects a unique leader.
- Cons: Involves movement—which isn't necessarily bad—but makes concurrency more difficult (more on that later). Also, the fast $\mathcal{O}(L + D)$ algorithm requires common chirality.

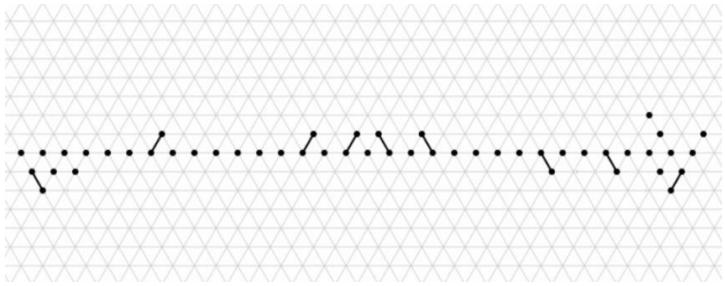
Other Problems with Stateful Distributed Algorithms

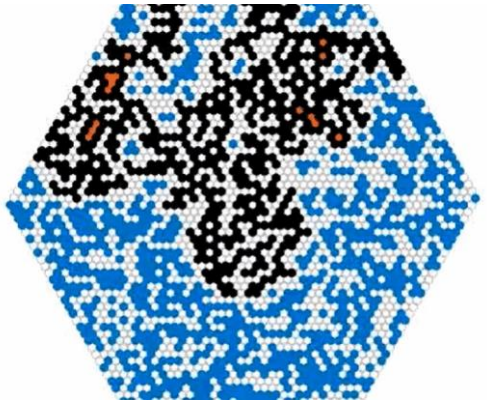
- Basic Shape Formation. Configure the system as some regular shape (e.g., a line, triangle, hexagon, or parallelogram) [DGRSS 2015, DGSBRS 2015, NS 2020, DRS 2021].
- General Shape Formation. Configure the system as a scaled-up version of a shape given as a (constant-size) blueprint of triangles [DGRSS 2016, DFSVY 2020].
- Object Coating. Given a static object, configure the system in as many even layers as possible over its surface [DGRSS 2017, DDGPRSS 2016/18].
- Convex Hull Formation. Given a static object, configure the system as its (restricted-orientation) convex hull [DGHKSR 2020].
- Collaborative Computation. Use amoebots as registers to perform counting, matrix/vector multiplication, or simulate Turing machines [PR 2018, DGHKSR 2020, DFSVY 2020].

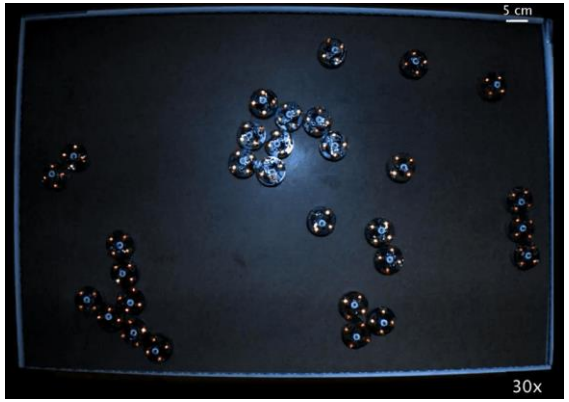
Leveraging Phase Transitions for Collective Behavior

Key Idea. Leverage physical interactions to translate digital algorithms for simple analog robots.

Markov chain $\mathcal{M}_C$ for Compression → Markov chain $\mathcal{M}_A$ for Aggregation → Aggregation in BOBbots







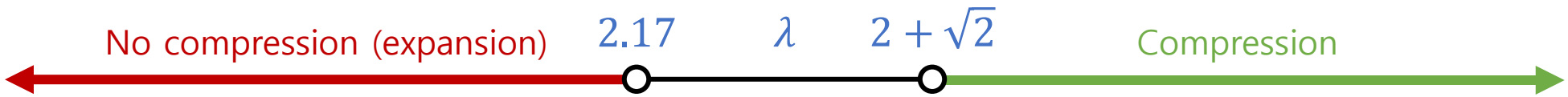
* All robot design, experiment data, and videos are the work of Shengkai Li, Bahnisikha Dutta, Ram Avinery, Enes Aydin, and Daniel I. Goldman.

Leveraging Phase Transitions for Collective Behavior

[CDRRicha 2016]: The distributed, stochastic algorithm for compression:

Fix $\lambda > 1$. Start in any connected configuration. Repeat:

1. Pick a random particle.
2. Pick a random neighboring node.
3. If the proposed node is empty, move with probability $\min\{\lambda^{e'-e}, 1\}$ if connectivity is maintained.
4. Otherwise, do nothing.



