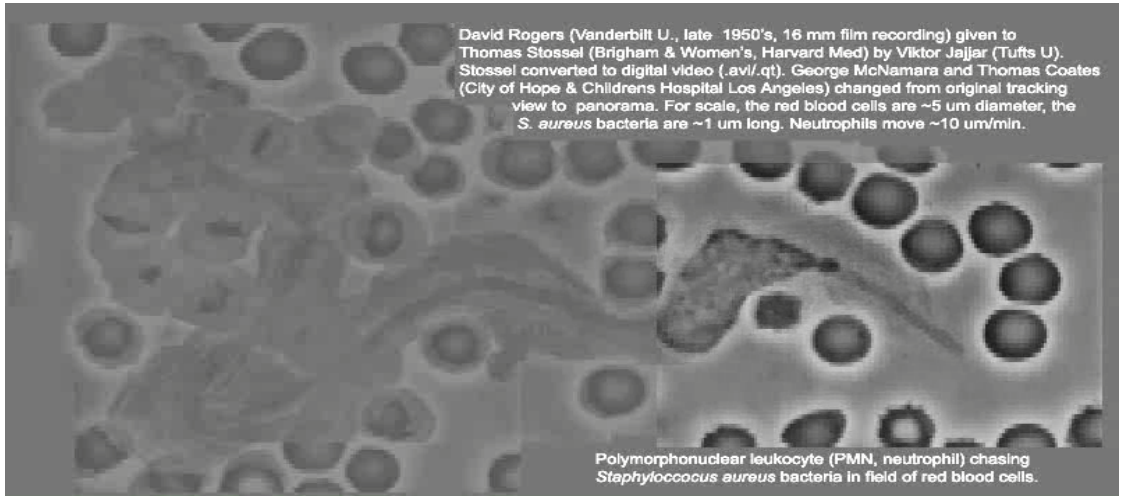
Agents and reagents: Distributed computing in a test tube

David Doty, David Soloveichik

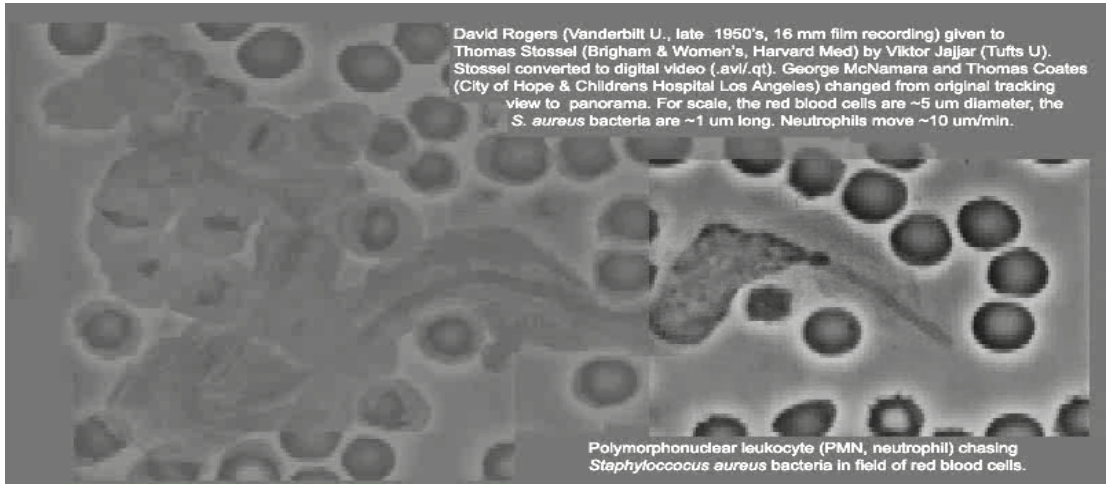
DISC 2014 Tutorial

The software of life

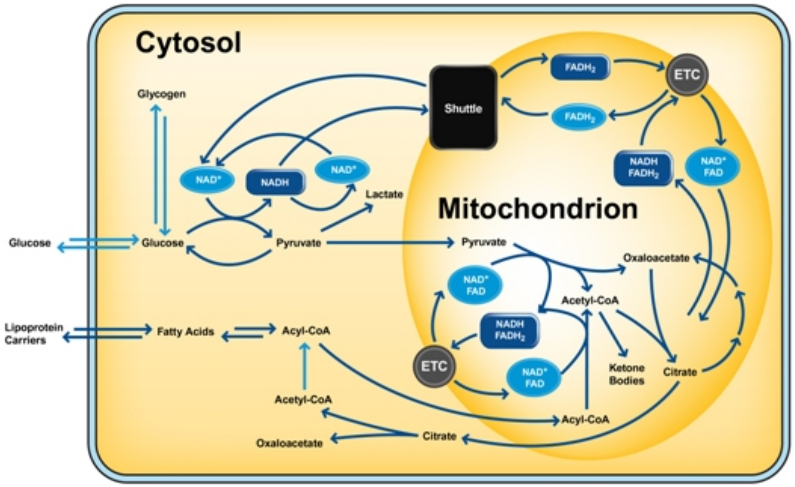


How does the cell compute?

The software of life

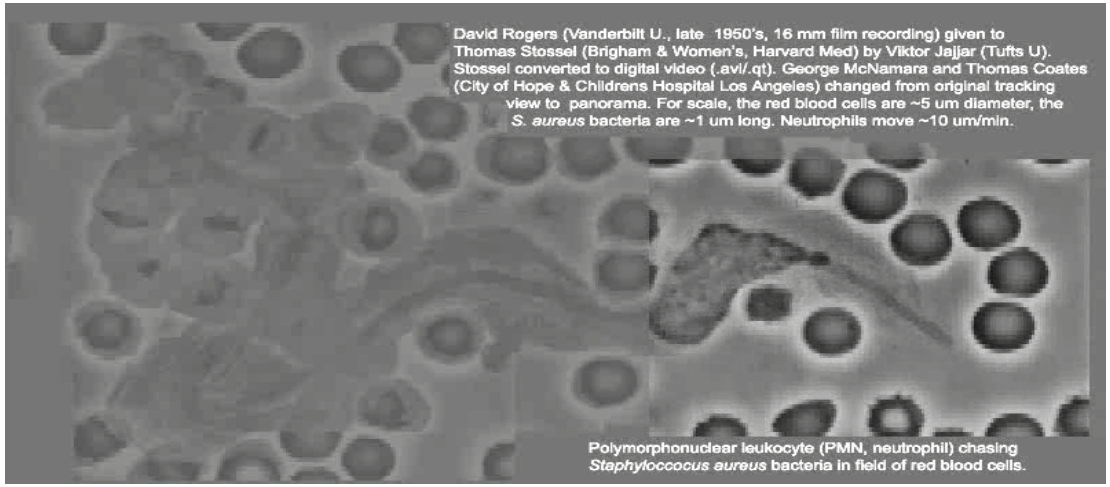


How does the cell compute?



chemistry / geometry

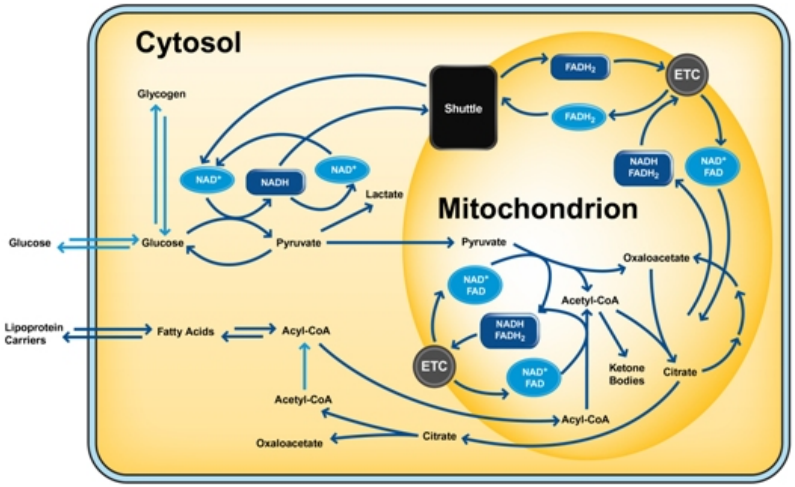
The software of life



~~How does the cell compute?~~

What is possible to compute with chemistry?

~~geometry~~



Chemical reaction networks (CRN)

Chemical reaction networks (CRN)

$$R \rightarrow P_1 + P_2$$

Chemical reaction networks (CRN)

$$R \rightarrow P_1 + P_2$$

$$A + B \rightarrow C$$

Chemical reaction networks (CRN)

$$R \rightarrow P_1 + P_2$$

$$A + B \rightarrow C$$

$$X + Y \rightarrow X + Z$$

Chemical reaction networks (CRN)

$$R \rightarrow P_1 + P_2$$

$$A + B \rightarrow C$$

$$X + Y \rightarrow X + Z$$

$$A + Z \rightarrow$$

(anonymous
waste product)

Chemical reaction networks (CRN)

$$R \rightarrow P_1 + P_2$$

$$A + B \rightarrow C$$

$$X + Y \rightarrow X + Z$$

$$A + Z \rightarrow$$

(anonymous waste product)

$$X \rightarrow 2X$$

(anonymous fuel source)

Chemical reaction networks (CRN)

$$R \xrightarrow{2.5} P_1 + P_2$$

$$A + B \xrightarrow{1} C$$

$$X + Y \xrightarrow{5} X + Z$$

$$A + Z \xrightarrow{0.1}$$

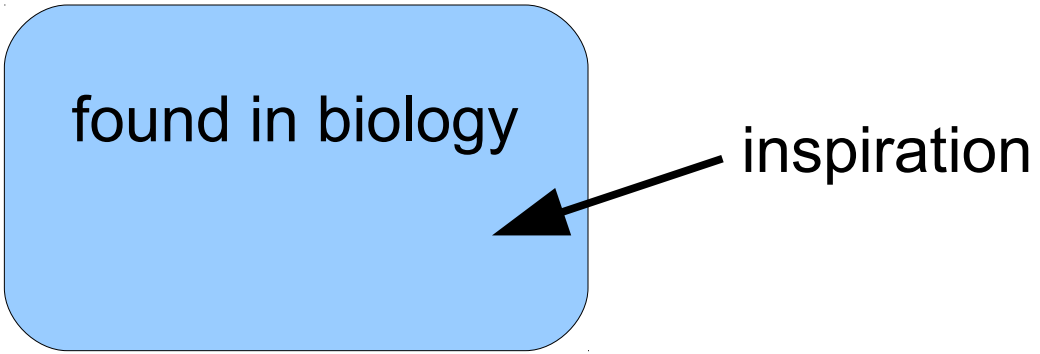
(anonymous waste product)

$$X \xrightarrow{0.1} 2X$$

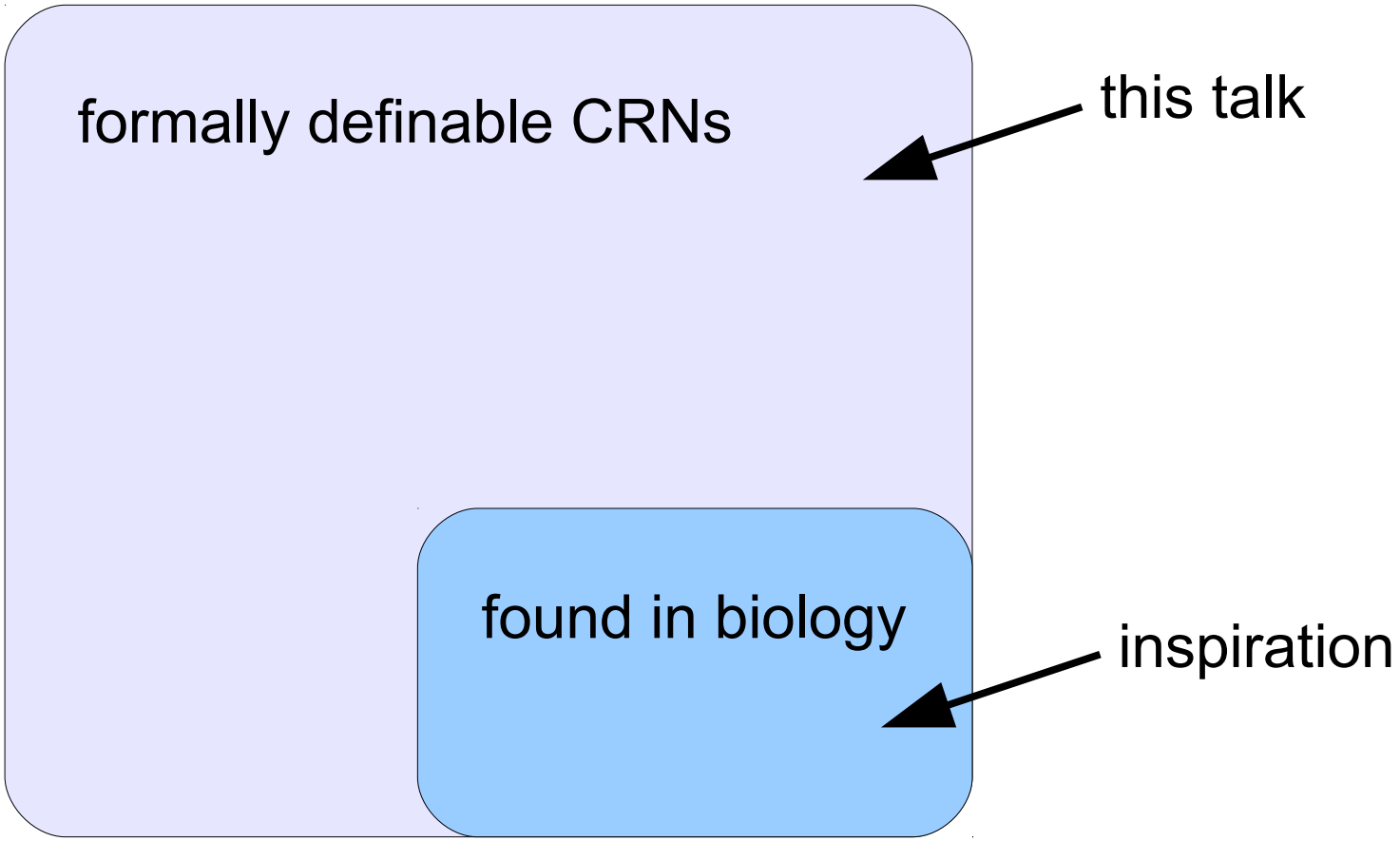
(anonymous fuel source)

What behavior is possible for chemistry in principle?

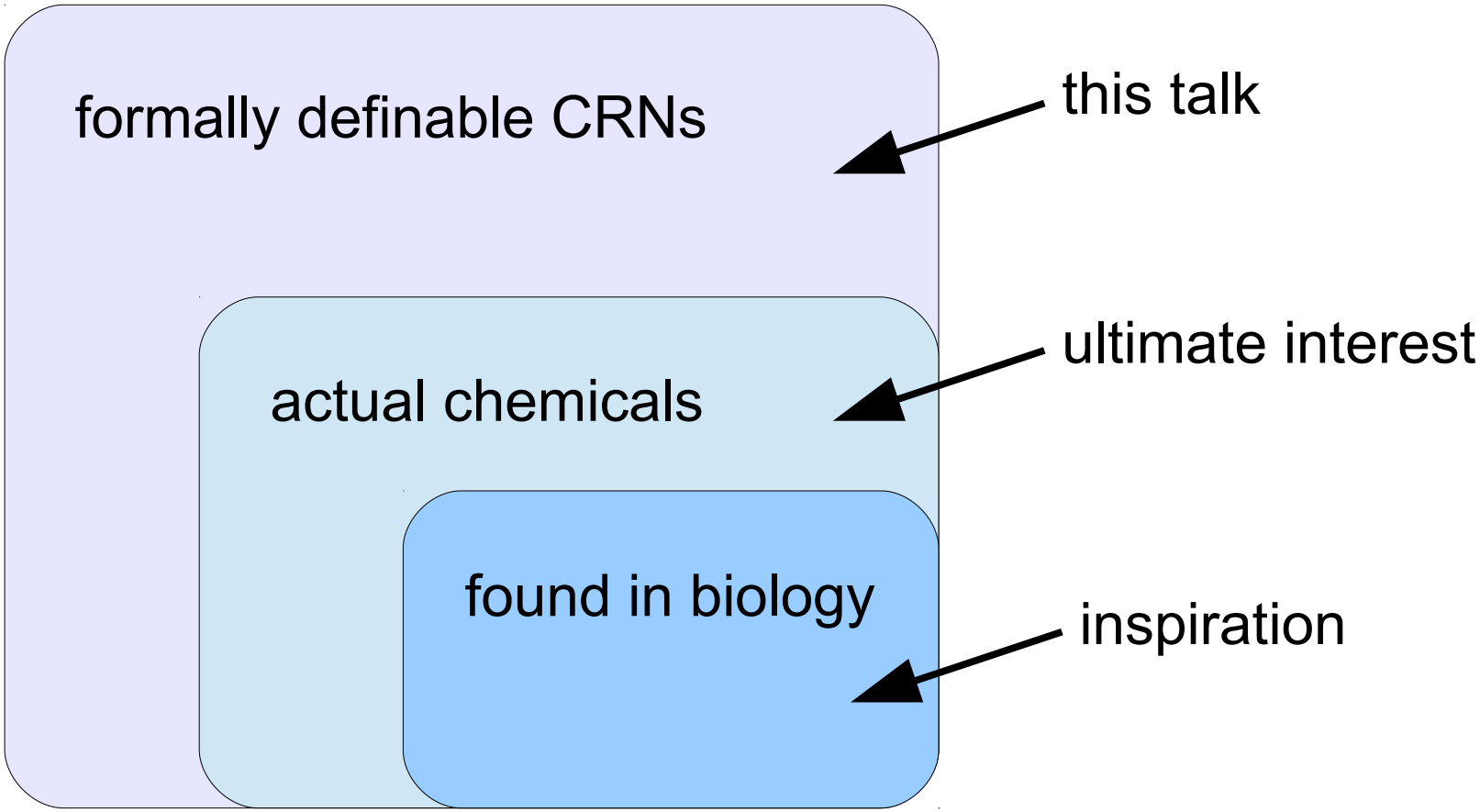
What behavior is possible for chemistry in principle?



What behavior is possible for chemistry in principle?



What behavior is possible for chemistry in principle?



Can we compute with chemistry?

"Not every crazy CRN you scribble on paper describes actual chemicals!"

Can we compute with chemistry?

"Not every crazy CRN you scribble on paper describes actual chemicals!"

Response to objection: Soloveichik et al. [*PNAS* 2010] showed a physical implementation of every CRN, using *DNA strand displacement*

Can we compute with chemistry?

"Not every crazy CRN you scribble on paper describes actual chemicals!"

Response to objection: Soloveichik et al. [*PNAS* 2010] showed a physical implementation of every CRN, using *DNA strand displacement*

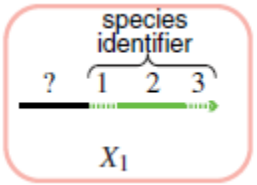
$$X_1 + X_2 \rightarrow X_3$$

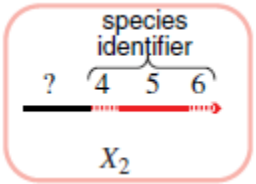
Can we compute with chemistry?

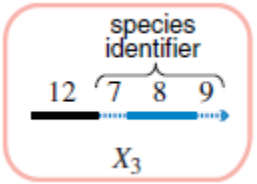
"Not every crazy CRN you scribble on paper describes actual chemicals!"