Similar results for shortcut bridging, separation, aggregation, alignment, foraging…

Fault Tolerance in the Amoebot Model

The vast majority of amoebot papers (>90%) assume reliable communication and processing.

[DFPSV 2018]: From a particular set of initial configurations, for any number $f \leq n - 4$ of permanently crashed amoebots, the remaining $n - f$ non-faulty amoebots can reform a line if they have access to a fault detector for their neighbors.

[NS 2020]: Similar to the above, but studies under what conditions connectivity is maintained.

[DRW 2021]: A spanning forest structure can be dynamically repaired in spite of crash failures as long as the set of non-faulty amoebots is connected.

[KSW 2022]: Under temporary crash failures, spanning forest formation and basic shape formation remain solvable.

Open Question: What classes of crash faults are actually worth distinguishing? What about Byzantine faults? What about self-stabilization?

The Canonical Amoebot Model [DRS 2021/23]

The **canonical amoebot model** was introduced to standardize the many disparate model assumptions made in 2014–2021. It also addressed concurrency (more on that later).

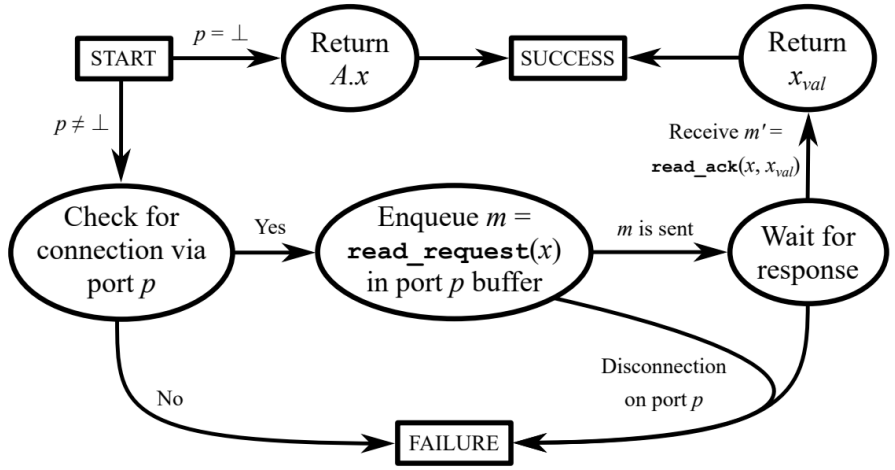
	Variant	Description
Space	General*	G is any infinite, undirected graph.
	Geometric*,†	$G = G_\Delta$, the triangular lattice.
Orientation	Assorted*,†	Assorted direction and chirality.
	Common Chirality*	Assorted direction but common chirality.
	Common Direction	Common direction but assorted chirality.
	Common	Common direction and chirality.
Memory	Oblivious	No persistent memory.
	Constant-Size*,†	Memory size is $\mathcal{O}(1)$.
	Finite	Memory size is $\mathcal{O}(f(n))$, some function of the system size.
	Unbounded	Memory size is unbounded.
Concurrency	Asynchronous†	Any amoebots can be simultaneously active.
	Synchronous*	Any amoebots can simultaneously execute a single action per discrete round. Each round has an evaluation phase and an execution phase.
	k-Isolated	No amoebots within distance k can be simultaneously active.
	Sequential*	At most one amoebot is active per time.
Fairness	Unfair†	Some enabled amoebot is eventually activated.
	Weakly Fair*	Every continuously enabled amoebot is eventually activated.
	Strongly Fair	Every amoebot enabled infinitely often is activated infinitely often.

The Canonical Amoebot Model [DRS 2021/23]

In the canonical amoebot model, amoebot functionality is partitioned 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
CONNECTED()	$[c_0, \ldots, c_{k-1}] \in \{N_1, \ldots, N_8, \text{FALSE}\}^k$ where $c_p = N_i$ if N_i is the locally identified neighbor connected via port p and $c_p = \text{FALSE}$ otherwise
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 this amoebot and the neighbors that were locked
UNLOCK($\mathcal{L}$)	Confirmation that the amoebots of $\mathcal{L}$ were unlocked



(a) READ(p, x)

The Canonical Amoebot Model [DRS 2021/23]

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

$$\langle label \rangle : \langle guard \rangle \rightarrow \langle operations \rangle$$

- $label$ 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:

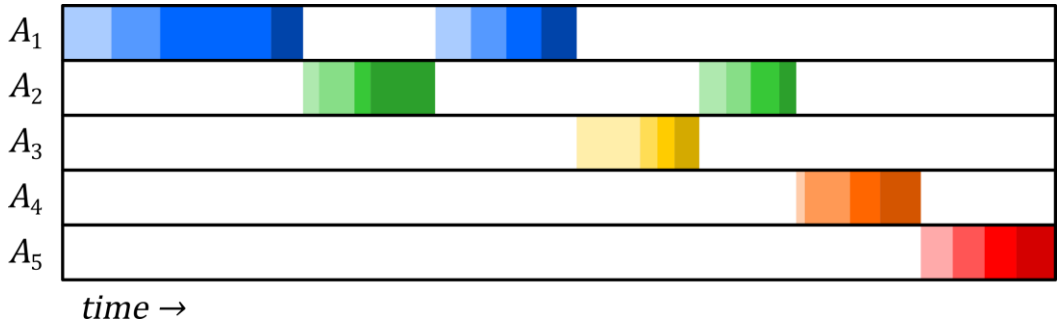
$$\alpha_2 : (A.\text{state} = \textsc{idle}) \wedge (\exists B \in N(A) : B.\text{state} \in \{\textsc{follower}, \textsc{root}\}) \rightarrow$$

Find a port p for which $\textsc{Connected}(p) = \textsc{true}$ and $\textsc{Read}(p, \text{state}) \in \{\textsc{follower}, \textsc{root}\}$.
$\textsc{Write}(\perp, \text{parent}, p)$.
$\textsc{Write}(\perp, \text{state}, \textsc{follower})$.

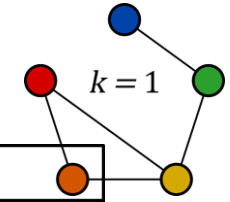
The Canonical Amoebot Model [DRS 2021/23]

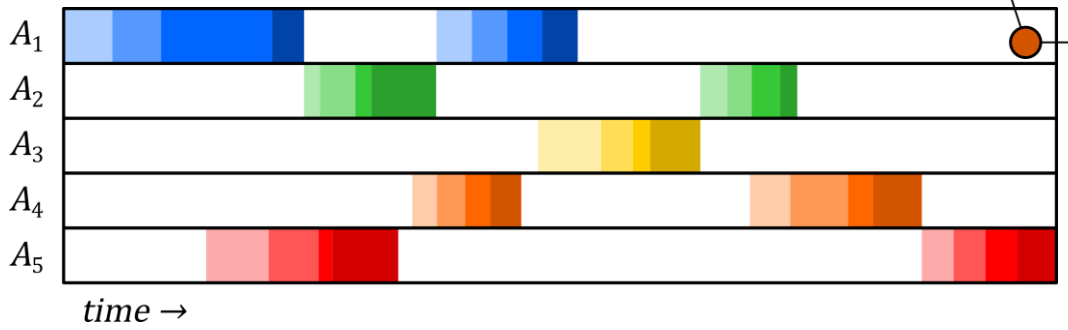
An **adversary** controls the timing of amoebot actions. Four levels of concurrency:

Sequential. At most one amoebot can be active at a time.

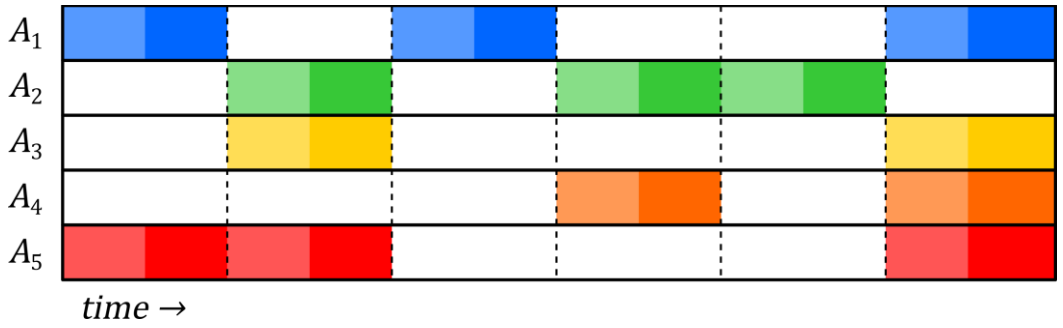


k-isolated. No amoebots within distance k can be simultaneously active.