Response to objection: Soloveichik et al. [*PNAS* 2010] showed a physical implementation of every CRN, using *DNA strand displacement*

$$X_1 + X_2 \rightarrow X_3$$





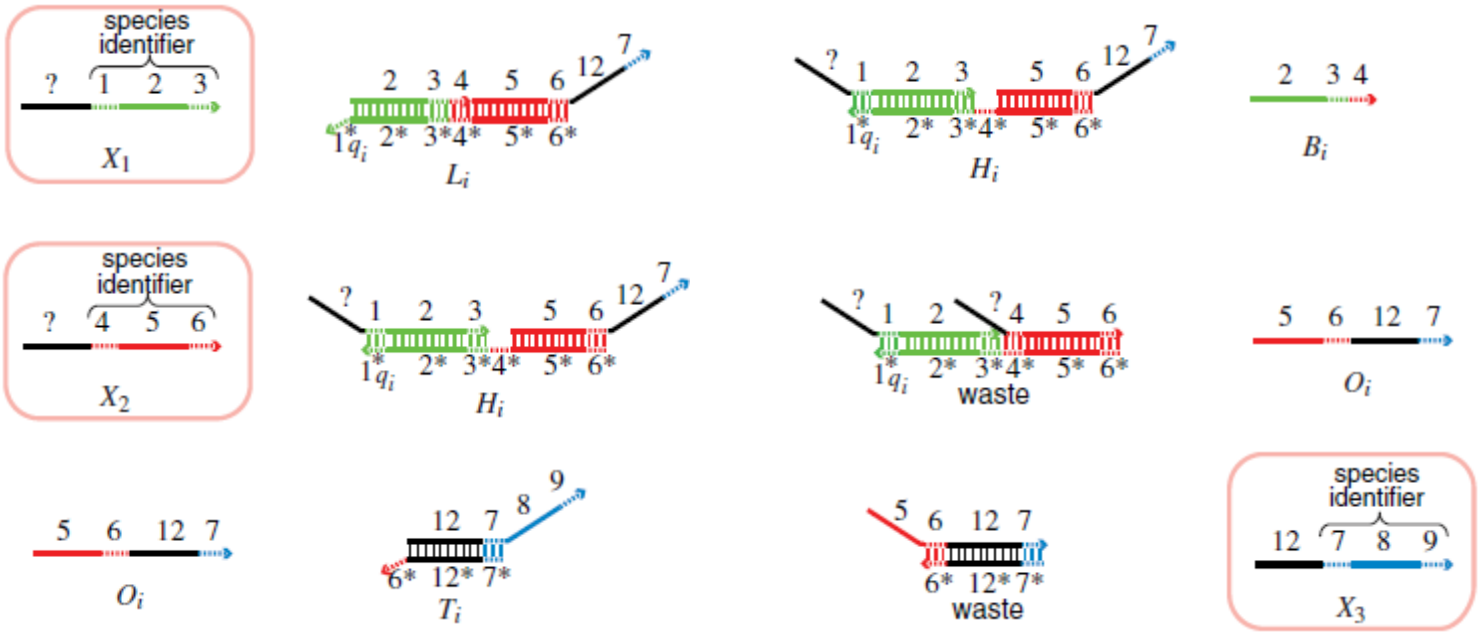


Can we compute with chemistry?

"Not every crazy CRN you scribble on paper describes actual chemicals!"

Response to objection: Soloveichik et al. [*PNAS* 2010] showed a physical implementation of <u>every</u> CRN, using *DNA strand displacement*

$$X_1 + X_2 \rightarrow X_3$$

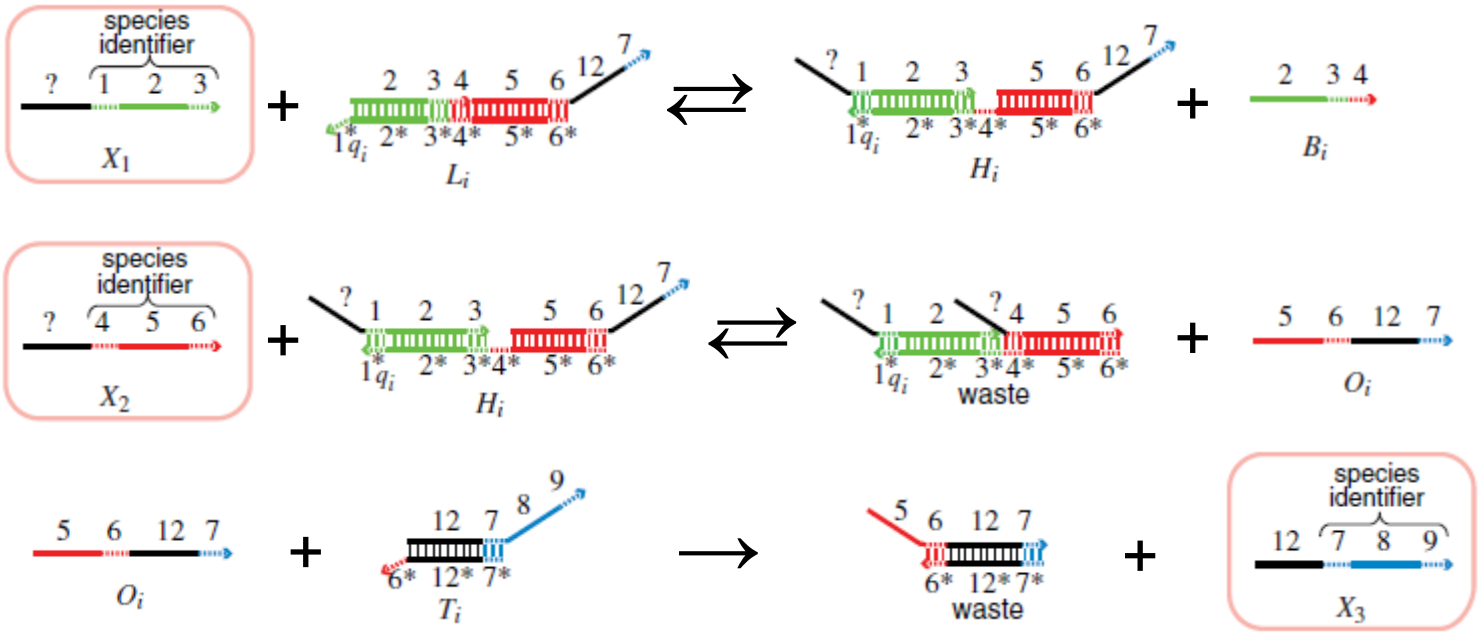


Can we compute with chemistry?

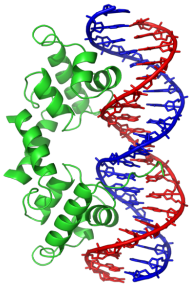
"Not every crazy CRN you scribble on paper describes actual chemicals!"

Response to objection: Soloveichik et al. [*PNAS* 2010] showed a physical implementation of every CRN, using *DNA strand displacement*

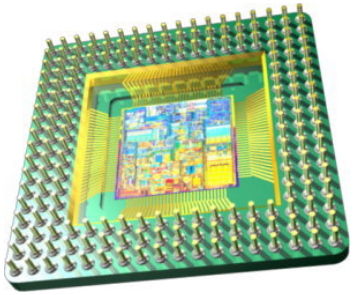
$$X_1 + X_2 \rightarrow X_3$$



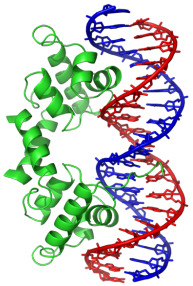
Why compute with chemistry?



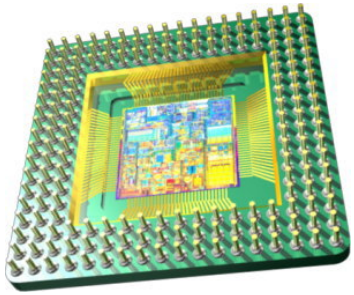
versus



Why compute with chemistry?

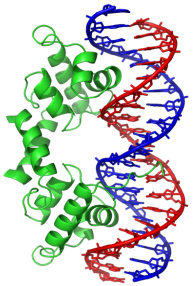


versus

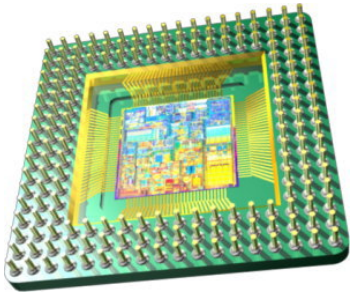


speed?

Why compute with chemistry?

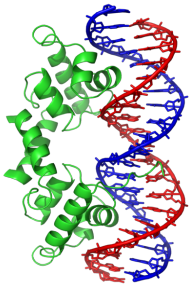


versus

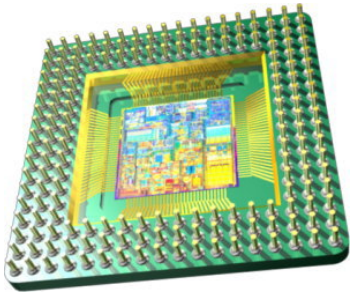


slower speed? faster

Why compute with chemistry?



versus

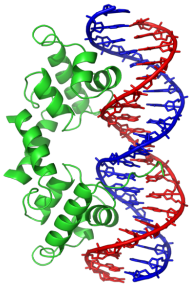


slower

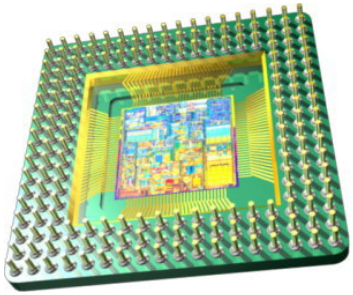
sped?

faster

Why compute with chemistry?



versus



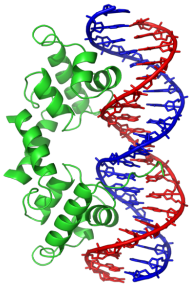
slower

speed?

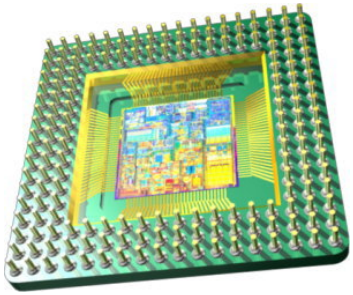
faster

component size?

Why compute with chemistry?



versus



slower

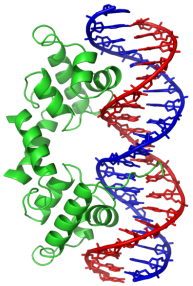
speed?

faster

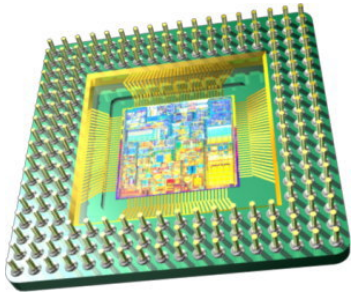
≈ 10-100 nm

component size?

Why compute with chemistry?



versus



slower

speed?

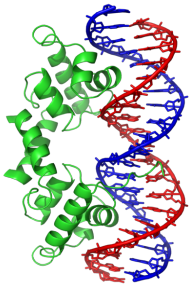
faster

≈ 10-100 nm

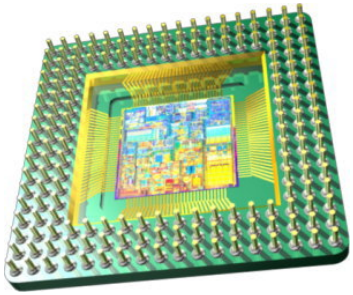
component size?

≈ 10-100 nm

Why compute with chemistry?



versus



slower

≈ 10-100 nm

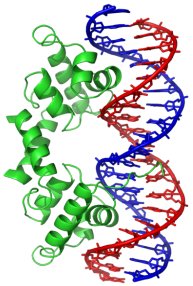
speed?

component size?

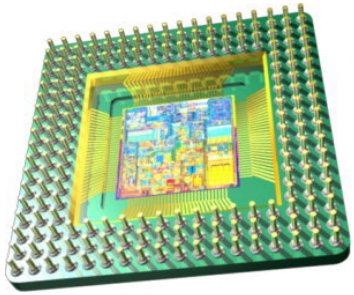
faster

≈ 10-100 nm

Why compute with chemistry?

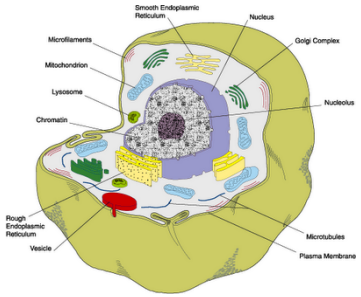


versus



slower	sped?	faster
≈ 10-100 nm	component size?	≈ 10-100 nm
yes	✓ Compatible with biological or other "wet environments"?	not easily

cells



"smart drug" to detect microRNAs levels of cell and release appropriate drug in response

bioreactors

"chemical controller" to increase yield of metabolically produced biofuels/drugs/etc.

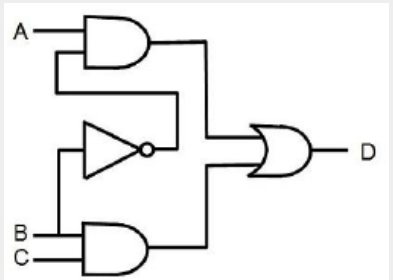
What does it mean to compute with chemistry?

CRNs have a wide range of behaviors:

What does it mean to compute with chemistry?

CRNs have a wide range of behaviors:

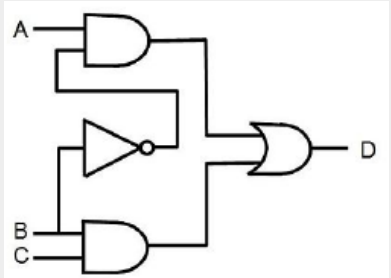
Boolean logic



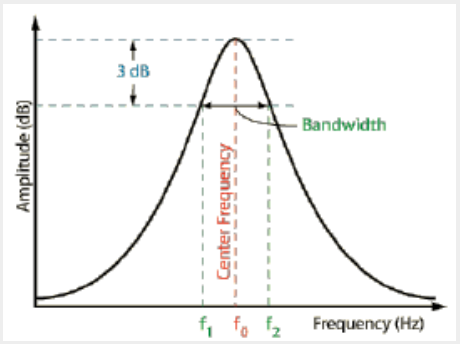
What does it mean to compute with chemistry?

CRNs have a wide range of behaviors:

Boolean logic



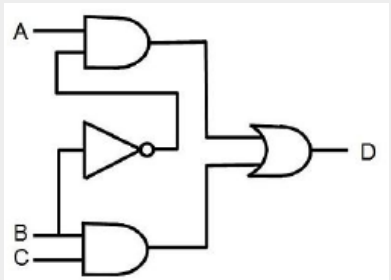
signal processing



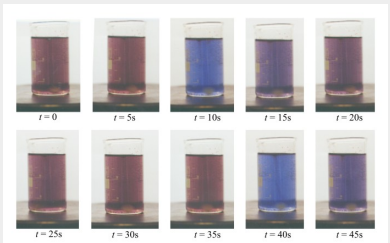
What does it mean to compute with chemistry?

CRNs have a wide range of behaviors:

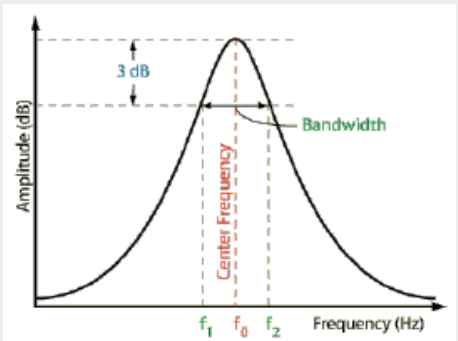
Boolean logic



oscillation



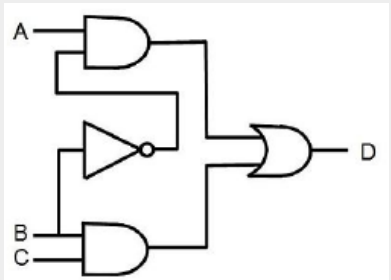
signal processing



What does it mean to compute with chemistry?

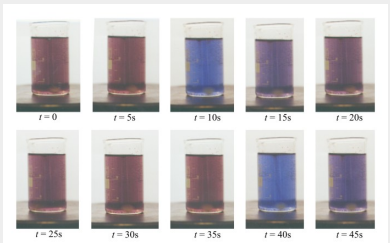
CRNs have a wide range of behaviors:

Boolean logic

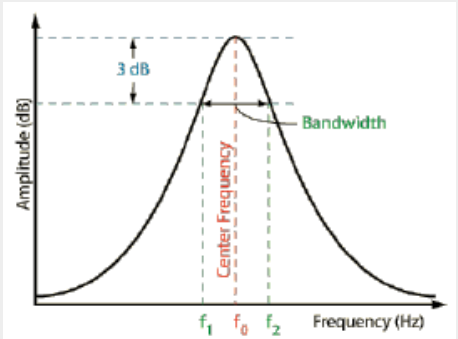


analog computing

oscillation



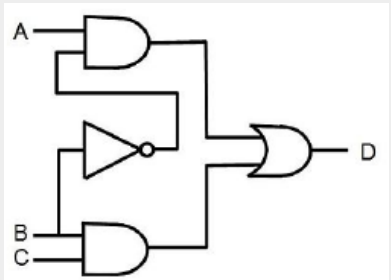
signal processing



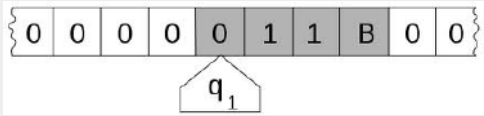
What does it mean to compute with chemistry?

CRNs have a wide range of behaviors:

Boolean logic

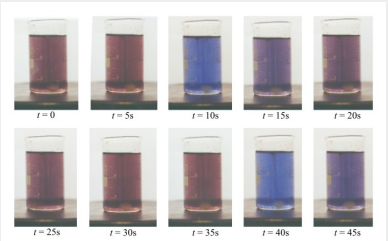


discrete algorithms

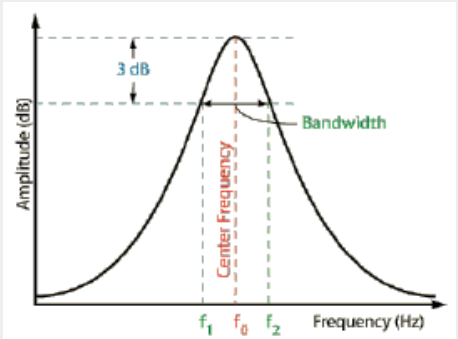


analog computing

oscillation



signal processing



Discrete (stochastic) kinetic CRN model

- **species**: {*X*, *Y*, …}

Discrete (stochastic) kinetic CRN model

- **species**: {X, Y, …}

- **reactions**:

$$X \xrightarrow{k_1} W + 2Y + Z$$

$$A + B \xrightarrow{k_2} X$$

Discrete (stochastic) kinetic CRN model

- **species**: {*X*, *Y*, …}
- **state**: integer vector of *counts*
  s = (#*X*, #*Y*, ...)
- **reactions**:

$$X \xrightarrow{k_1} W + 2Y + Z$$

$$A + B \xrightarrow{k_2} X$$

Discrete (stochastic) kinetic CRN model

- **species**: {*X*, *Y*, …}
- **state**: integer vector of *counts*
  s = (#*X*, #*Y*, ...)
- **reactions**:
- **rate** of reaction:

$X \xrightarrow{k_1} W + 2Y + Z$	$k_1 \bullet \#X$
$A + B \xrightarrow{k_2} X$	$k_2 \bullet \#A \bullet \#B$ / volume

Discrete (stochastic) kinetic CRN model

- **species**: {*X*, *Y*, …}
- **state**: integer vector of *counts*
  s = (#*X*, #*Y*, ...)
- **reactions**:
- **rate** of reaction:

$X \xrightarrow{k_1} W + 2Y + Z$	$k_1 \bullet \#X$
$A + B \xrightarrow{k_2} X$	$k_2 \bullet \#A \bullet \#B$ / volume

$$\text{Prob[some reaction]} = \frac{\text{rate of that reaction}}{\text{sum of all reaction rates}}$$

Discrete (stochastic) kinetic CRN model

- **species**: {*X*, *Y*, …}
- **state**: integer vector of *counts*
  s = (#*X*, #*Y*, ...)
- **reactions**:
- **rate** of reaction:

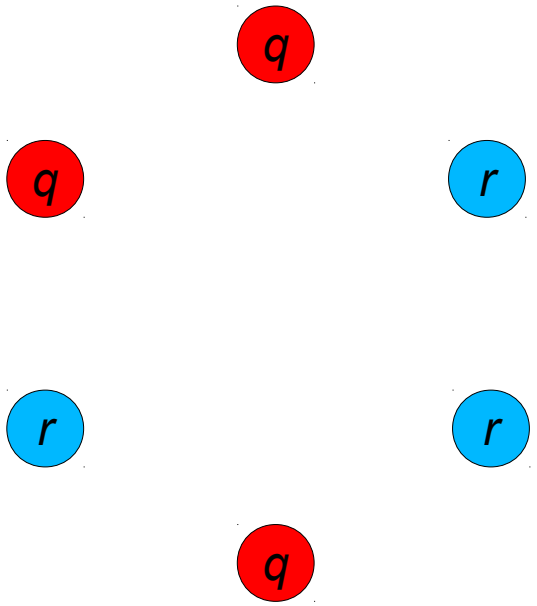
$X \xrightarrow{k_1} W + 2Y + Z$	$k_1 \bullet \#X$
$A + B \xrightarrow{k_2} X$	$k_2 \bullet \#A \bullet \#B$ / volume

$$\text{Prob[some reaction]} = \frac{\text{rate of that reaction}}{\text{sum of all reaction rates}}$$

time until next reaction = exponential random variable

Population protocols

n finite-state agents

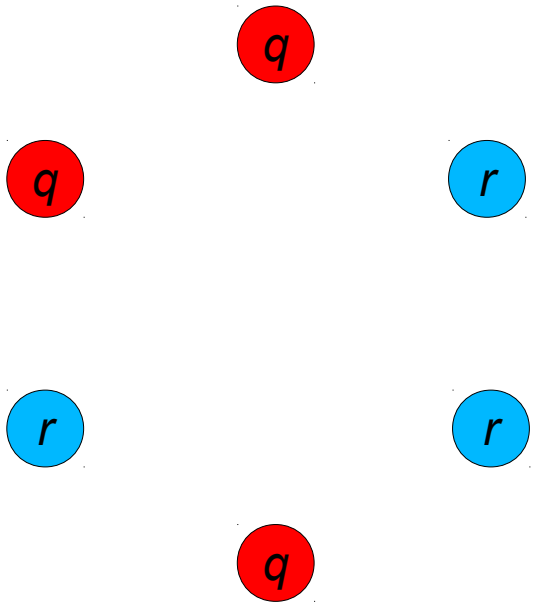


(Angluin, Aspnes, Diamadi, Fisher, Peralta, PODC 2004)