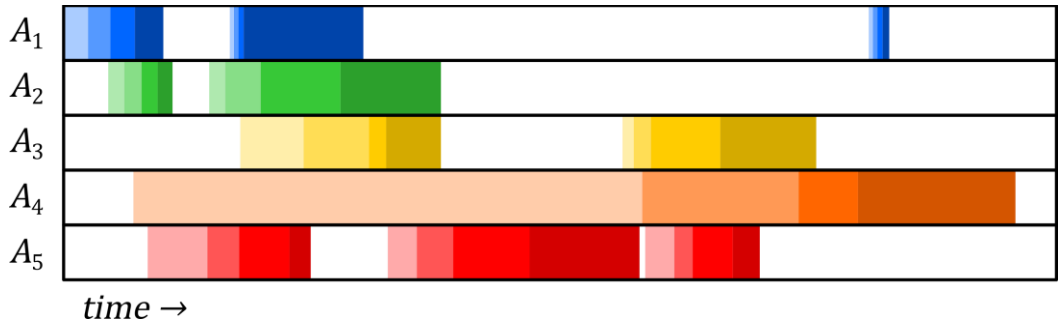




(Semi-)Synchronous. Arbitrary sets of amoebots can be active in each discrete round.



Asynchronous. Arbitrary sets of amoebots can be simultaneously active.



The Canonical Amoebot Model [DRS 2021/23]

The adversary can only activate amoebots with enabled actions. Three levels of fairness:

1. Strongly Fair. Every amoebot that is enabled infinitely often is activated infinitely often.
2. Weakly Fair. Every continuously enabled amoebot is eventually activated.
3. Unfair. Some enabled amoebot is eventually activated.

Informally, a **round** is the time required for the slowest continuously enabled amoebot to execute a single action.

Definition. Let t_i be the time round i starts, and let $\mathcal{E}_i$ be the set of amoebots enabled or executing an action at time t_i. Round i ends at the earliest time $t_{i+1} > t_i$ by which every amoebot in $\mathcal{E}_i$ either completed an action execution or was disabled.

A General Framework for Concurrency Control

Except the semi-synchronous Look-Compute-Move based algorithms of [DFSVY 2020, NS 2020], all algorithms before 2021 were built for sequential, fair adversaries.

The asynchronous message passing of the canonical model lets us study concurrency.

This is the best of both worlds: the ease of designing algorithms in the sequential setting and the relevance of correct execution in the more realistic concurrent setting.

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

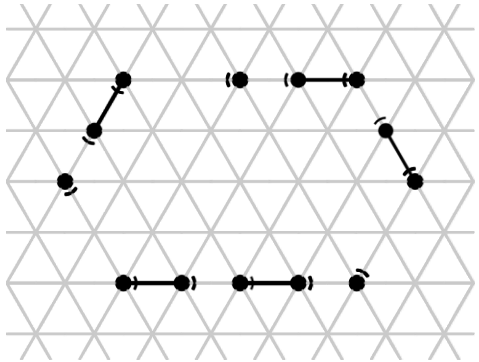
A General Framework for Concurrency Control

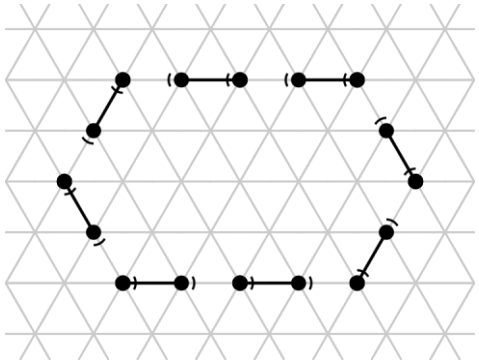
Key Ideas:

- On activation, an amoebot A first attempts to lock its neighborhood [DRS 2022].
- 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!

Example. "If I have no neighbors, then expand in the 'forward' direction."





A General Framework for Concurrency Control

1. Validity. Any execution of an enabled action succeeds in the sequential setting.
2. Phase Structure. Compute operations precede (at most one) movement operation.
3. Expansion-Robustness. Action executions are not affected by (unlocked) amoebots that concurrently enter the acting amoebot's neighborhood.

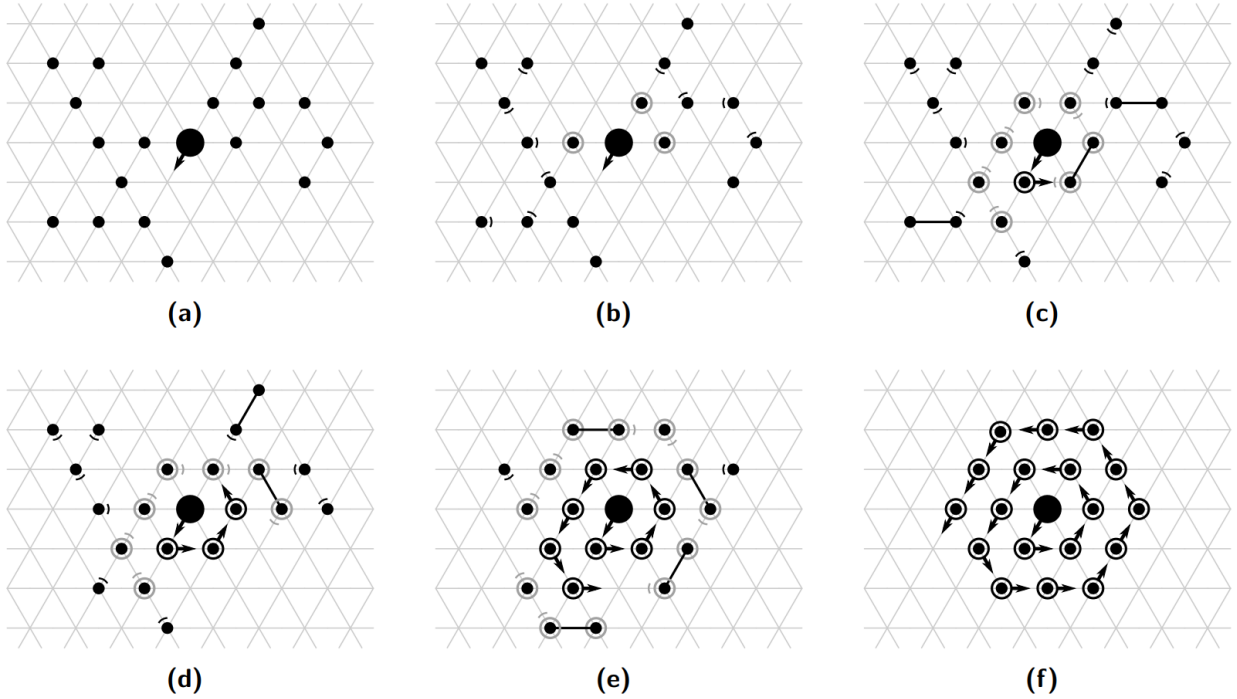
Theorem [DRS 2021/23]. Consider any algorithm $\mathcal{A}$ satisfying Conventions 1–3 and let $\mathcal{A}'$ be the algorithm obtained by the concurrency control framework. If $\mathcal{A}$ terminates under any sequential execution, then every asynchronous execution of $\mathcal{A}'$ terminates in an outcome that some sequential execution of $\mathcal{A}$ also terminates in.

A General Framework for Concurrency Control

Expansion-robustness seems technical and hard. Do any algorithms satisfy it?

Observation. All stationary algorithms (those that don't move) are trivially expansion-robust.

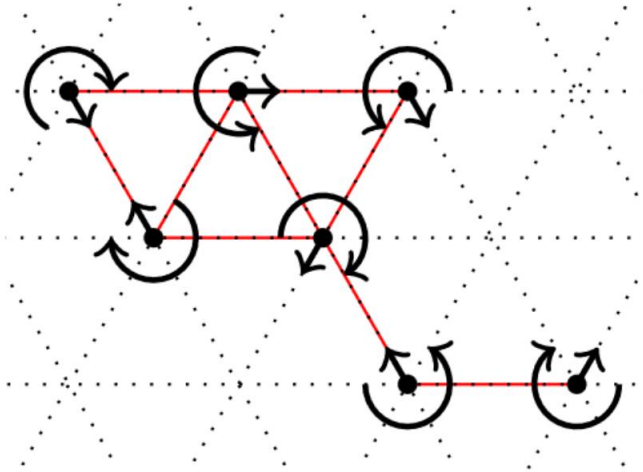
Theorem [DRS 2023]. The Hexagon-Formation algorithm (adapted from [DGRSS 2015]) satisfies expansion robustness, and thus is compatible with the concurrency control framework.

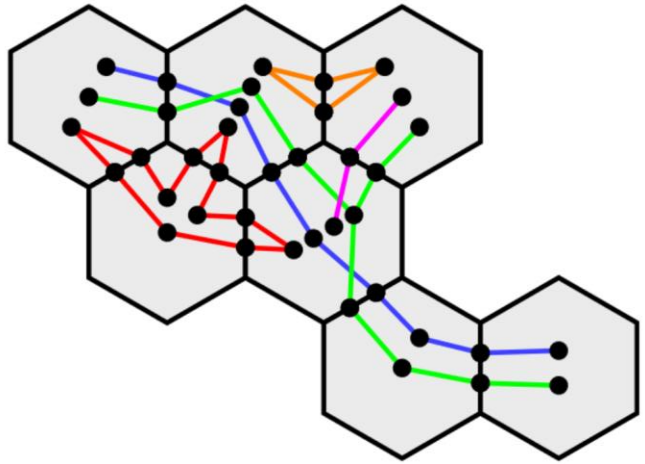


Long-Range Coordination: Reconfigurable Circuits

In the human nervous and muscular systems, cells rapidly coordinate via electrical signals that are propagated along biological "circuits".