Population protocols

n finite-state agents

repeatedly pick pair to interact



$\delta(q,r) = (s,t)$

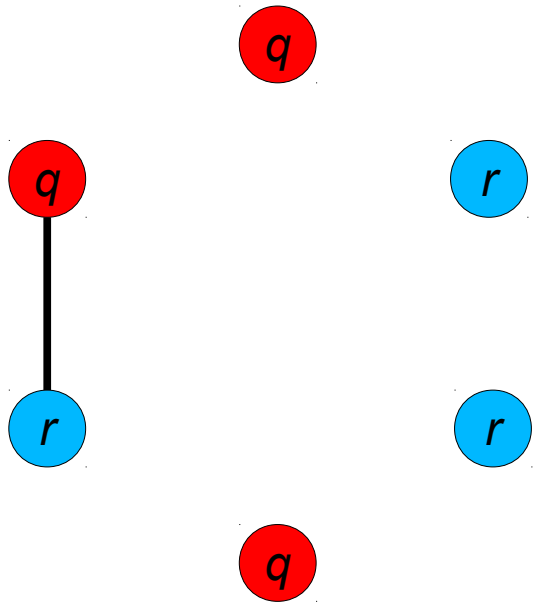
$\delta(s,r) = (q,q)$

(Angluin, Aspnes, Diamadi, Fisher, Peralta, PODC 2004)

Population protocols

n finite-state agents

repeatedly pick pair to interact



$\delta(q,r) = (s,t)$

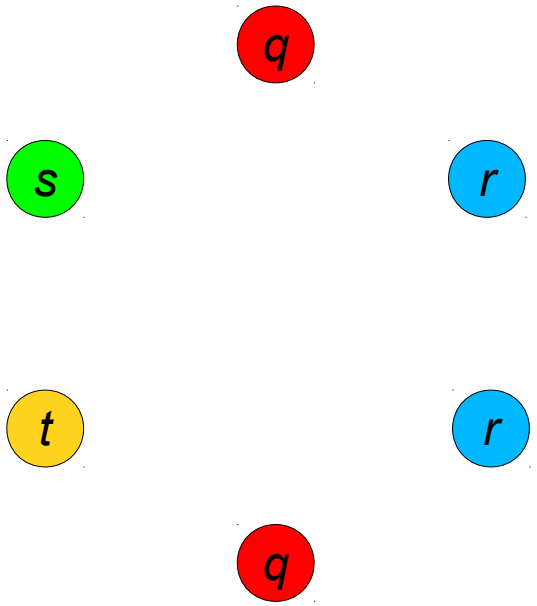
$\delta(s,r) = (q,q)$

(Angluin, Aspnes, Diamadi, Fisher, Peralta, PODC 2004)

Population protocols

n finite-state agents

repeatedly pick pair
to interact



$\delta(q,r) = (s,t)$

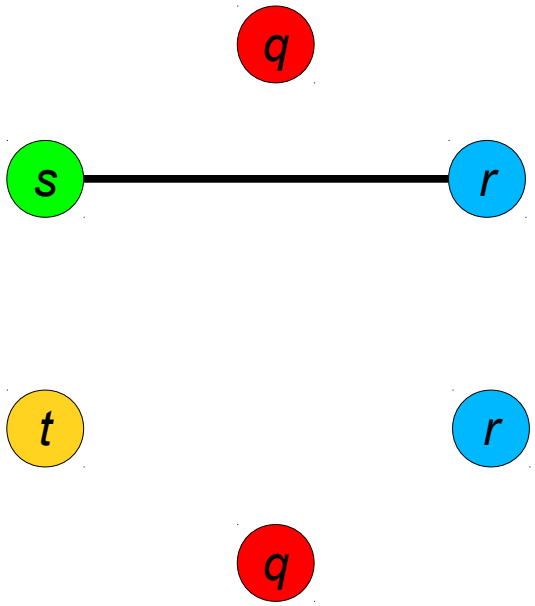
$\delta(s,r) = (q,q)$

(Angluin, Aspnes, Diamadi, Fisher, Peralta, PODC 2004)

Population protocols

n finite-state agents

repeatedly pick pair to interact



$\delta(q,r) = (s,t)$

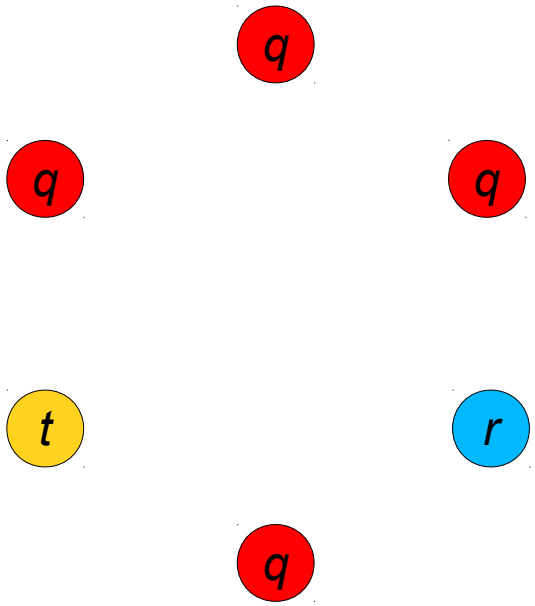
$\delta(s,r) = (q,q)$

(Angluin, Aspnes, Diamadi, Fisher, Peralta, PODC 2004)

Population protocols

n finite-state agents

repeatedly pick pair
to interact



$\delta(q,r) = (s,t)$

$\delta(s,r) = (q,q)$

(Angluin, Aspnes, Diamadi, Fisher, Peralta, PODC 2004)

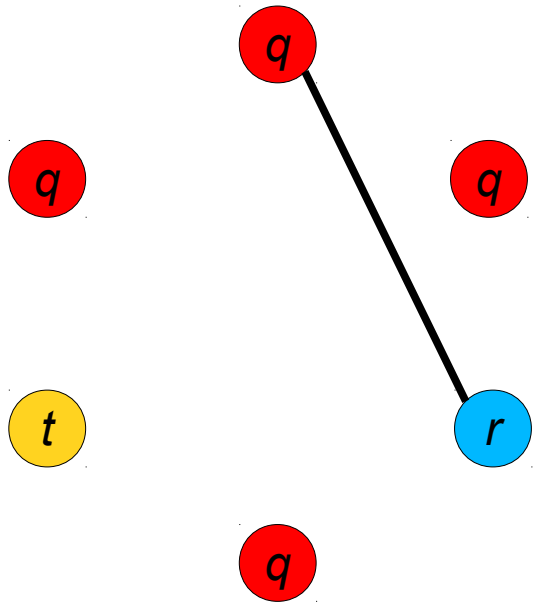
Population protocols

n finite-state agents

repeatedly pick pair to interact

$\delta(q,r) = (s,t)$

$\delta(s,r) = (q,q)$

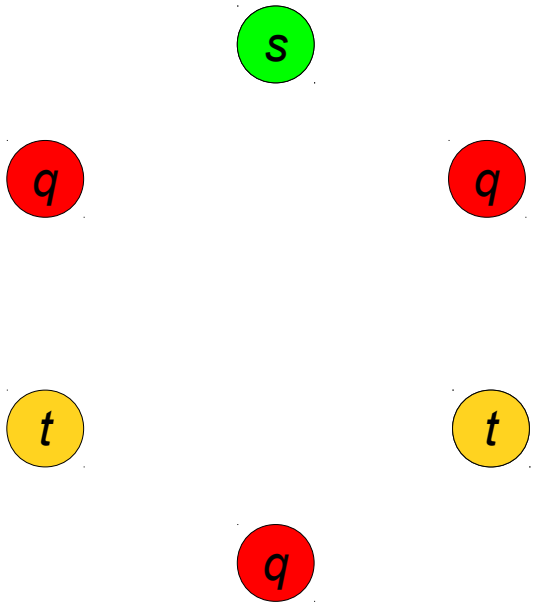


(Angluin, Aspnes, Diamadi, Fisher, Peralta, PODC 2004)

Population protocols

n finite-state agents

repeatedly pick pair to interact



$\delta(q,r) = (s,t)$

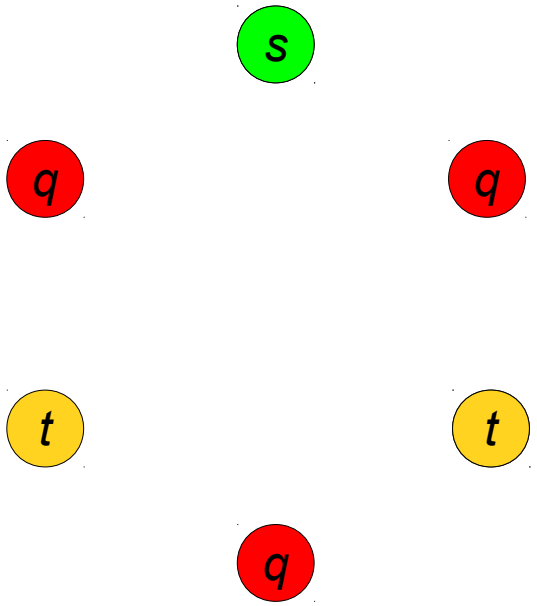
$\delta(s,r) = (q,q)$

(Angluin, Aspnes, Diamadi, Fisher, Peralta, PODC 2004)

Population protocols

n finite-state agents

repeatedly pick pair to interact



$\delta(q,r) = (s,t)$

$\delta(s,r) = (q,q)$

"parallel time" = # of interactions / *n*

(Angluin, Aspnes, Diamadi, Fisher, Peralta, PODC 2004)

A PP is a CRN such that...

A PP is a CRN such that...

- all reactions have 2 reactants and 2 products

A PP is a CRN such that...

- all reactions have 2 reactants and 2 products
- all rate constants are 1

A PP is a CRN such that...

- all reactions have 2 reactants and 2 products
- all rate constants are 1
- volume = number of molecules
  (constant over time because of first constraint)

A PP is a CRN such that...

- all reactions have 2 reactants and 2 products
- all rate constants are 1
- volume = number of molecules
  (constant over time because of first constraint)
- order of reactants can matter
  (there's a “sender” and a “receiver” molecule)

A PP is a CRN such that...

- all reactions have 2 reactants and 2 products
- all rate constants are 1
- volume = number of molecules
  (constant over time because of first constraint)
- order of reactants can matter
  (there's a “sender” and a “receiver” molecule)
- sender/receiver states uniquely determine products
  (e.g., cannot have A+B → C+D and A+B → X+Y)

Computation with CRNs: Outline

- Stable computation (“deterministic”)
- Randomized computation:
  - probability of error = small
  - probability of error = 0

Stable (deterministic) CRN computation

Stable CRN predicate computation (definition)

task: compute predicate $p(x_1,...,x_k)$, $x_1,...,x_k \in \mathbb{N}$

Stable CRN predicate computation (definition)

task: compute predicate $p(x_1,...,x_k)$, $x_1,...,x_k \in \mathbb{N}$

votes: two disjoint subsets of species: “yes” and “no” voters

Stable CRN predicate computation (definition)

task: compute predicate $p(x_1,...,x_k)$, $x_1,...,x_k \in \mathbb{N}$

votes: two disjoint subsets of species: "yes" and "no" voters

output $\varphi(s)$ of state s: the consensus vote (if voters unanimous)

Stable CRN predicate computation (definition)

task: compute predicate $p(x_1,...,x_k)$, $x_1,...,x_k \in \mathbb{N}$

votes: two disjoint subsets of species: "yes" and "no" voters

output $\varphi(s)$ of state s: the consensus vote (if voters unanimous)

initial state: $\#X_1 = x_1,\ldots,\#X_k = x_k$, constant counts of other species

Stable CRN predicate computation (definition)

task: compute predicate $p(x_1,...,x_k)$, $x_1,...,x_k \in \mathbb{N}$

votes: two disjoint subsets of species: "yes" and "no" voters

output $\varphi(s)$ of state s: the consensus vote (if voters unanimous)

initial state: $\#X_1 = x_1,\ldots,\#X_k = x_k$, constant counts of other species

output-stable state: all states reachable from it have same output

Stable CRN predicate computation (definition)

task: compute predicate $p(x_1,...,x_k)$, $x_1,...,x_k \in \mathbb{N}$

votes: two disjoint subsets of species: "yes" and "no" voters

output $\varphi(\mathbf{s})$ of state s: the consensus vote (if voters unanimous)

initial state: $\#X_1 = x_1,\ldots,\#X_k = x_k$, constant counts of other species

output-stable state: all states reachable from it have same output

stable computation: for all states **s** reachable from the initial state **x**, a correct output-stable state **o** is reachable from **s**

Stable CRN predicate computation (definition)

task: compute predicate $p(x_1,...,x_k)$, $x_1,...,x_k \in \mathbb{N}$

votes: two disjoint subsets of species: "yes" and "no" voters

output $\varphi(\mathbf{s})$ of state s: the consensus vote (if voters unanimous)

initial state: $\#X_1 = x_1,\ldots,\#X_k = x_k$, constant counts of other species

output-stable state: all states reachable from it have same output

stable computation: for all states **s** reachable from the initial state **x**, a correct output-stable state **o** is reachable from **s**

Stable CRN predicate computation (definition)

task: compute predicate $p(x_1,...,x_k)$, $x_1,...,x_k \in \mathbb{N}$

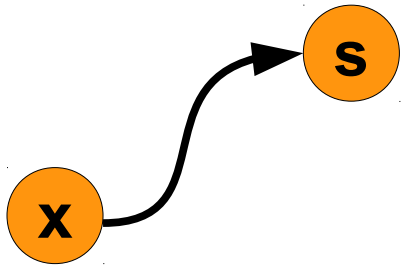
votes: two disjoint subsets of species: "yes" and "no" voters

output $\varphi(\mathbf{s})$ of state s: the consensus vote (if voters unanimous)

initial state: $\#X_1 = x_1,\ldots,\#X_k = x_k$, constant counts of other species

output-stable state: all states reachable from it have same output

stable computation: for all states **s** reachable from the initial state **x**, a correct output-stable state **o** is reachable from **s**



Stable CRN predicate computation (definition)

task: compute predicate $p(x_1,...,x_k)$, $x_1,...,x_k \in \mathbb{N}$

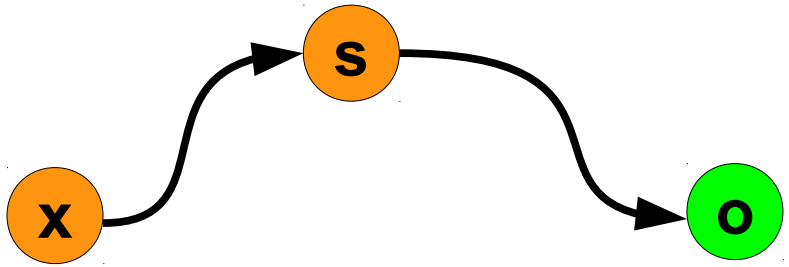
votes: two disjoint subsets of species: "yes" and "no" voters

output φ(s) of state s: the consensus vote (if voters unanimous)

initial state: $\#X_1 = x_1,\ldots,\#X_k = x_k$, constant counts of other species

output-stable state: all states reachable from it have same output

stable computation: for all states **s** reachable from the initial state **x**, a correct output-stable state **o** is reachable from **s**



$\varphi(\boldsymbol{o}) = p(x_1,..,x_k)$

Stable CRN predicate computation (definition)

task: compute predicate $p(x_1,...,x_k)$, $x_1,...,x_k \in \mathbb{N}$

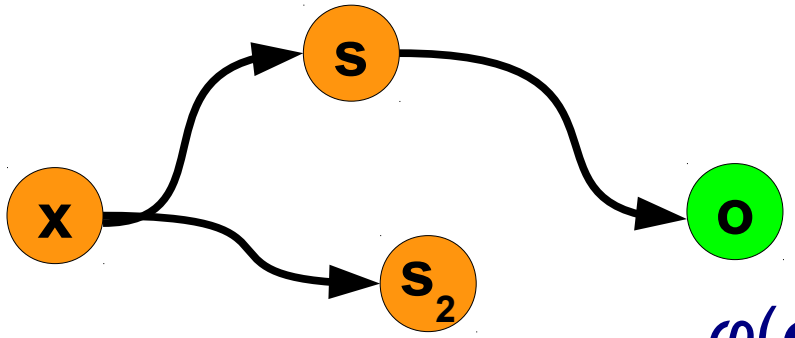
votes: two disjoint subsets of species: "yes" and "no" voters

output φ(s) of state s: the consensus vote (if voters unanimous)

initial state: $\#X_1 = x_1,\ldots,\#X_k = x_k$, constant counts of other species

output-stable state: all states reachable from it have same output

stable computation: for all states **s** reachable from the initial state **x**, a correct output-stable state **o** is reachable from **s**



$\varphi(\boldsymbol{o}) = p(x_1,..,x_k)$

Stable CRN predicate computation (definition)

task: compute predicate $p(x_1,...,x_k)$, $x_1,...,x_k \in \mathbb{N}$

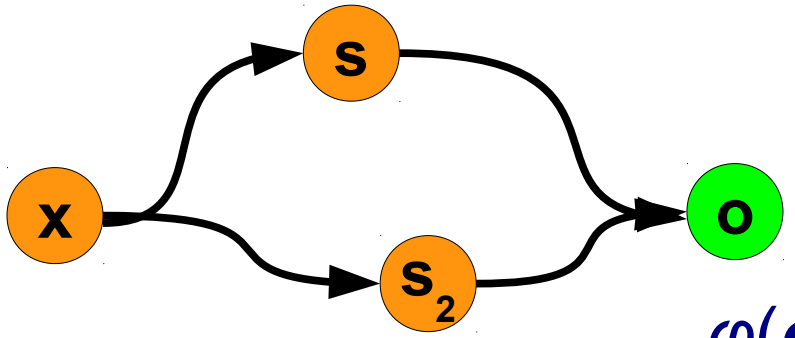
votes: two disjoint subsets of species: "yes" and "no" voters

output $\varphi(\mathbf{s})$ of state s: the consensus vote (if voters unanimous)

initial state: $\#X_1 = x_1,\ldots,\#X_k = x_k$, constant counts of other species

output-stable state: all states reachable from it have same output

stable computation: for all states **s** reachable from the initial state **x**, a correct output-stable state **o** is reachable from **s**



$\varphi(\mathbf{o}) = p(x_1,..,x_k)$

Stable CRN predicate computation (definition)

task: compute predicate $p(x_1,...,x_k)$, $x_1,...,x_k \in \mathbb{N}$

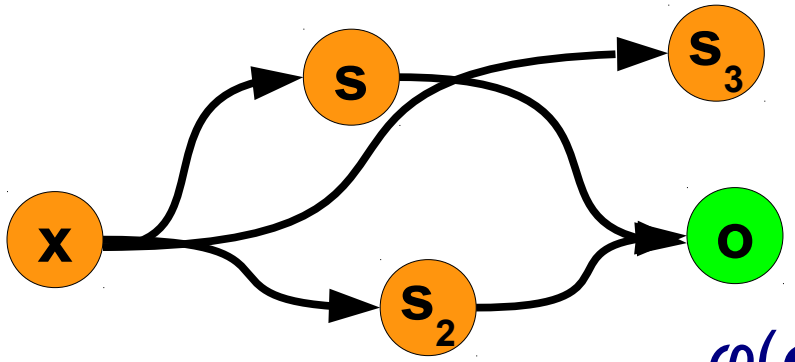
votes: two disjoint subsets of species: "yes" and "no" voters

output φ(s) of state s: the consensus vote (if voters unanimous)

initial state: $\#X_1 = x_1,\ldots,\#X_k = x_k$, constant counts of other species

output-stable state: all states reachable from it have same output

stable computation: for all states **s** reachable from the initial state **x**, a correct output-stable state **o** is reachable from **s**



$\varphi(\boldsymbol{o}) = p(x_1,..,x_k)$

Stable CRN predicate computation (definition)

task: compute predicate $p(x_1,...,x_k)$, $x_1,...,x_k \in \mathbb{N}$

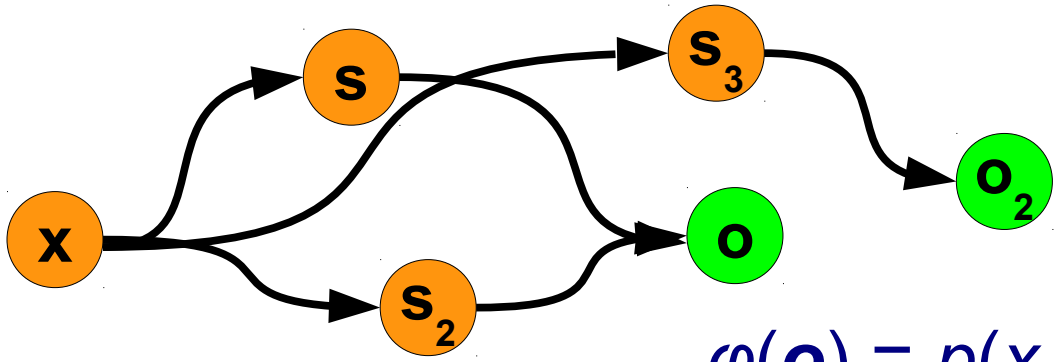
votes: two disjoint subsets of species: "yes" and "no" voters

output $\varphi(\mathbf{s})$ of state s: the consensus vote (if voters unanimous)

initial state: $\#X_1 = x_1,\ldots,\#X_k = x_k$, constant counts of other species

output-stable state: all states reachable from it have same output

stable computation: for all states **s** reachable from the initial state **x**, a correct output-stable state **o** is reachable from **s**



$\varphi(\mathbf{o}) = p(x_1,..,x_k) = \varphi(\mathbf{o}_2)$

Stable CRN predicate computation (alternate definition)

Stable CRN predicate computation (alternate definition)

execution: infinite sequence of states $\mathbf{s}_1, \mathbf{s}_2, \ldots$, where $\mathbf{s}_{i+1}$ is $\mathbf{s}_i$ after applying a reaction (allow "null" reaction for convenience)

Stable CRN predicate computation (alternate definition)

execution: infinite sequence of states $\mathbf{s}_1$, $\mathbf{s}_2$, …, where $\mathbf{s}_{i+1}$ is $\mathbf{s}_i$ after applying a reaction (allow "null" reaction for convenience)

fair execution: every state always **reachable** is infinitely often **reached**

Stable CRN predicate computation (alternate definition)

execution: infinite sequence of states $\mathbf{s}_1$, $\mathbf{s}_2$, …, where $\mathbf{s}_{i+1}$ is $\mathbf{s}_i$ after applying a reaction (allow "null" reaction for convenience)

fair execution: every state always **reachable** is infinitely often **reached**

stable computation: predicate $p(x_1,...,x_k)$ is stably computed if every fair execution contains an output stable state **o** with $\varphi(\mathbf{o}) = p(x_1,...,x_k)$

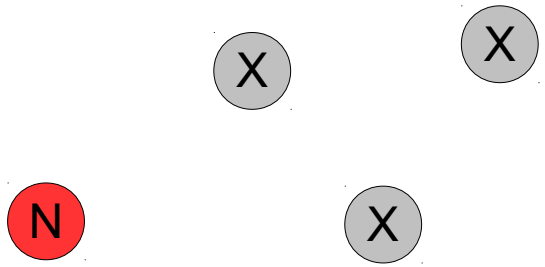
Stable CRN predicate computation (example)

predicate: $p(x)$: parity of x

Stable CRN predicate computation (example)

predicate: $p(x)$: parity of x

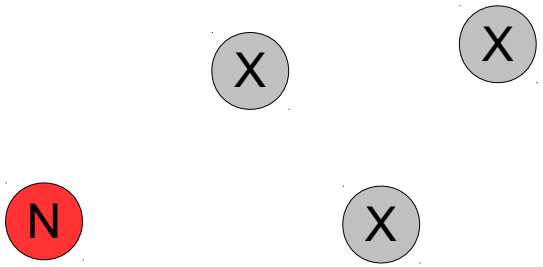
initial state: { x X , 1 N }



Stable CRN predicate computation (example)

predicate: $p(x)$: parity of x

initial state: $\{ x\ X\ ,\ 1\ N \}$



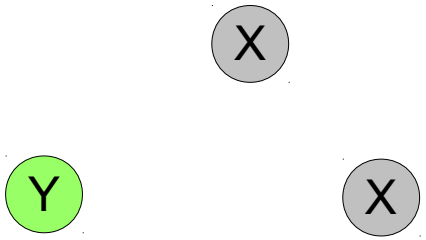
reactions: $N + X \rightarrow Y$

$Y + X \rightarrow N$

Stable CRN predicate computation (example)

predicate: $p(x)$: parity of x

initial state: $\{ x\ X\ ,\ 1\ N \}$



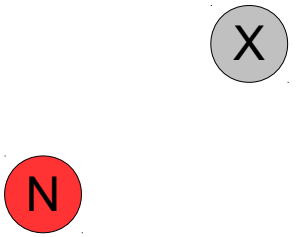
reactions: $N + X \rightarrow Y$

$Y + X \rightarrow N$

Stable CRN predicate computation (example)

predicate: $p(x)$: parity of x

initial state: { x X , 1 N }



reactions: $N + X \rightarrow Y$

$Y + X \rightarrow N$

Stable CRN predicate computation (example)

predicate: $p(x)$: parity of x

initial state: $\{ x\ X\ ,\ 1\ N \}$

reactions: $N + X \rightarrow Y$

$Y + X \rightarrow N$

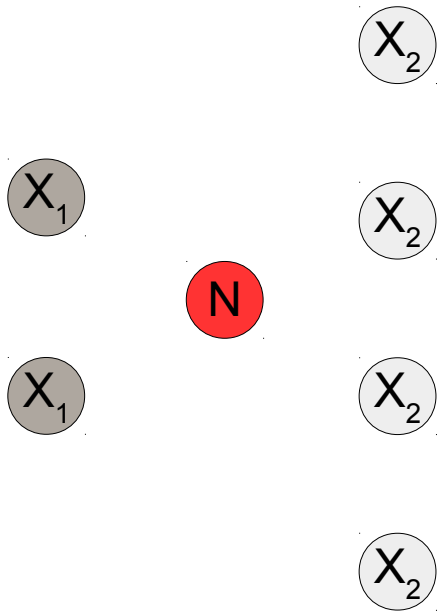
Stable CRN predicate computation (example)

predicate: $p(x_1,x_2)$: "$x_1 > x_2$"?

Stable CRN predicate computation (example)

predicate: $p(x_1,x_2)$: "$x_1 > x_2$"?

initial state: $\{ x_1\, X_1 , x_2\, X_2 , 1\, N \}$

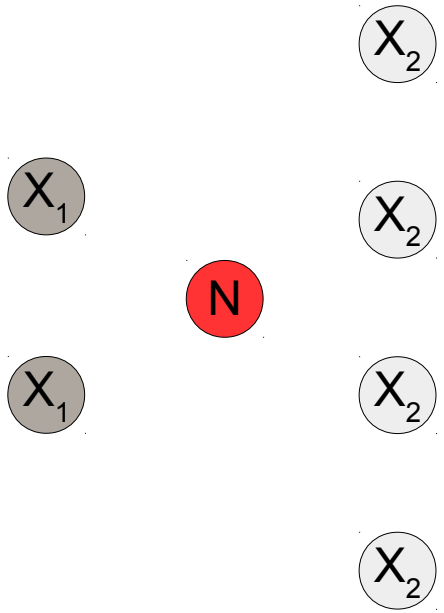


Stable CRN predicate computation (example)

predicate: $p(x_1,x_2)$: "$x_1 > x_2$"?

initial state: $\{ x_1 X_1 , x_2 X_2 , 1 N \}$

reactions:
$$N + X_1 \rightarrow Y$$
$$Y + X_2 \rightarrow N$$

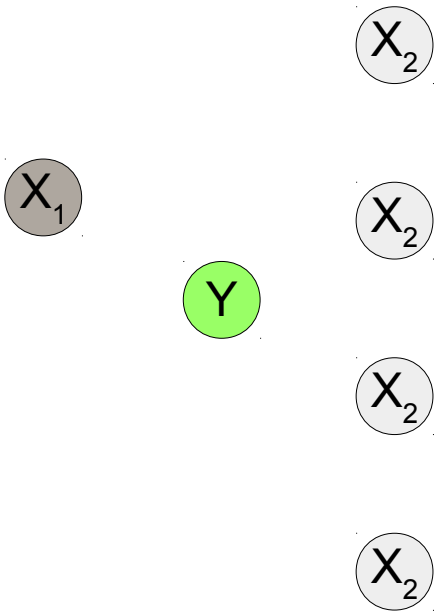


Stable CRN predicate computation (example)

predicate: $p(x_1,x_2)$: "$x_1 > x_2$"?

initial state: $\{ x_1 X_1 , x_2 X_2 , 1 N \}$

reactions:
$$N + X_1 \rightarrow Y$$
$$Y + X_2 \rightarrow N$$



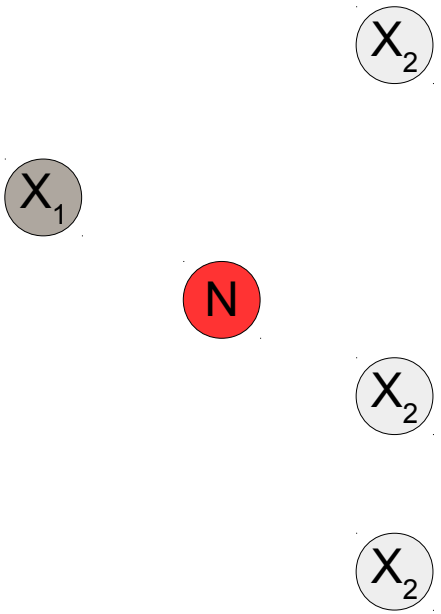
Stable CRN predicate computation (example)

predicate: $p(x_1,x_2)$: "$x_1 > x_2$"?

initial state: $\{ x_1 X_1 , x_2 X_2 , 1 N \}$

reactions: $N + X_1 \rightarrow Y$

$Y + X_2 \rightarrow N$



Stable CRN predicate computation (example)

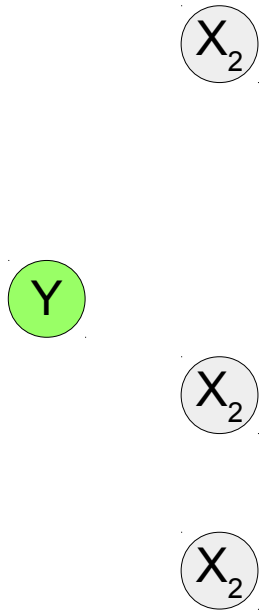
predicate: $p(x_1,x_2)$: "$x_1 > x_2$"?

initial state: $\{ x_1\, X_1 , x_2\, X_2 , 1\, N \}$

reactions:

$$N + X_1 \rightarrow Y$$

$$Y + X_2 \rightarrow N$$



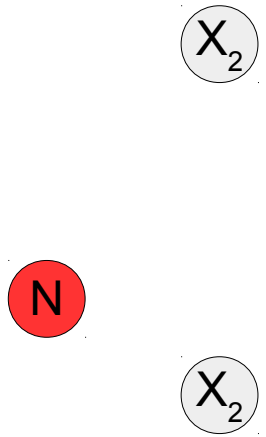
Stable CRN predicate computation (example)

predicate: $p(x_1,x_2)$: "$x_1 > x_2$"?

initial state: $\{ x_1\, X_1 , x_2\, X_2 , 1\, N \}$

reactions:

$$N + X_1 \rightarrow Y$$
$$Y + X_2 \rightarrow N$$



Stable CRN predicate computation (example)

predicate: $p(x_1,x_2)$: "$x_1=x_2$"?

initial state: $\{ x_1\ X_1\ ,\ x_2\ X_2\ ,\ 1\ Y \}$

Stable CRN predicate computation (example)

predicate: $p(x_1,x_2)$: "$x_1=x_2$"?

initial state: $\{ x_1\, X_1 , x_2\, X_2 , 1\, Y \}$

reactions:

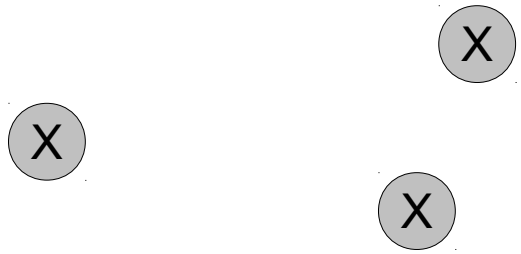
$$X_1 + X_2 \rightarrow Y$$
$$Y + N \rightarrow Y$$
$$X_1 + Y \rightarrow X_1 + N$$
$$X_2 + Y \rightarrow X_2 + N$$

Stable CRN function computation (example)

function: $f(x) = 2x$

Stable CRN function computation (example)

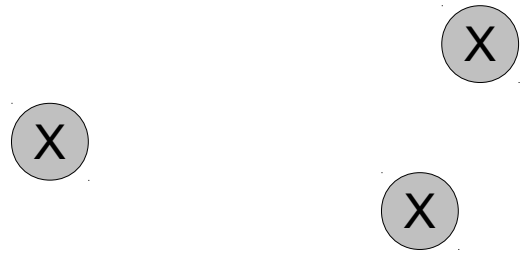
function: $f(x) = 2x$



Stable CRN function computation (example)

function: $f(x) = 2x$

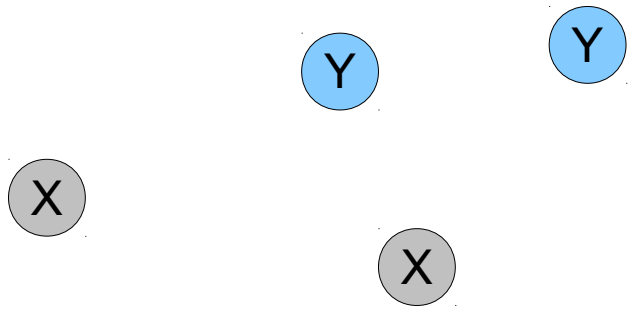
reactions: $X \longrightarrow 2Y$



Stable CRN function computation (example)

function: $f(x) = 2x$

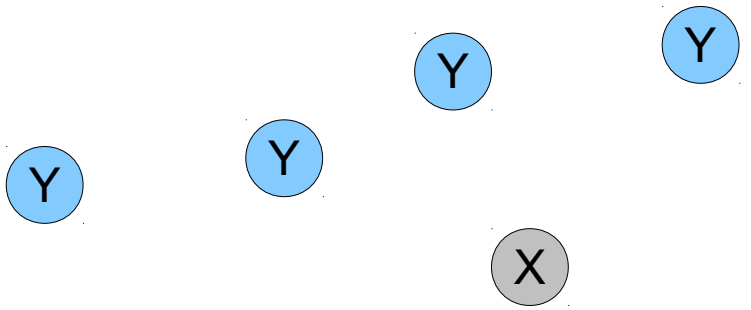
reactions: $X \longrightarrow 2Y$



Stable CRN function computation (example)

function: $f(x) = 2x$

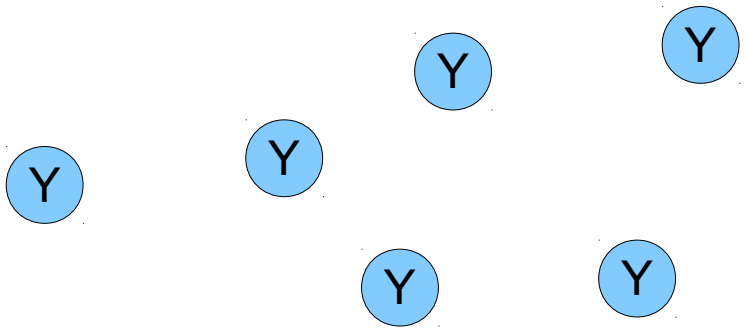
reactions: $X \longrightarrow 2Y$



Stable CRN function computation (example)

function: $f(x) = 2x$

reactions: $X \longrightarrow 2Y$

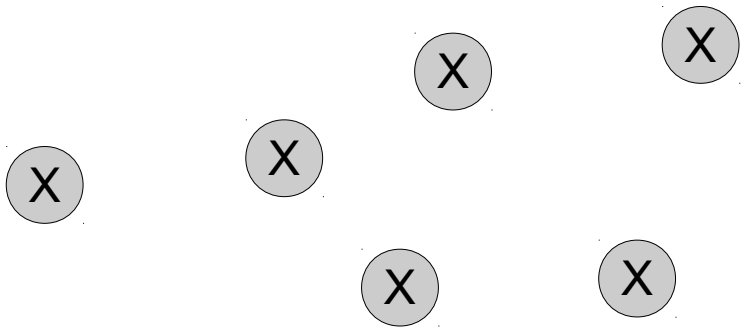


Stable CRN function computation (example)

function: $f(x) = x/2$

Stable CRN function computation (example)

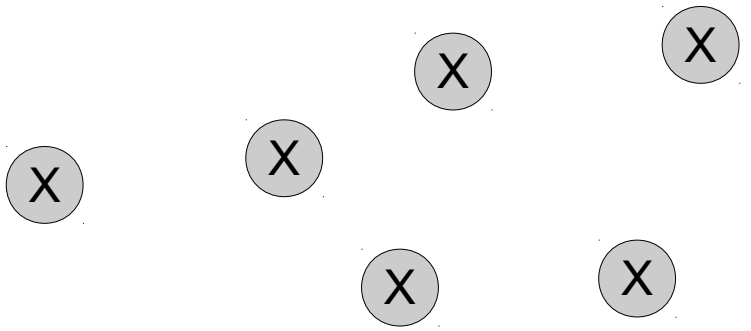
function: $f(x) = x/2$



Stable CRN function computation (example)

function: $f(x) = x/2$

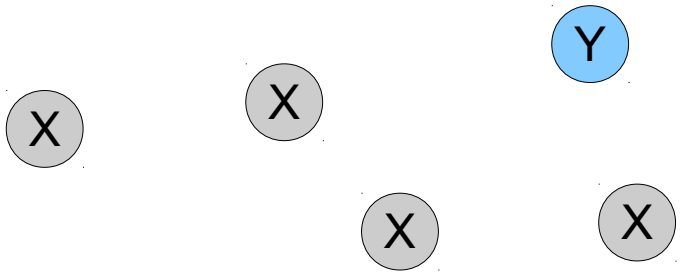
reactions: $2X \rightarrow Y$



Stable CRN function computation (example)

function: $f(x) = x/2$

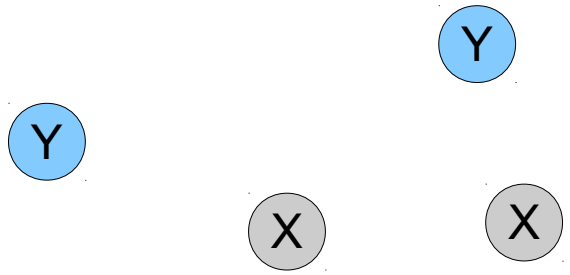
reactions: $2X \rightarrow Y$



Stable CRN function computation (example)

function: $f(x) = x/2$

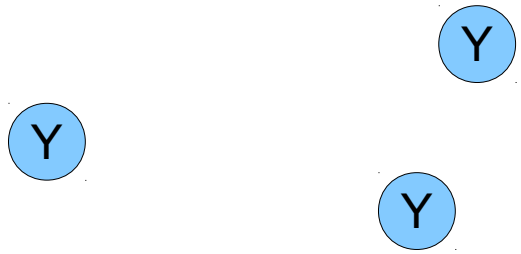
reactions: $2X \rightarrow Y$



Stable CRN function computation (example)

function: $f(x) = x/2$

reactions: $2X \rightarrow Y$



Stable CRN function computation (example)

function: $f(x_1,x_2) = x_1 + x_2$

Stable CRN function computation (example)

function: $f(x_1,x_2) = x_1 + x_2$

reactions: $X_1 \rightarrow Y$

$X_2 \rightarrow Y$

Stable CRN function computation (example)

function: $f(x_1,x_2) = x_1 - x_2$

Stable CRN function computation (example)

function: $f(x_1,x_2) = x_1 - x_2$

reactions: $X_1 \longrightarrow Y$

$X_2 + Y \longrightarrow$

Stable CRN function computation (example)

function: $f(x_1,x_2) = \min\{x_1,x_2\}$

Stable CRN function computation (example)

function: $f(x_1,x_2) = \min\{x_1,x_2\}$

reactions: $X_1 + X_2 \rightarrow Y$

Stable CRN function computation (example)

function: $f(x_1,x_2) = \max\{x_1,x_2\}$

Stable CRN function computation (example)

function: $f(x_1,x_2) = \max\{x_1,x_2\} = x_1 + x_2 - \min\{x_1,x_2\}$

Stable CRN function computation (example)

function: $f(x_1,x_2) = \max\{x_1,x_2\} = \boxed{x_1 + x_2} - \min\{x_1,x_2\}$

reactions:

$$\boxed{\begin{aligned} X_1 &\longrightarrow Y + X_1' \\ X_2 &\longrightarrow Y + X_2' \end{aligned}}$$

Stable CRN function computation (example)

function: $f(x_1,x_2) = \max\{x_1,x_2\} = x_1 + x_2 - \boxed{\min\{x_1,x_2\}}$

reactions:

$$X_1 \longrightarrow Y + X_1'$$

$$X_2 \longrightarrow Y + X_2'$$

$$\boxed{X_1' + X_2' \longrightarrow K}$$

Stable CRN function computation (example)

function: $f(x_1,x_2) = \max\{x_1,x_2\} = x_1 + x_2 - \min\{x_1,x_2\}$

reactions:

$$X_1 \longrightarrow Y + X_1'$$
$$X_2 \longrightarrow Y + X_2'$$
$$X_1' + X_2' \longrightarrow K$$
$$K + Y \longrightarrow$$

Stable CRN predicate computation (example)

predicate: $p(x_1,x_2)$: "$3x_1 > x_2/2$"?

initial state: $\{ x_1\ X_1\ ,\ x_2\ X_2\ ,\ 1\ N \}$

Stable CRN predicate computation (example)

predicate: $p(x_1,x_2)$: "$3x_1 > x_2/2$"?

initial state: $\{ x_1\, X_1 , x_2\, X_2 , 1\, N \}$

reactions:

$$X_1 \rightarrow 3Z_1$$
$$2X_2 \rightarrow Z_2$$

Stable CRN predicate computation (example)

predicate: $p(x_1,x_2)$: "$3x_1 > x_2/2$"?

initial state: $\{ x_1\, X_1 , x_2\, X_2 , 1\, N \}$

reactions:

$$X_1 \rightarrow 3Z_1$$
$$2X_2 \rightarrow Z_2$$
$$N + Z_1 \rightarrow Y$$
$$Y + Z_2 \rightarrow N$$

Stable computation characterization

Theorem: A predicate is stably computed by a CRN if and only if it is *semilinear*.