The **reconfigurable circuit** extension [FPSD 2021/22] aims to do this for amoebots:





Each amoebot has $k \geq 1$ **pins** per neighbor that it can dynamically organize into circuits.

"Beeping" sends a signal to all amoebots on a circuit in the next synchronous round, but the receiving amoebots do not know the source of this beep or its multiplicity.

Long-Range Coordination: Reconfigurable Circuits

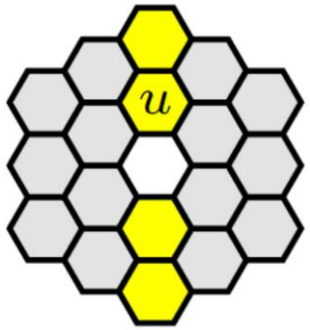
Using reconfigurable circuits yields the first **sublinear** amoebot algorithms for existing problems like leader election, consensus, and shape recognition [FPSD 2021/22]:

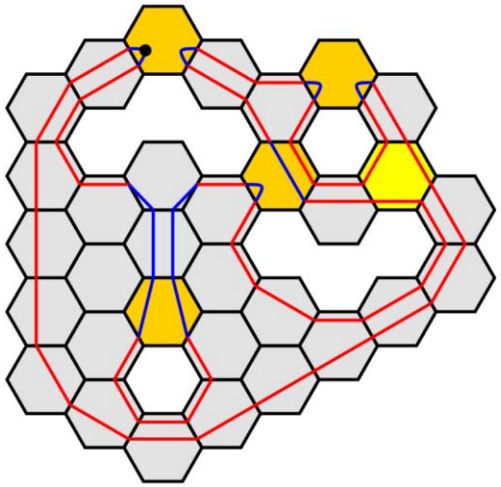
Problem	Minimum required pins	Common chirality	Runtime
Leader election	1	No	$\Theta(\log n)$ w.h.p.
Consensus	1	No	$O(1)$
Compass alignment	1	Yes	$O(\log n)$ w.h.p.
Chirality agreement	2	No	$O(\log n)$ w.h.p.
Shape recognition			
Shapes composed of triangles	1	Yes	$O(1)$
Parallelograms	1	No	$O(1)$
Parallelograms with linear side ratio	1	No	$\Theta(\log n)$ w.h.p.
Parallelograms with polynomial side ratio	2	No	$\Theta(\log n)$ w.h.p

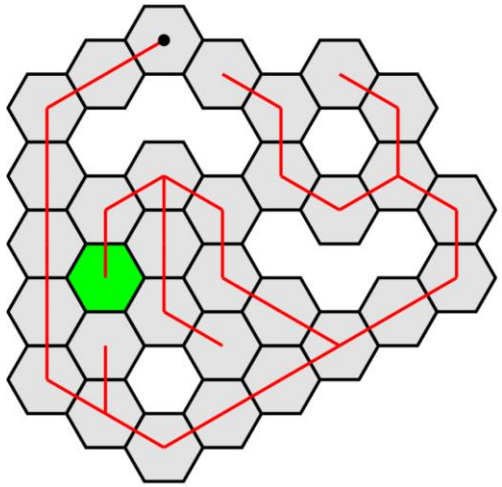
Long-Range Coordination: Reconfigurable Circuits

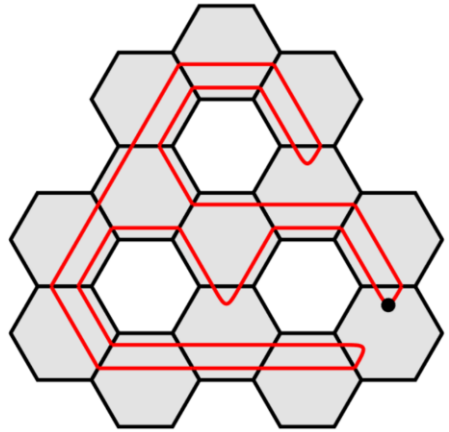
Reconfigurable circuits also admit **polylogarithmic** algorithms for computing various structural components, like extrema, boundaries, and spanning trees [PSW 2022]:

Problem	Required pins	Runtime
Stripe	2	$O(\log n)$
Global maxima	2	$O(\log^2 n)$ w.h.p.
Canonical skeleton	4	$O(\log^2 n)$ w.h.p.
Spanning tree	4	$O(\log^2 n)$ w.h.p.
Symmetry detection	4	$O(\log^5 n)$ w.h.p.