(Angluin, Aspnes, Diamadi, Fisher, Peralta, PODC 2004)

(Angluin, Aspnes, Eisenstat, PODC 2006)

Stable computation characterization

Theorem: A predicate is stably computed by a CRN if and only if it is *semilinear*.

"semilinear" = Boolean combination of *threshold* and *mod* tests

(Angluin, Aspnes, Diamadi, Fisher, Peralta, PODC 2004)

(Angluin, Aspnes, Eisenstat, PODC 2006)

Stable computation characterization

Theorem: A predicate is stably computed by a CRN if and only if it is *semilinear*.

"semilinear" = Boolean combination of *threshold* and *mod* tests

$x_1 - 3x_2 < -7$

(Angluin, Aspnes, Diamadi, Fisher, Peralta, PODC 2004)

(Angluin, Aspnes, Eisenstat, PODC 2006)

Stable computation characterization

Theorem: A predicate is stably computed by a CRN if and only if it is *semilinear*.

"semilinear" = Boolean combination of *threshold* and *mod* tests

$2x_1 + x_2 \equiv 3 \bmod 5$ $\qquad$ $x_1 - 3x_2 < -7$

(Angluin, Aspnes, Diamadi, Fisher, Peralta, PODC 2004)

(Angluin, Aspnes, Eisenstat, PODC 2006)

Stable computation characterization

Theorem: A predicate is stably computed by a CRN if and only if it is *semilinear*.

"semilinear" = Boolean combination of *threshold* and *mod* tests

$2x_1 + x_2 \equiv 3 \bmod 5$ $\qquad$ $x_1 - 3x_2 < -7$

(Angluin, Aspnes, Diamadi, Fisher, Peralta, PODC 2004)

(Angluin, Aspnes, Eisenstat, PODC 2006)

(Chen, Doty, Soloveichik, DNA 2012, for function computation)

Deciding output stability

Deciding output stability

- Each CRN has a set of p vectors $\{\mathbf{u}_1,...,\mathbf{u}_p\}$ such that $\mathbf{o}$ is output stable if and only if no $\mathbf{u}_i \leq \mathbf{o}$

Deciding output stability

- Each CRN has a set of p vectors $\{\mathbf{u}_1,...,\mathbf{u}_p\}$ such that $\mathbf{o}$ is output stable if and only if no $\mathbf{u}_i \leq \mathbf{o}$
- [Brijder, DNA 2014]: An algorithm can compute $\{\mathbf{u}_1,...,\mathbf{u}_p\}$ in time $O(p \log^{s-0.5}(p)\, r\, s^2 \log(u))$ for population protocols

$u = \max_i |\mathbf{u}_i|$

s = # species

r = # reactions

Deciding output stability

- Each CRN has a set of p vectors $\{\mathbf{u}_1,...,\mathbf{u}_p\}$ such that $\mathbf{o}$ is output stable if and only if no $\mathbf{u}_i \leq \mathbf{o}$
- [Brijder, DNA 2014]: An algorithm can compute $\{\mathbf{u}_1,...,\mathbf{u}_p\}$ in time $O(p \log^{s-0.5}(p)\, r\, s^2 \log(u))$ for population protocols

$u = \max_i |\mathbf{u}_i|$
s = # species
r = # reactions

- Open question: how big can p and u get?

Deciding output stability

- Each CRN has a set of p vectors $\{\mathbf{u}_1,\ldots,\mathbf{u}_p\}$ such that $\mathbf{o}$ is output stable if and only if no $\mathbf{u}_i \leq \mathbf{o}$
- [Brijder, DNA 2014]: An algorithm can compute $\{\mathbf{u}_1,\ldots,\mathbf{u}_p\}$ in time $O(p \log^{s-0.5}(p)\, r\, s^2 \log(u))$ for population protocols

$u = \max_i |\mathbf{u}_i|$
s = # species
r = # reactions

- Open question: how big can p and u get?
- Open question: extension to general CRNs?

Time complexity of stable computation

n = # molecules in initial state

Time complexity of stable computation

n = # molecules in initial state

$O(n)$ if initial state contains only input molecules

(Angluin, Aspnes, Eisenstat, PODC 2006, for predicates)
(Doty, Hajiaghayi, DNA 2013, for functions)

Time complexity of stable computation

n = # molecules in initial state

$O(n)$ if initial state contains only input molecules

(Angluin, Aspnes, Eisenstat, PODC 2006, for predicates)
(Doty, Hajiaghayi, DNA 2013, for functions)

$O(\text{polylog}(n))$ otherwise (if the CRN can start with a **leader**)

(Angluin, Aspnes, Eisenstat, DISC 2006, for predicates)
(Chen, Doty, Soloveichik, DNA 2012, for functions)

Time complexity of stable computation

n = # molecules in initial state

$O(n)$ if initial state contains only input molecules

(Angluin, Aspnes, Eisenstat, PODC 2006, for predicates)
(Doty, Hajiaghayi, DNA 2013, for functions)

$O(\text{polylog}(n))$ otherwise (if the CRN can start with a **leader**)

(Angluin, Aspnes, Eisenstat, DISC 2006, for predicates)
(Chen, Doty, Soloveichik, DNA 2012, for functions)

polylogarithmic time = "fast" = polynomial in binary expansion of n

linear time = "slow" = exponential in binary expansion of n

Time complexity in CRNs

time until next reaction = exponential r.v.

reaction	expected time
$X \rightarrow W + 2Y + Z$	$1 / \#X$
$A + B \rightarrow X$	volume / ($\#A \bullet \#B$)

Time complexity (example)

$\{n\ X\}$
$X \rightarrow Y + Y$

Time complexity (example)

$$\{n\ X\}$$
$$X \rightarrow Y + Y$$

E[time to consume all X] =

Time complexity (example)

$\{n\ X\}$

$X \rightarrow Y + Y$

E[time to consume all X] = E[time to consume first X]
+ E[time to consume second X]
+ E[time to consume third X]
+ ...
+ E[time to consume final X]

Time complexity (example)

$\{n\ X\}$

$X \rightarrow Y + Y$

E[time to consume all X] = E[time to consume first X]
+ E[time to consume second X]
+ E[time to consume third X]
+ ...
+ E[time to consume final X]

= 1/n + 1/(n-1) + 1/(n-2) + … + 1/1

≈ log n

Time complexity (example)

$\{n\ X\}$, volume n

$X + X \rightarrow Y$

E[time to consume all X] =

Time complexity (example)

$\{n\ X\}$, volume n

$X + X \rightarrow Y$

E[time to consume all X] = $n/n^2 + n/(n-2)^2 + n/(n-4)^2 + \ldots + n$

$< n(1/2^2 + 1/4^2 + 1/6^2 + 1/8^2 + \ldots)$

$= O(n)$

Time complexity (example)

$\{n\ X\}$, volume n

$X + X \rightarrow Y$

E[time to consume all X] = $n/n^2 + n/(n-2)^2 + n/(n-4)^2 + \ldots + n$

$< n(1/2^2 + 1/4^2 + 1/6^2 + 1/8^2 + \ldots)$

$= O(n)$

Time complexity (example)

$\{n\ X\}$, volume n

$X + X \rightarrow Y$

E[time to consume all X] = $n/n^2 + n/(n-2)^2 + n/(n-4)^2 + \ldots + n$

$< n(1/2^2 + 1/4^2 + 1/6^2 + 1/8^2 + \ldots)$

$= O(n)$

Time complexity (leader election)

{*n L*}, volume *n*

$L + L \rightarrow L$

Time complexity (leader election)

$\{n\ L\}$, volume n

$L + L \rightarrow L$

E[time get to 1 L] = $O(n)$

Time complexity (leader election)

$\{n\ L\}$, volume n

$L + L \longrightarrow L$

E[time get to 1 L] = $O(n)$

Is there a faster CRN?

Time complexity (leader election)

$\{n\ L\}$, volume n
$L + L \longrightarrow L$

E[time get to 1 L] = $O(n)$

Is there a faster CRN?

- If we really abuse the CRN model, yes (use $2X \rightarrow 3X$)
- In mass-conserving CRNs, we don't know
  - Angluin, Aspnes, Eisenstat [DISC 2006] have a PP that seems to work in simulation
  - If we require 0 probability of error, no (unpublished)

Time complexity (leader election)

$\{n\ L\}$, volume n
$L + L \longrightarrow L$

E[time get to 1 L] = $O(n)$

Is there a faster CRN?

- If we really abuse the CRN model, yes (use $2X \rightarrow 3X$)
- In mass-conserving CRNs, we don't know
  - Angluin, Aspnes, Eisenstat [DISC 2006] have a PP that seems to work in simulation
  - If we require 0 probability of error, no (unpublished)

Time complexity (leader election)

$\{n\ L\}$, volume n
$L + L \longrightarrow L$

E[time get to 1 L] = $O(n)$

Is there a faster CRN?

- If we really abuse the CRN model, yes (use $2X \rightarrow 3X$)
- In mass-conserving CRNs, we don't know
  - Angluin, Aspnes, Eisenstat [DISC 2006] have a PP that seems to work in simulation
  - If we require 0 probability of error, no (unpublished)

Time complexity (leader election)

$\{n\ L\}$, volume n
$L + L \longrightarrow L$

E[time get to 1 L] = $O(n)$

Is there a faster CRN?

- If we really abuse the CRN model, yes (use $2X \rightarrow 3X$)
- In mass-conserving CRNs, we don't know
  - Angluin, Aspnes, Eisenstat [DISC 2006] have a PP that seems to work in simulation
  - If we require 0 probability of error, no (unpublished)

What if we allow a small probability of error?
(Randomized CRN computation)

Randomized CRNs are Turing universal

"in a sense"

(Angluin, Aspnes, Eisenstat, DISC 2006)

(Soloveichik, Cook, Winfree, Bruck, Natural Computing 2008)

Randomized CRNs are Turing universal

Informal: A CRN can simulate a Turing machine with polynomial slowdown and small chance of error.

(Angluin, Aspnes, Eisenstat, DISC 2006) ← "in a sense"

(Soloveichik, Cook, Winfree, Bruck, Natural Computing 2008)

Randomized CRNs are Turing universal

Informal: A CRN can simulate a Turing machine with polynomial slowdown and small chance of error.

Implication: CRN simulation algorithms are the fastest way to predict their behavior.

"in a sense"

(Angluin, Aspnes, Eisenstat, DISC 2006)
(Soloveichik, Cook, Winfree, Bruck, Natural Computing 2008)

Randomized CRNs are Turing universal

Informal: A CRN can simulate a Turing machine with polynomial slowdown and small chance of error.

Implication: CRN simulation algorithms are the fastest way to predict their behavior.

Formal: For each TM M, there is a CRN C so that, for each $\varepsilon > 0$ and natural number n, there is an initial state $\mathbf{x}$ of C so that C simulates $M(n)$ with probability ε of error, and expected time poly(s•t), where t and s are the time and space usage of $M(n)$.

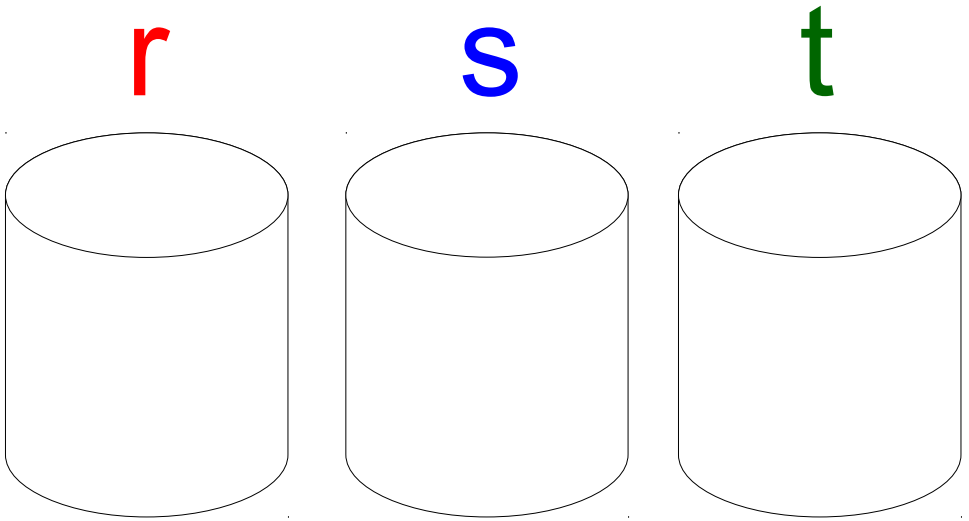
"in a sense"

(Angluin, Aspnes, Eisenstat, DISC 2006)

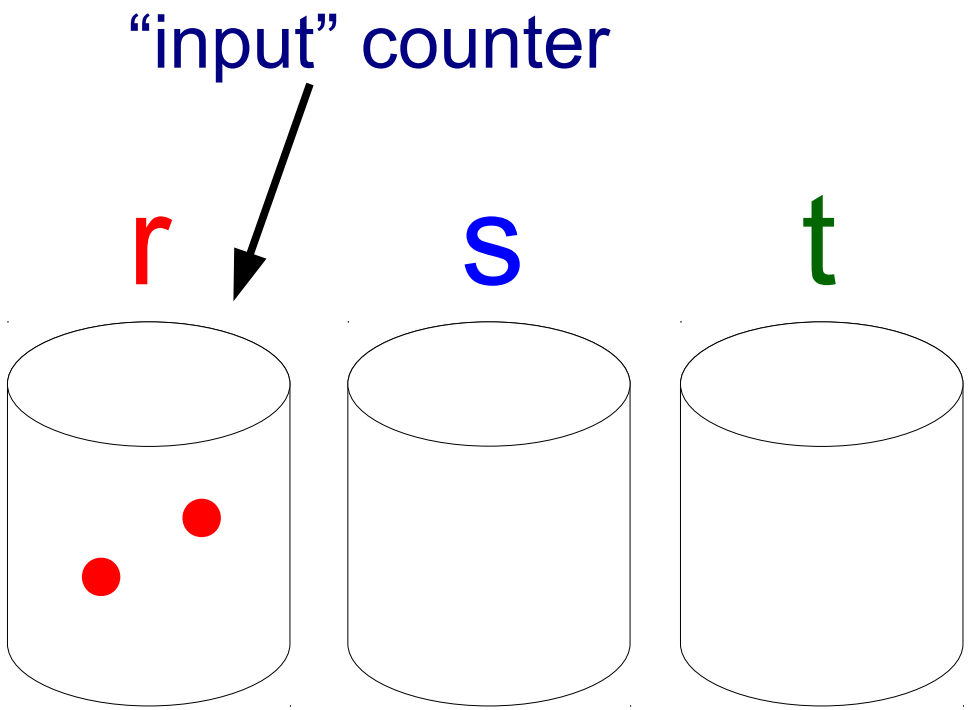
(Soloveichik, Cook, Winfree, Bruck, Natural Computing 2008)

Counter (register) machine

Counter (register) machine

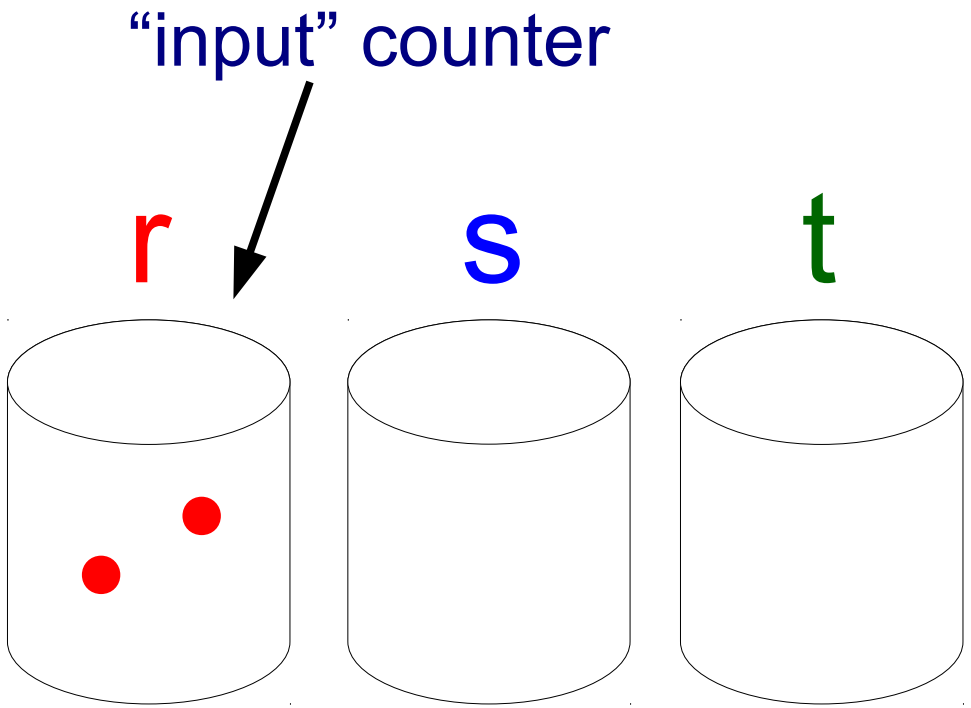


Counter (register) machine



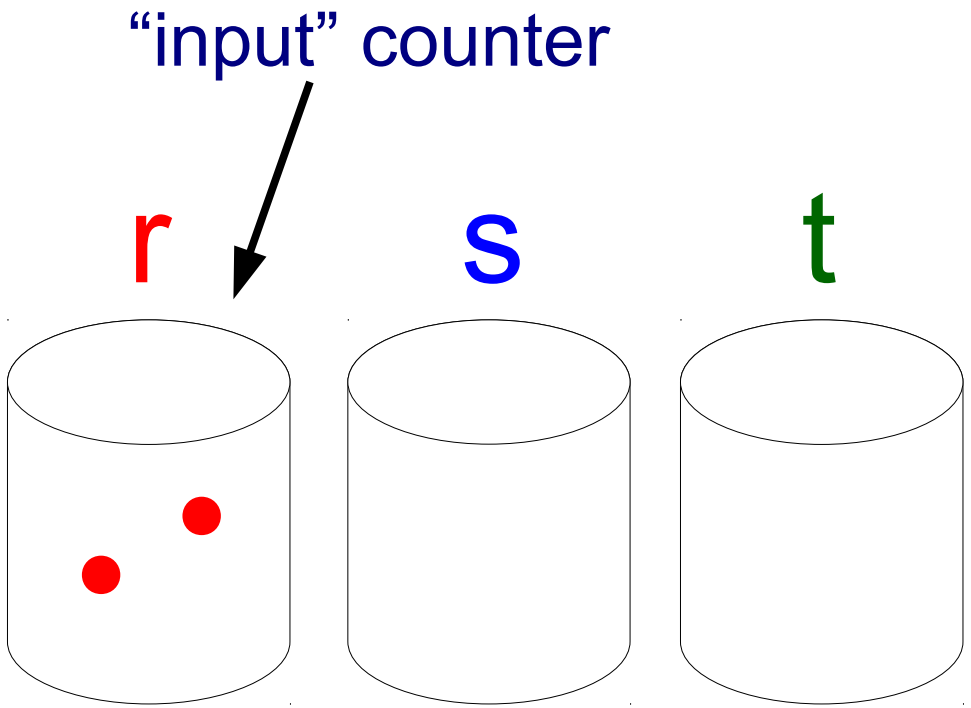
Counter (register) machine

1) *dec*(r)
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t)
6) *inc*(s)



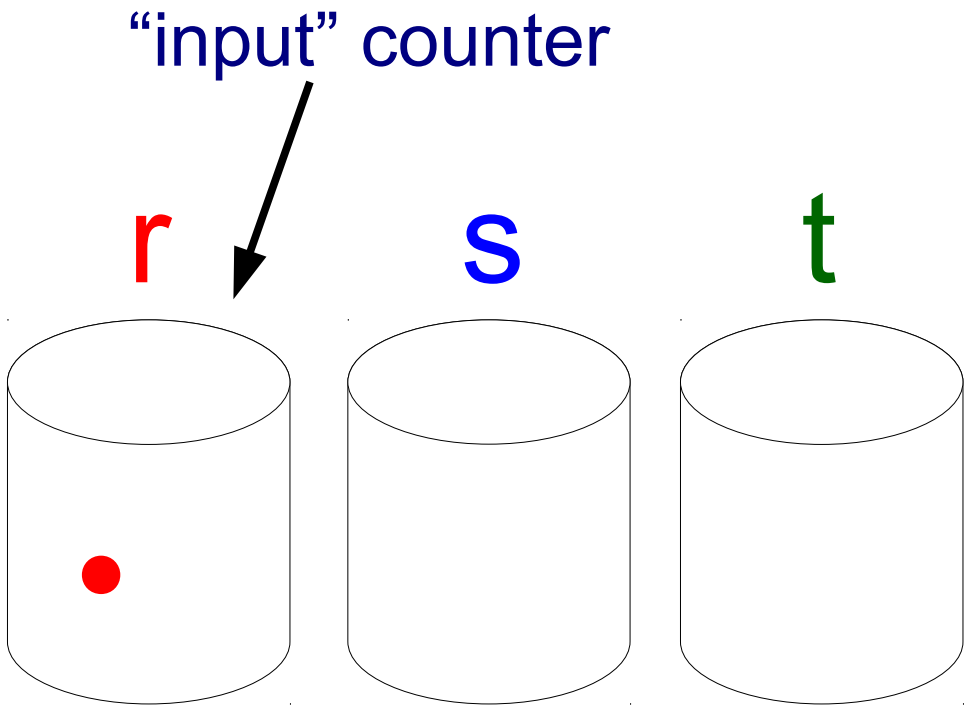
Counter (register) machine

1) *dec*(r)
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t)
6) *inc*(s)



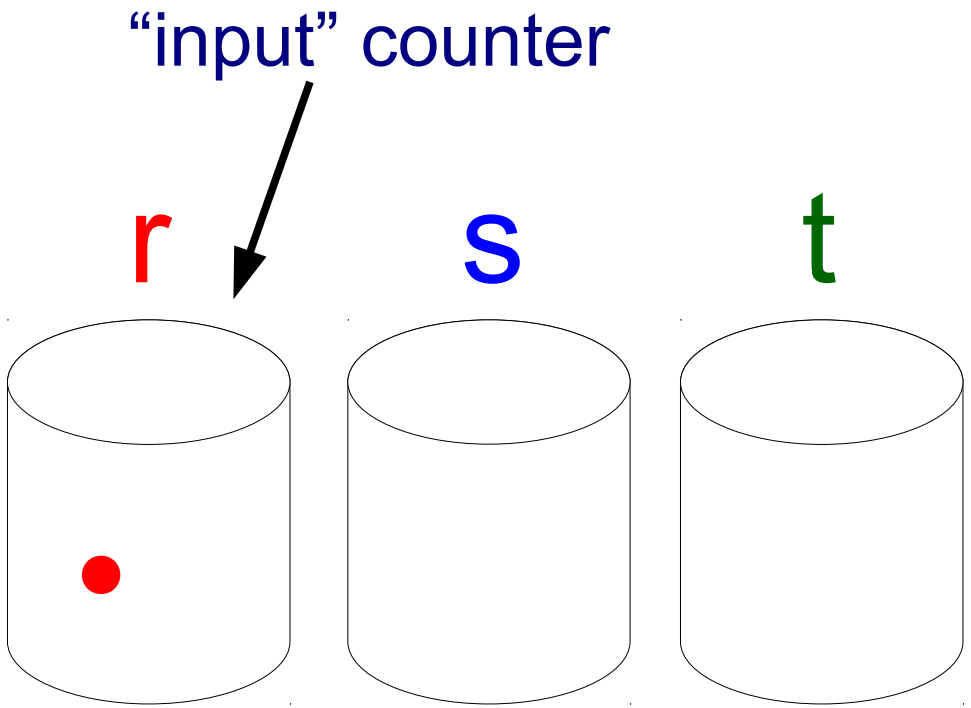
Counter (register) machine

1) *dec*(r)
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t)
6) *inc*(s)



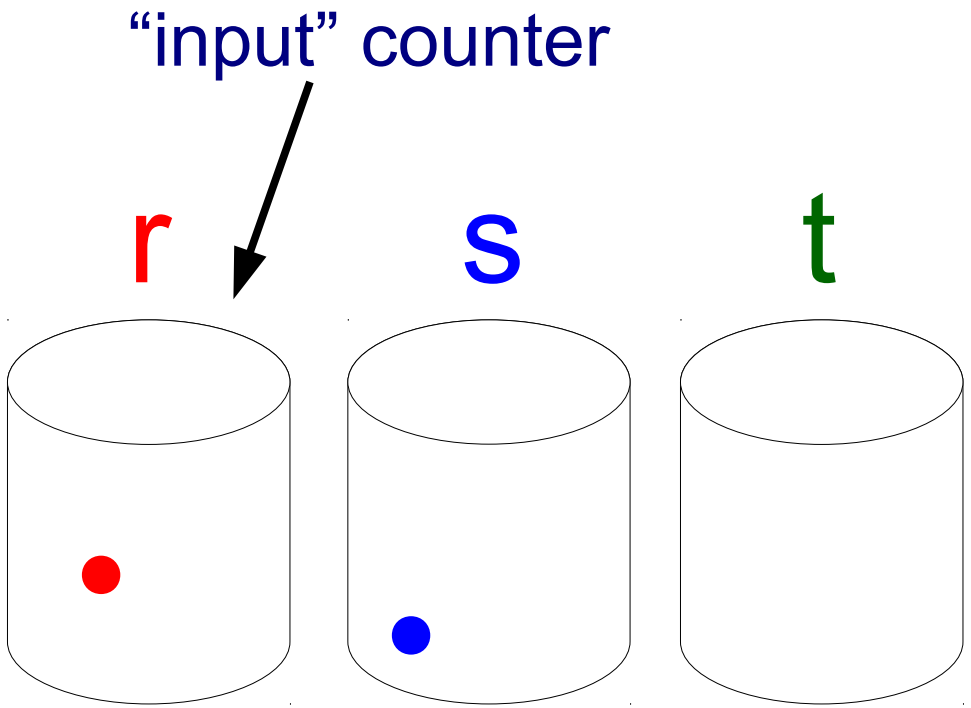
Counter (register) machine

1) *dec*(r)
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t)
6) *inc*(s)



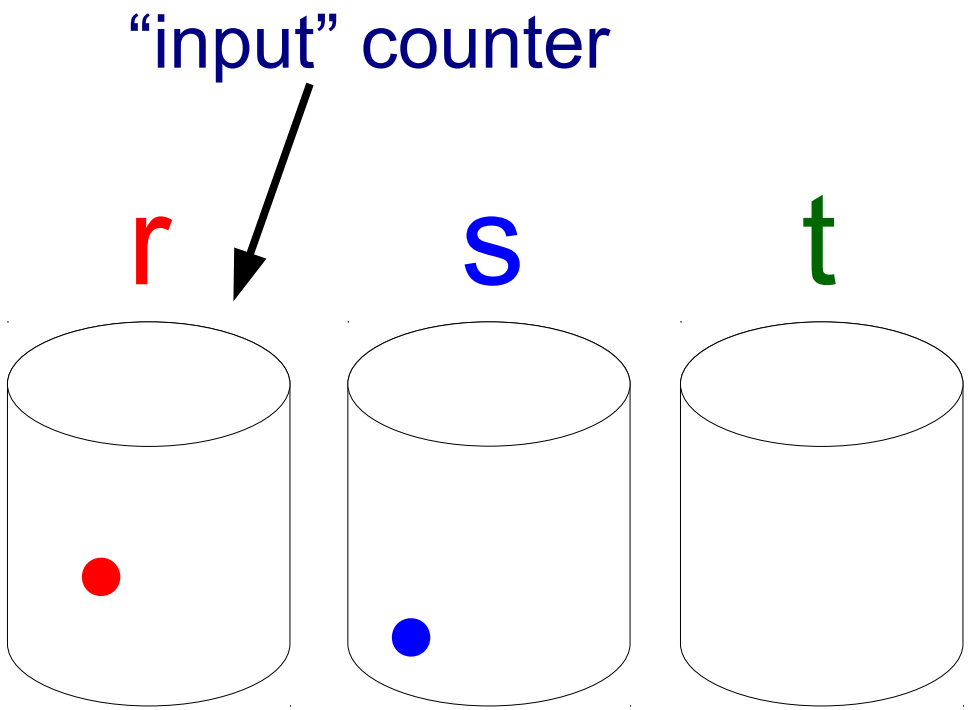
Counter (register) machine

1) *dec*(r)
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t)
6) *inc*(s)



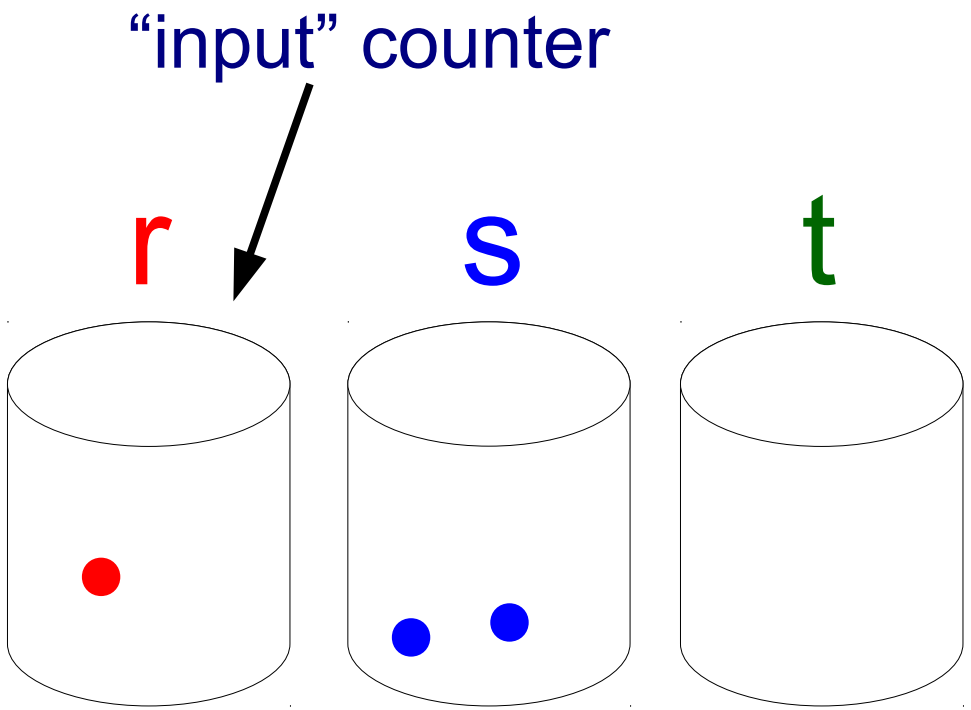
Counter (register) machine

1) *dec*(r)
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t)
6) *inc*(s)



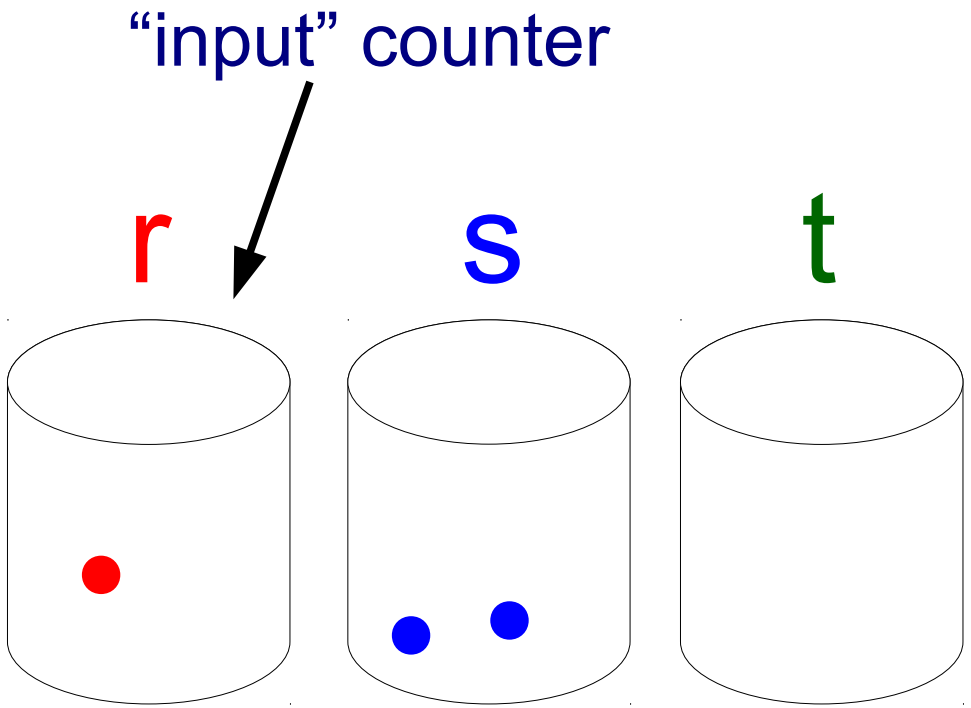
Counter (register) machine

1) *dec*(r)
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t)
6) *inc*(s)



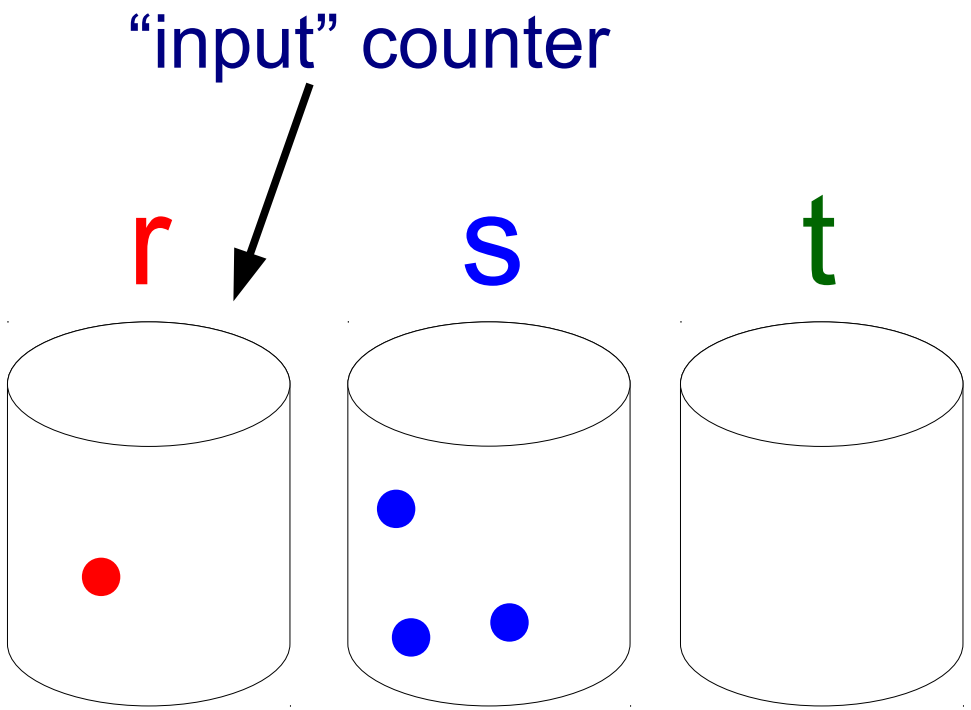
Counter (register) machine

1) *dec*(r)
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t)
6) *inc*(s)



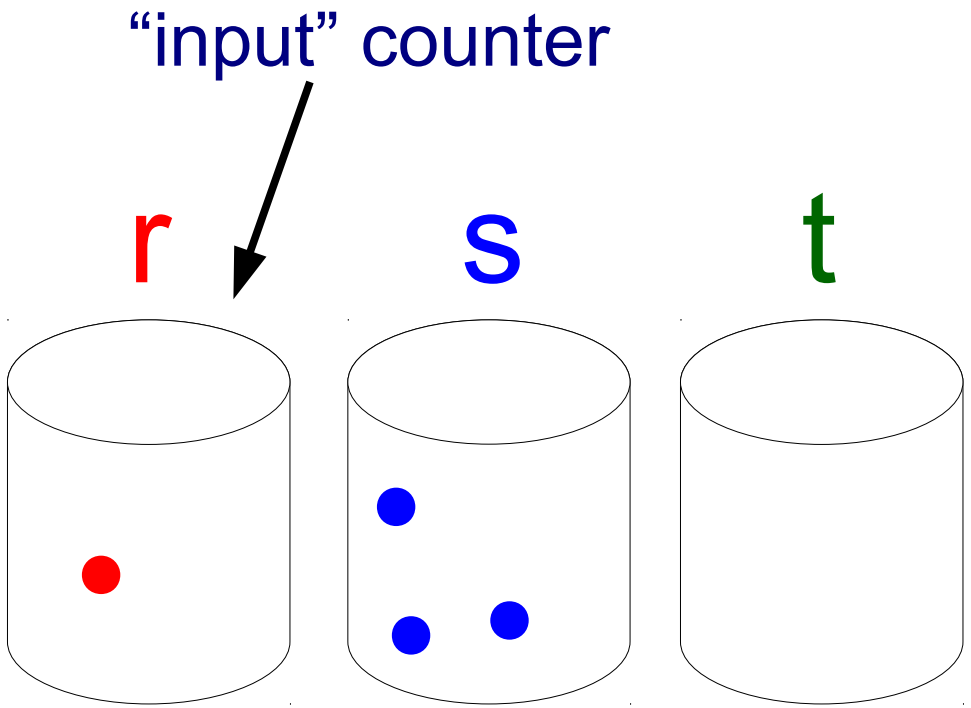
Counter (register) machine

1) *dec*(r)
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t)
6) *inc*(s)



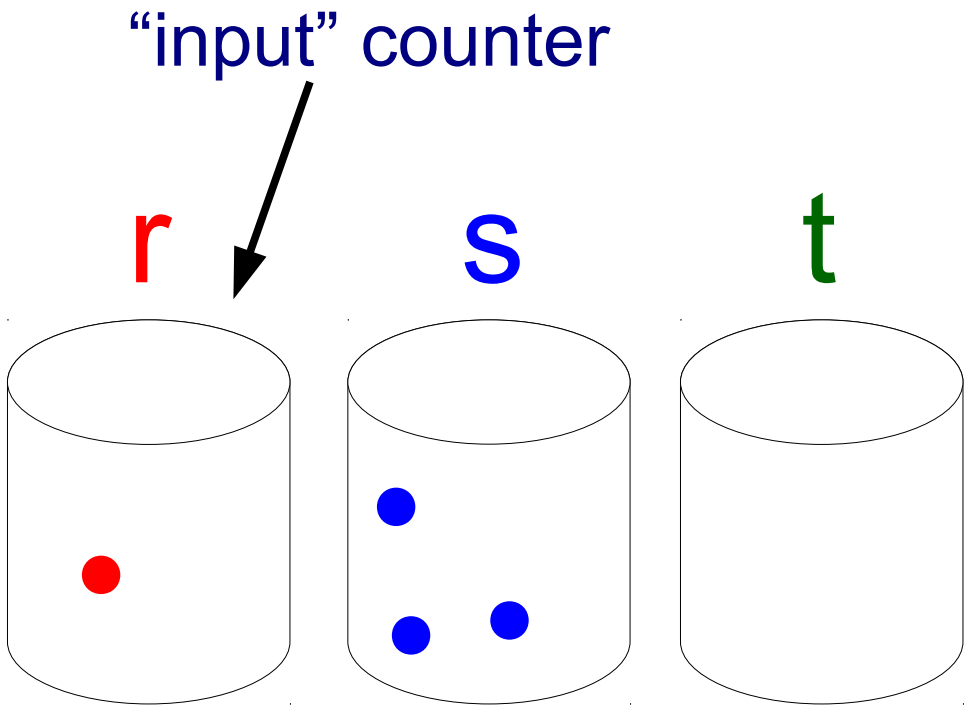
Counter (register) machine

1) *dec*(r)
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t)
6) *inc*(s)