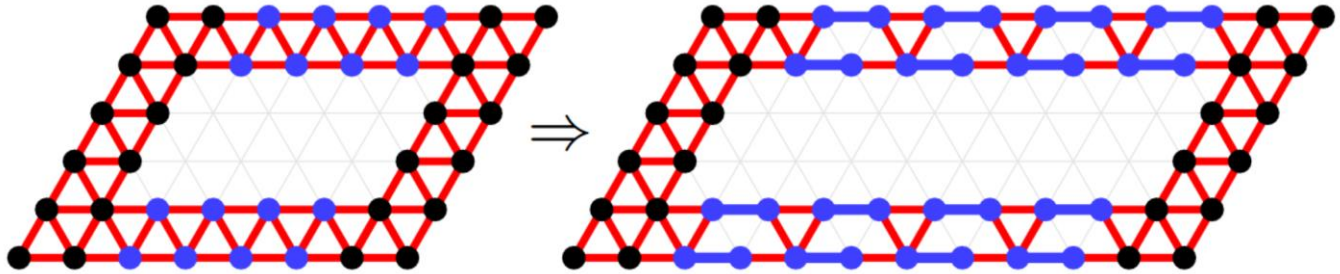




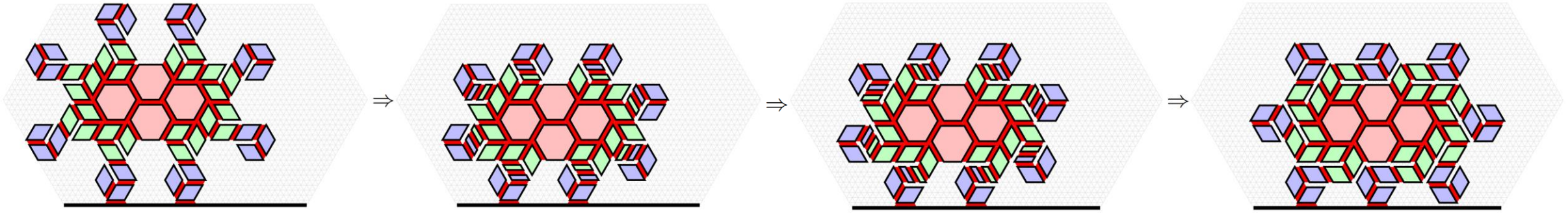


Long-Range Coordination: Joint Movements

The related **joint movement** extension allows sets of amoebots to simultaneously expand or contract (similar to the nubot model), moving the rest of the system accordingly [PKS 2023]:



Amoebots can be organized into rhombic and hexagonal "meta-modules" with higher-level movement primitives that can then simulate modular robot shape formation and locomotion:



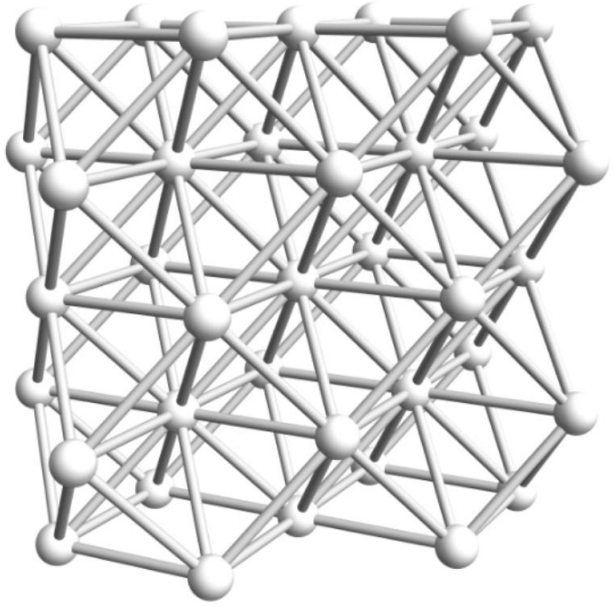
Extending to 3D (Geometric) Space

Almost all theoretical research for programmable matter takes place in some planar graph structure (e.g., the geometric amoebot model's triangular lattice), but modular robotics and programmable matter practitioners work in 3D.

"Catoms"
PB 2018

The **3D geometric space variant** uses the face-centered cubic lattice:

- Equivalent to the "cannonball packing".
- Spherical or rhombic-dodecahedral modules.
- Each amoebot has 12 neighbors when contracted and up to 18 when expanded.