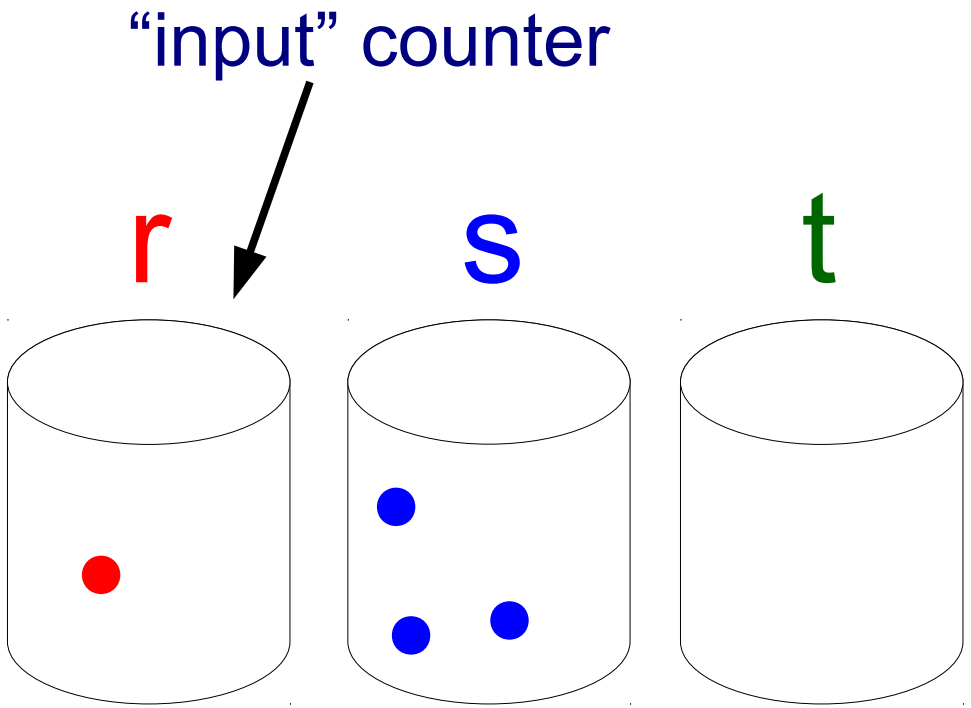
Counter (register) machine

1) *dec*(r) if empty goto 6
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t) if empty goto 1
6) *inc*(s)



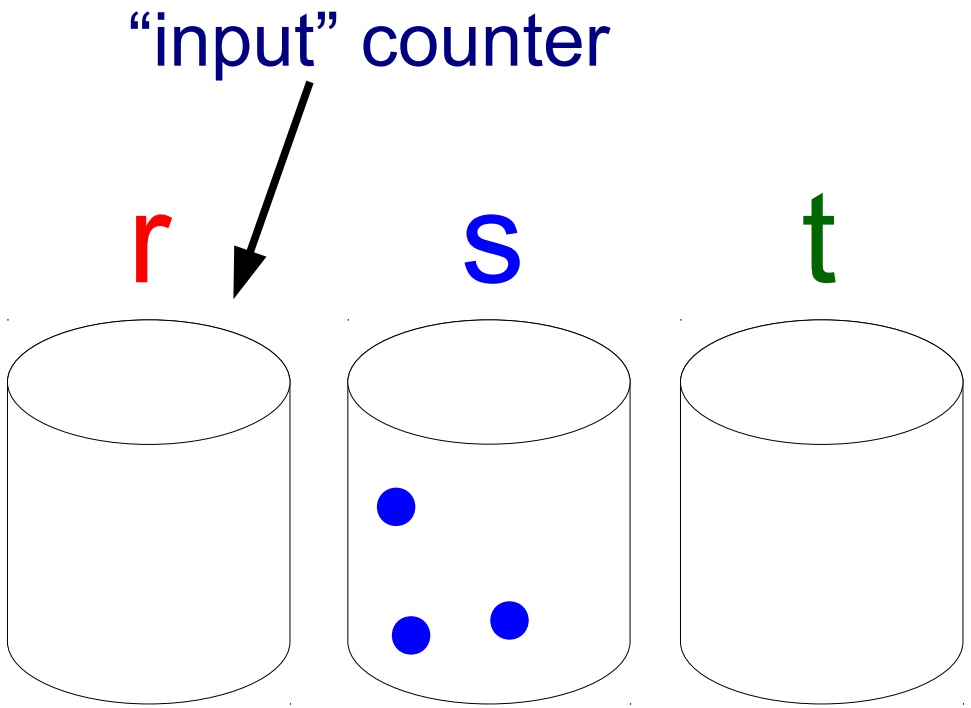
Counter (register) machine

1) *dec*(r) if empty goto 6
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t) if empty goto 1
6) *inc*(s)



Counter (register) machine

1) *dec*(r) if empty goto 6
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t) if empty goto 1
6) *inc*(s)



Counter (register) machine

1) *dec*(r) if empty goto 6

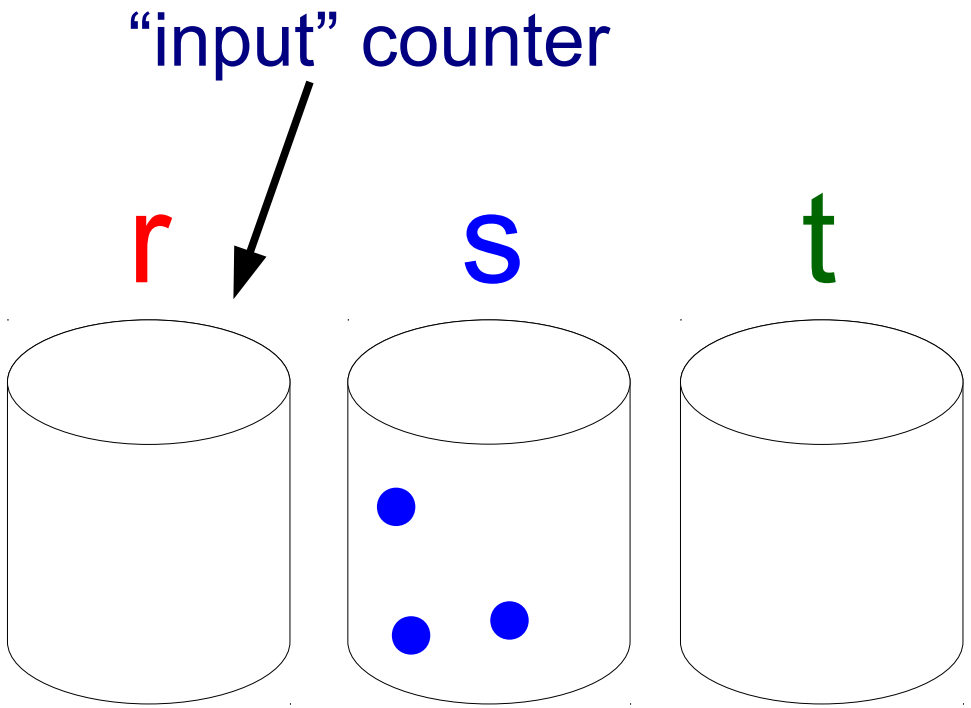
2) *inc*(s)

3) *inc*(s)

4) *inc*(s)

5) *dec*(t) if empty goto 1

6) *inc*(s)



Counter (register) machine

1) *dec*(r) if empty goto 6

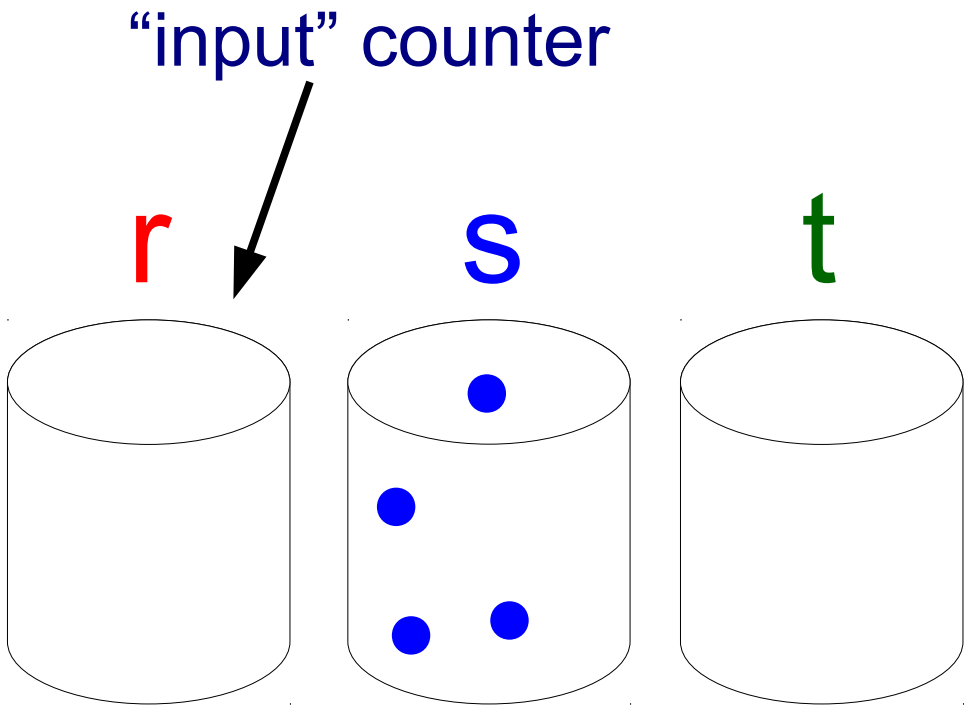
2) *inc*(s)

3) *inc*(s)

4) *inc*(s)

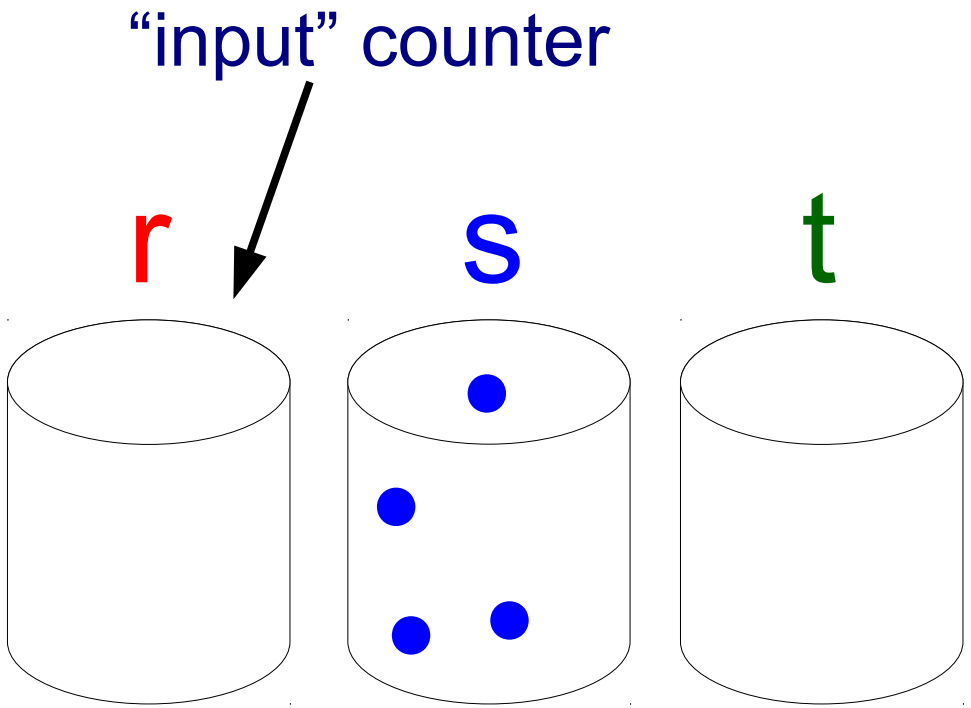
5) *dec*(t) if empty goto 1

6) *inc*(s)



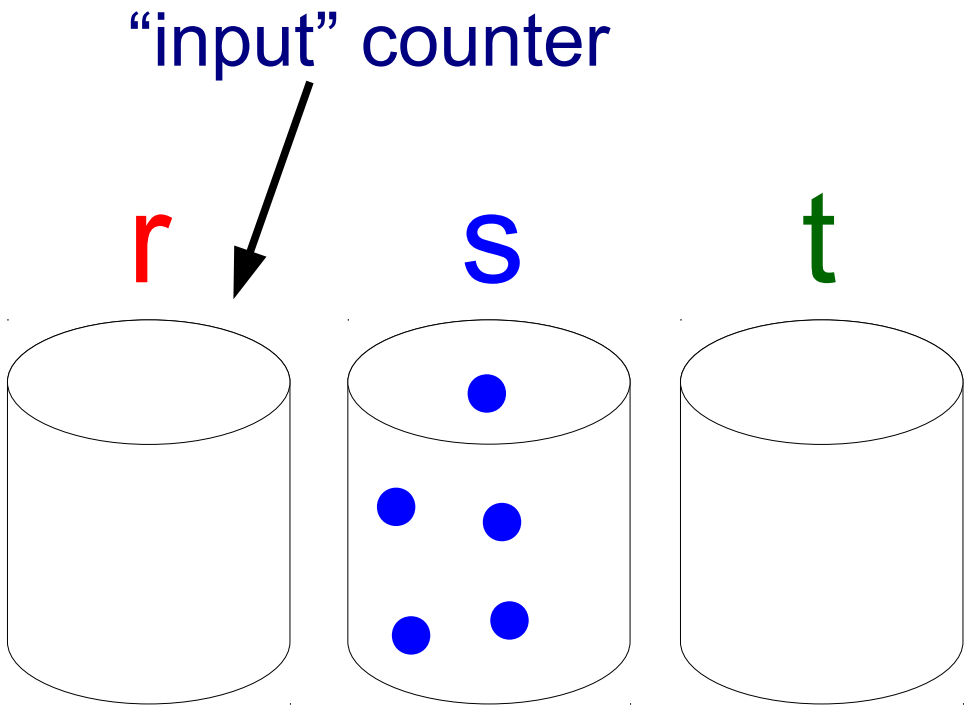
Counter (register) machine

1) *dec*(r) if empty goto 6
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t) if empty goto 1
6) *inc*(s)



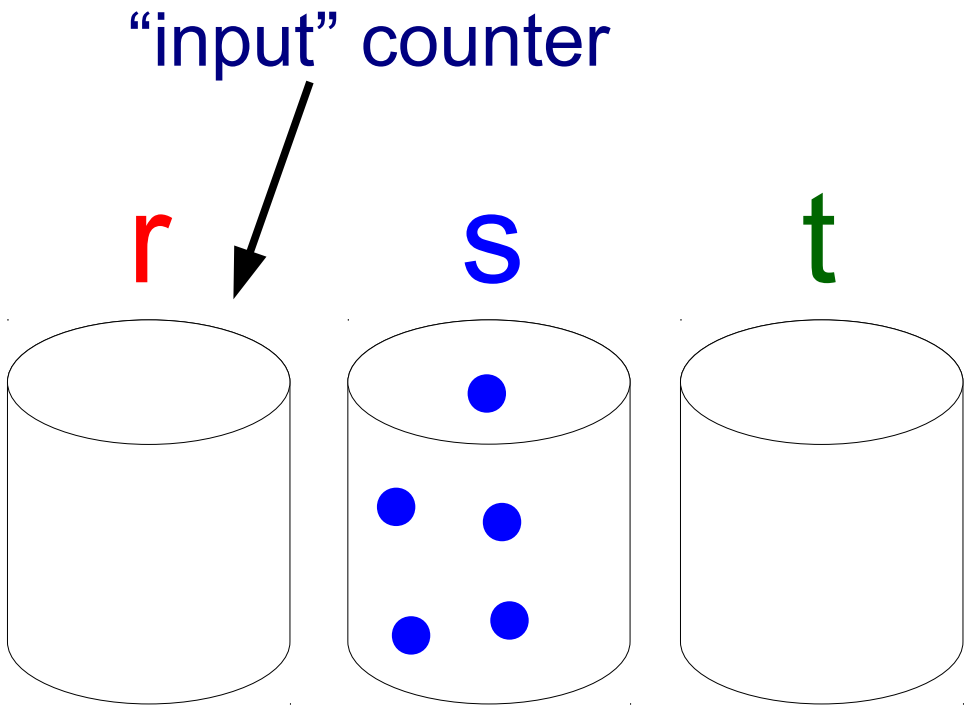
Counter (register) machine

1) *dec*(r) if empty goto 6
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t) if empty goto 1
6) *inc*(s)



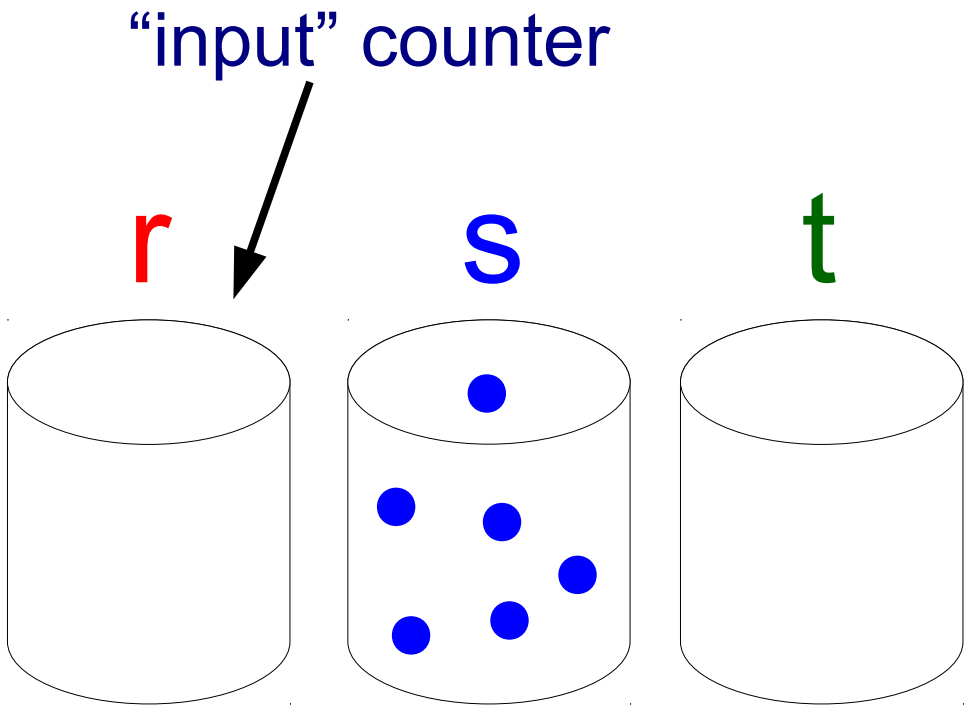
Counter (register) machine

1) *dec*(r) if empty goto 6
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t) if empty goto 1
6) *inc*(s)



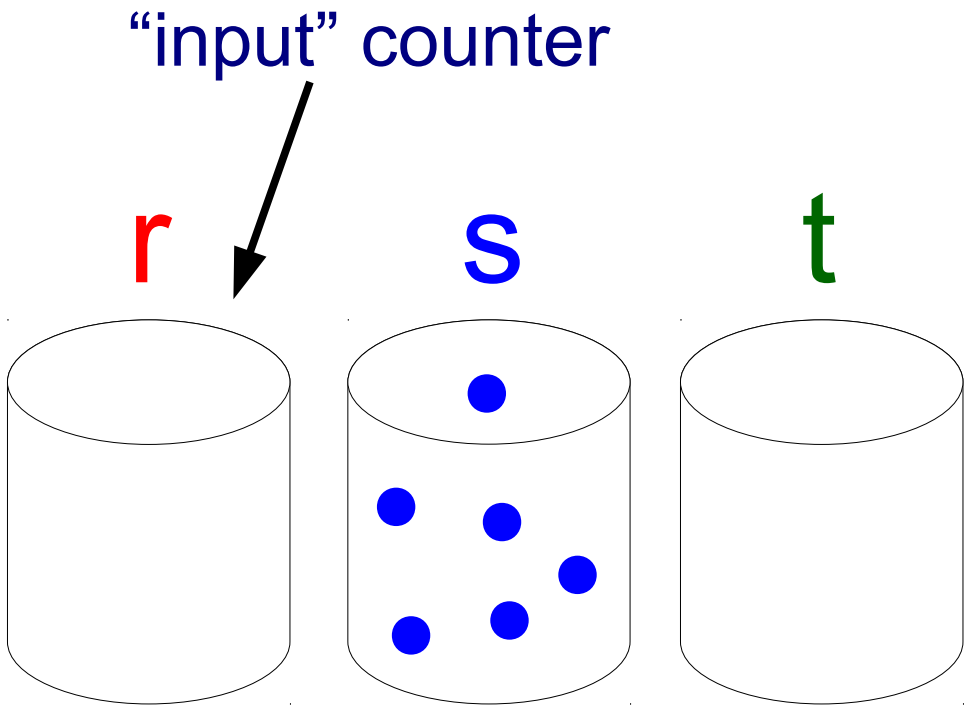
Counter (register) machine

1) *dec*(r) if empty goto 6
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t) if empty goto 1
6) *inc*(s)