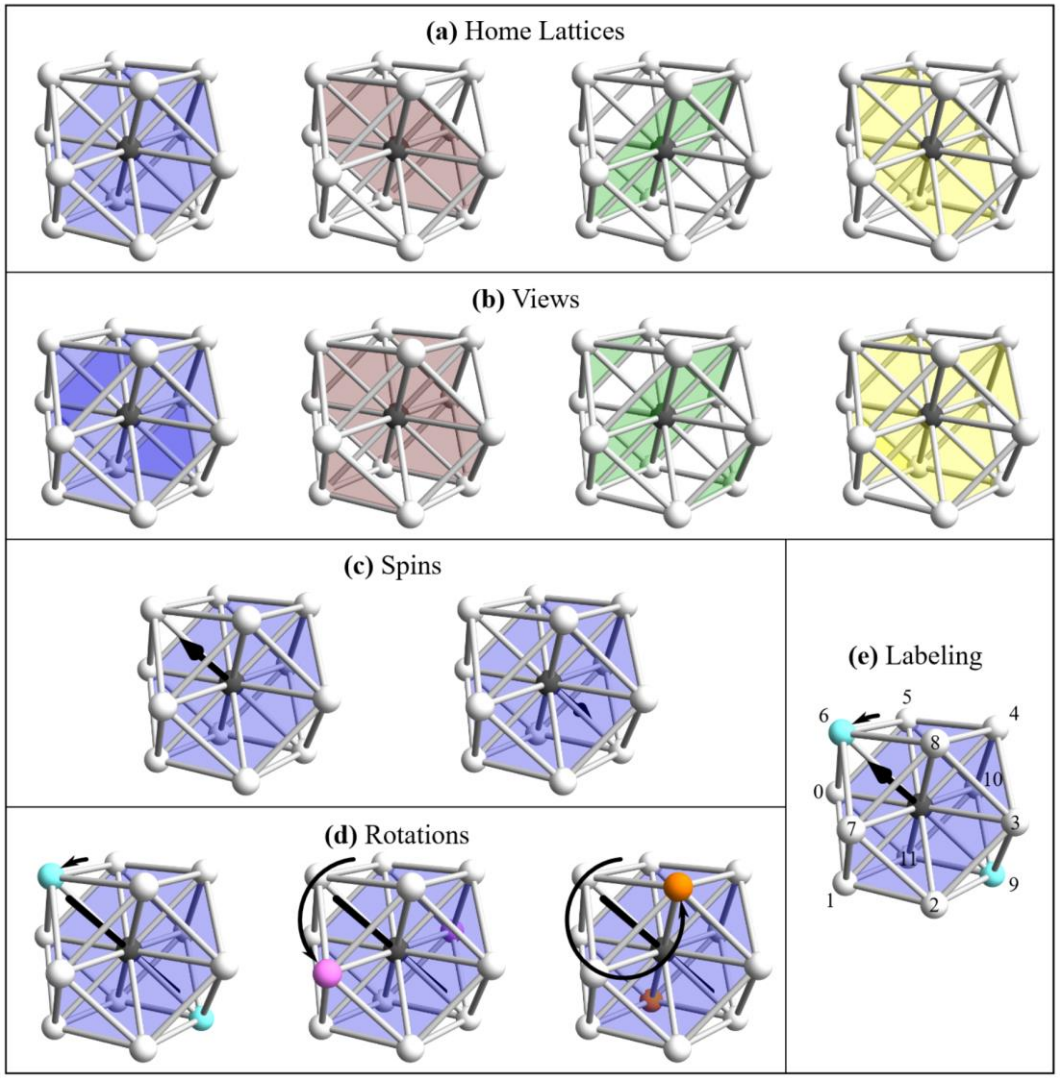


Extending to 3D (Geometric) Space

The complexity in 3D is a combination of amoebots' orientations and the topological analysis required to understand system shapes.

Only leader election has been studied in this 3D variant of the amoebot model [GAMT 2022, BCDR 2023]; 3D coating has recently been discussed in the hybrid model [KLS 2023] and shape formation has been studied for 3D catoms [TPB 2021].

Eventually, it will be interesting to study these problems under physical constraints relevant for practical systems, like **gravity stability** and **strain**.



Closing Thoughts and Future Directions

- A request for (some) unity: Please use the canonical amoebot model's assumption variants!
- A careful revisiting of pre-canonical works may reveal yet unresolved questions, as the comparison of leader election results did.
- Fault tolerance, self-stabilization, reconfigurable circuits, joint movements, and 3D space are developing areas—plenty of room for new ideas.

A Brief Memory of SIROCCO 2015

Thank you!

Email: jdaymude@asu.edu

Website: jdaymude.github.io

@joshdaymude

fediscience.org/@joshdaymude