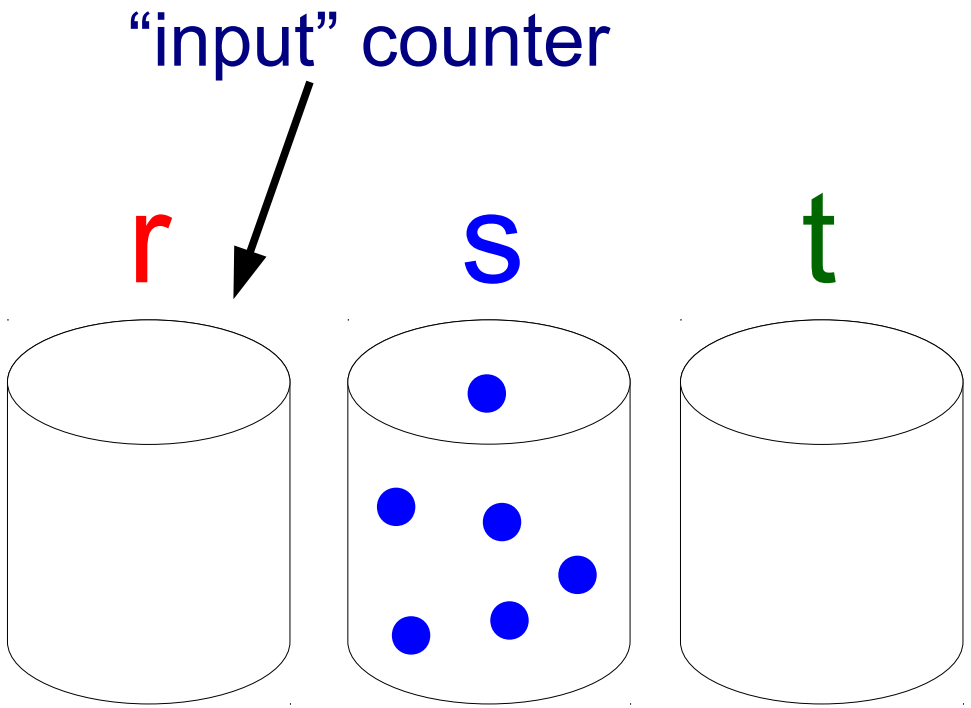
Counter (register) machine

1) *dec*(r) if empty goto 6
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t) if empty goto 1
6) *inc*(s)



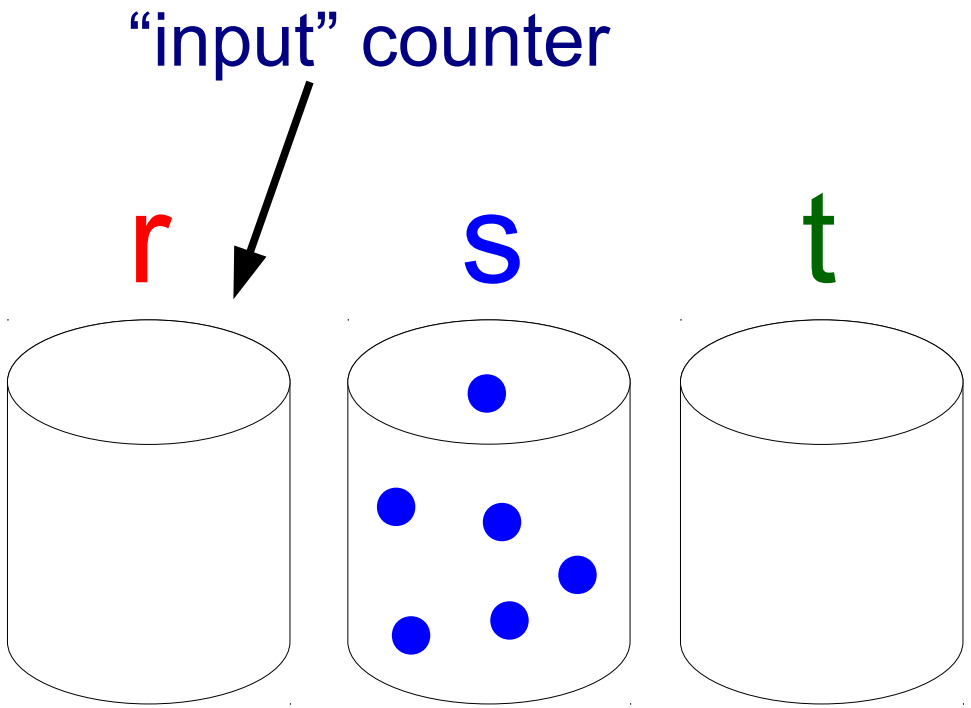
Counter (register) machine

1) *dec*(r) if empty goto 6
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t) if empty goto 1
6) *inc*(s)



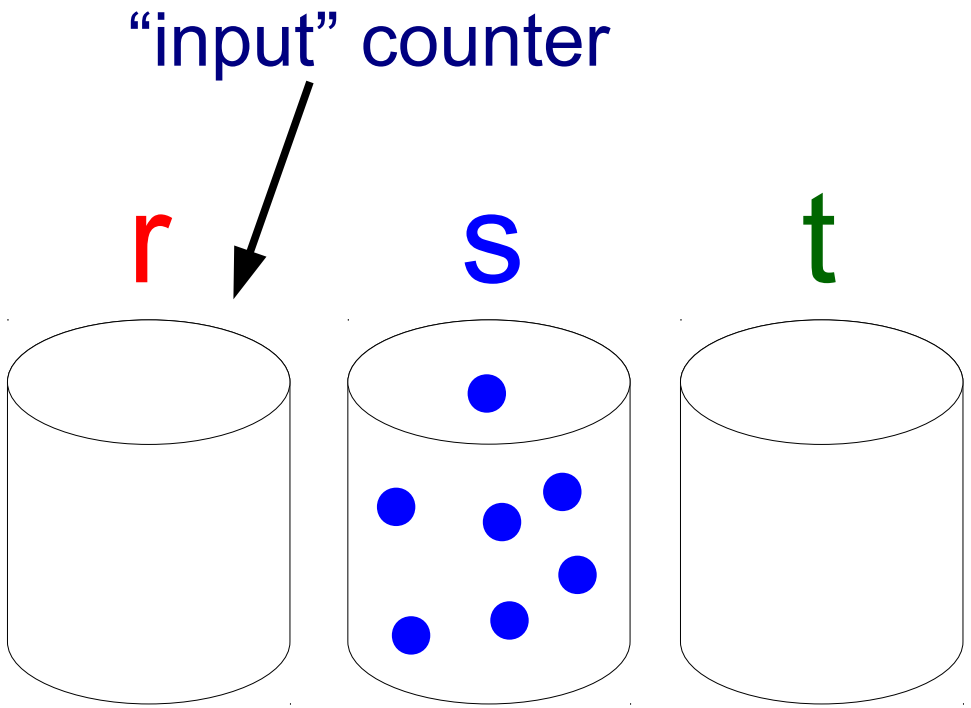
Counter (register) machine

1) *dec*(r) if empty goto 6
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t) if empty goto 1
6) *inc*(s)



Counter (register) machine

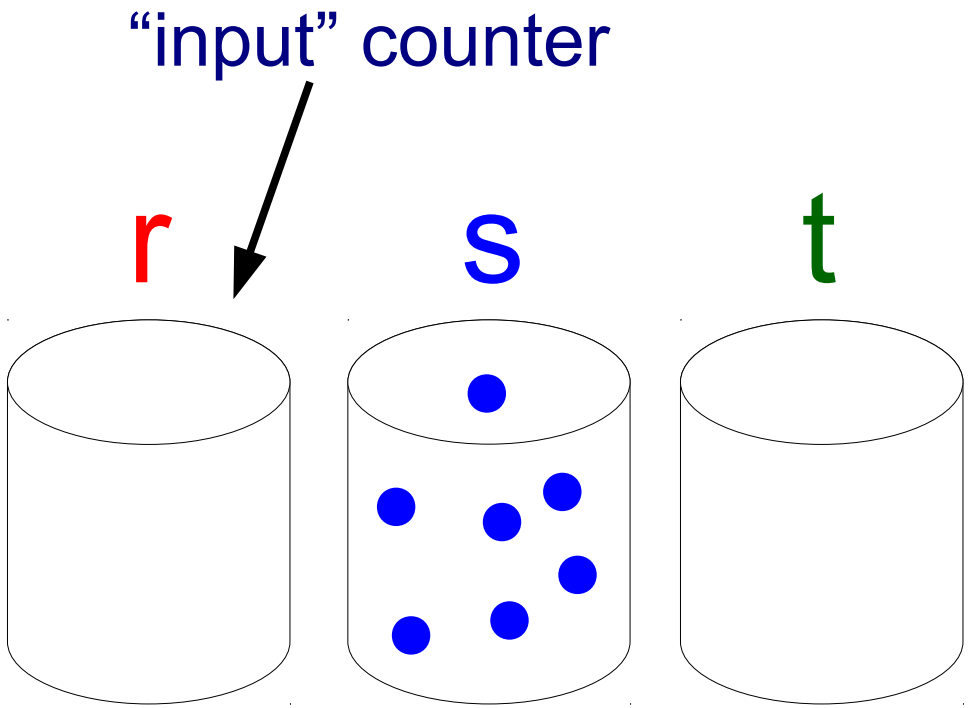
1) *dec*(r) if empty goto 6
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t) if empty goto 1
6) *inc*(s)



Counter (register) machine

1) *dec*(r) if empty goto 6
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t) if empty goto 1
6) *inc*(s)

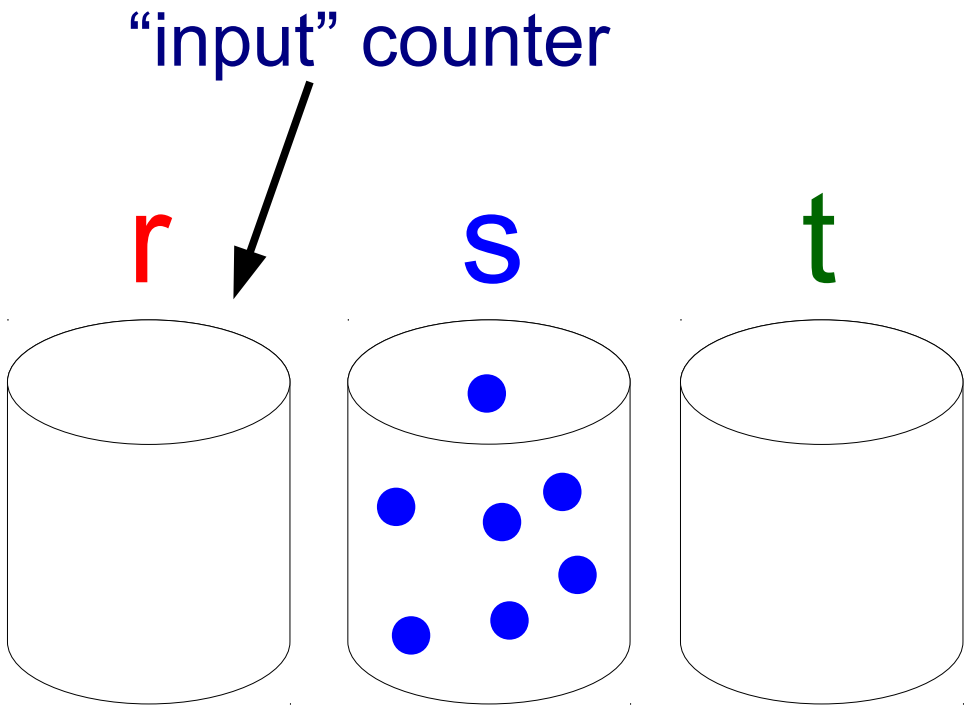
HALT



Counter (register) machine

1) *dec*(r) if empty goto 6
2) *inc*(s)
3) *inc*(s)
4) *inc*(s)
5) *dec*(t) if empty goto 1
6) *inc*(s)

HALT



computes $f(n) = 3n+1$

CRNs can simulate counter machines with probability < 1

CRNs can simulate counter machines with probability < 1

Counter machine:

r = input *n*, start line 1

1) *inc*(r)
2) *dec*(r) if zero goto 1
3) *inc*(s)
4) *dec*(s) if zero goto 2

CRNs can simulate counter machines with probability < 1

Counter machine:

r = input *n*, start line 1

1) *inc*(r)
2) *dec*(r) if zero goto 1
3) *inc*(s)
4) *dec*(s) if zero goto 2

CRN:

initial state $\{n\ R, 1\ L_1\}$

CRNs can simulate counter machines with probability < 1

Counter machine:

r = input *n*, start line 1

1) *inc*(r)
2) *dec*(r) if zero goto 1
3) *inc*(s)
4) *dec*(s) if zero goto 2

CRN:

initial state $\{n\ R,\ 1\ L_1\}$

$$L_1 \rightarrow L_2 + R$$

CRNs can simulate counter machines with probability < 1

Counter machine:

r = input *n*, start line 1

1) *inc*(r)
2) *dec*(r) if zero goto 1
3) *inc*(s)
4) *dec*(s) if zero goto 2

CRN:

initial state $\{n\ R,\ 1\ L_1\}$

$$L_1 \rightarrow L_2 + R$$

$$L_2 + R \rightarrow L_3$$

CRNs can simulate counter machines with probability < 1

Counter machine:

r = input *n*, start line 1

1) *inc*(r)
2) *dec*(r) if zero goto 1
3) *inc*(s)
4) *dec*(s) if zero goto 2

CRN:

initial state $\{n\ R,\ 1\ L_1\}$

$$L_1 \rightarrow L_2 + R$$

$$L_2 + R \rightarrow L_3$$

CRNs can simulate counter machines with probability < 1

Counter machine:

r = input *n*, start line 1

1) *inc*(r)
2) *dec*(r) if zero goto 1
3) *inc*(s)
4) *dec*(s) if zero goto 2

CRN:

initial state $\{n\ R, 1\ L_1\}$

$L_1 \rightarrow L_2 + R$

$L_2 + R \rightarrow L_3$; $L_2 \rightarrow L_1$

CRNs can simulate counter machines with probability < 1

Counter machine:

r = input *n*, start line 1

1) *inc*(r)
2) *dec*(r) if zero goto 1
3) *inc*(s)
4) *dec*(s) if zero goto 2

CRN:

initial state $\{n\ R,\ 1\ L_1\}$

$$L_1 \rightarrow L_2 + R$$

$$L_2 + R \rightarrow L_3 \ ; \ L_2 \rightarrow L_1$$

$$L_3 \rightarrow L_4 + S$$

$$L_4 + S \rightarrow L_5 \ ; \ L_4 \rightarrow L_2$$

CRNs can simulate counter machines with probability < 1

Counter machine:

r = input *n*, start line 1

1) *inc*(r)
2) *dec*(r) if zero goto 1
3) *inc*(s)
4) *dec*(s) if zero goto 2

CRN:

initial state $\{n\ R,\ 1\ L_1\}$

$L_1 \rightarrow L_2 + R$

$L_2 + R \rightarrow L_3$; $L_2 \rightarrow L_1$

$L_3 \rightarrow L_4 + S$

$L_4 + S \rightarrow L_5$; $L_4 \rightarrow L_2$

Need to be **very** slow!

How to slow down reaction $L_2 \rightarrow L_1$?

How to slow down reaction $L_2 \rightarrow L_1$?

Use a clock:

1 C_1, 1 F, n B

How to slow down reaction $L_2 \rightarrow L_1$?

Use a clock:

1 C_1, 1 F, n B

$$F + C_1 \rightarrow F + C_2 \qquad B + C_2 \rightarrow B + C_1$$

$$F + C_2 \rightarrow F + C_3 \qquad B + C_3 \rightarrow B + C_2$$

$$\vdots$$

How to slow down reaction $L_2 \rightarrow L_1$?

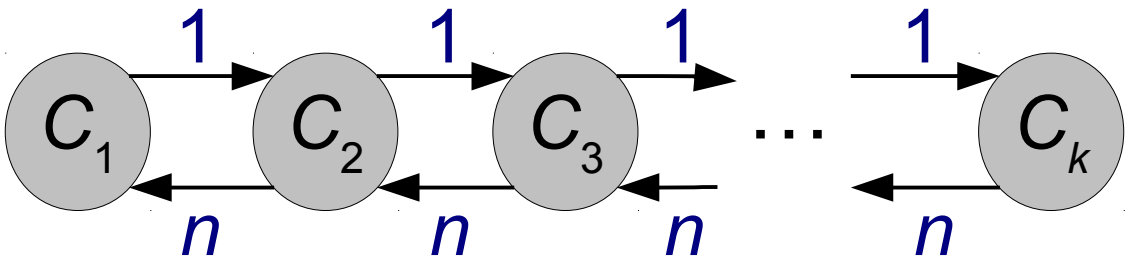
Use a clock:

1 C_1, 1 F, n B

$$F + C_1 \rightarrow F + C_2 \qquad B + C_2 \rightarrow B + C_1$$

$$F + C_2 \rightarrow F + C_3 \qquad B + C_3 \rightarrow B + C_2$$

$$\vdots$$



reverse-biased random walk

How to slow down reaction $L_2 \rightarrow L_1$?

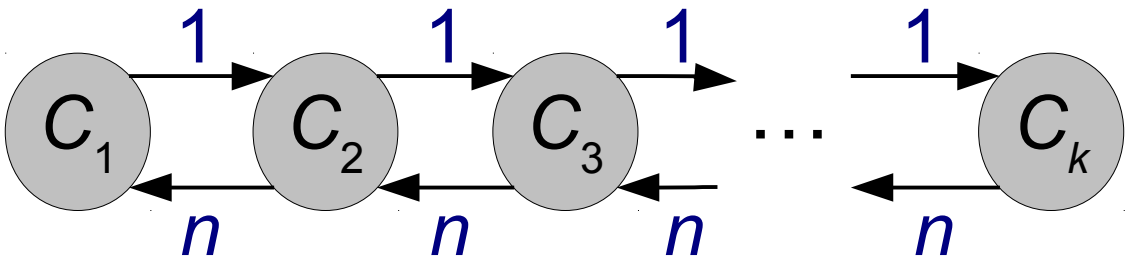
Use a clock:

1 C_1, 1 F, n B

$$F + C_1 \rightarrow F + C_2 \qquad B + C_2 \rightarrow B + C_1$$

$$F + C_2 \rightarrow F + C_3 \qquad B + C_3 \rightarrow B + C_2$$

$$\vdots$$



reverse-biased random walk

C_k appears after expected time $\approx n^{k-1}$

How to slow down reaction $L_2 \to L_1$?

Use a clock:

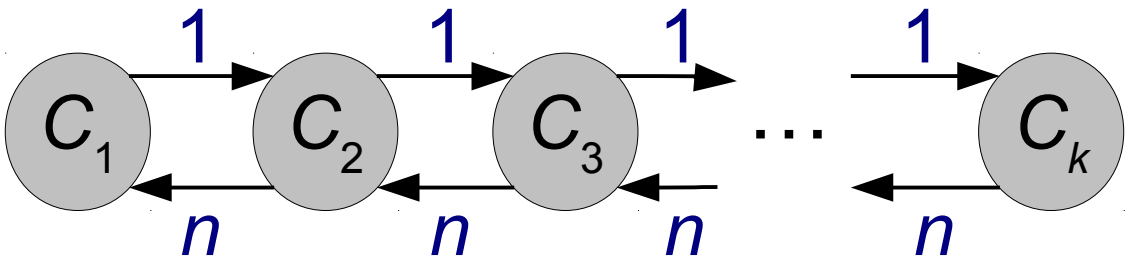
1 C_1, 1 F, n B

$$C_k + L_2 \to C_1 + L_1$$

$$F + C_1 \to F + C_2 \qquad B + C_2 \to B + C_1$$

$$F + C_2 \to F + C_3 \qquad B + C_3 \to B + C_2$$

$$\vdots$$



reverse-biased random walk

C_k appears after expected time $\approx n^{k-1}$

How to slow down reaction $L_2 \to L_1$?

Use a clock:

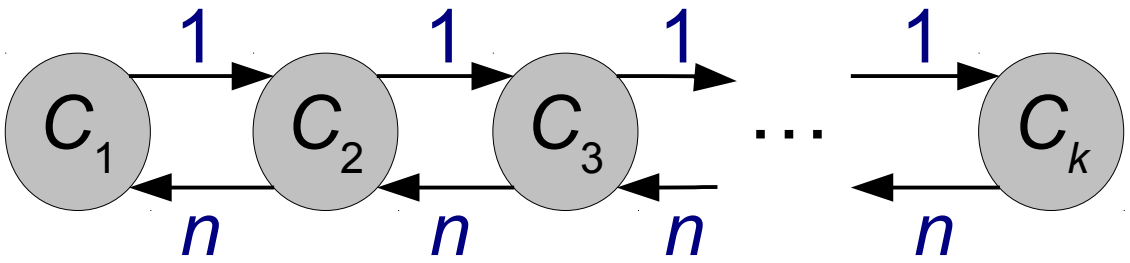
1 C_1, 1 F, n B

$$C_k + L_2 \to C_1 + L_1$$

$$F + C_1 \to F + C_2 \qquad B + C_2 \to B + C_1$$

$$F + C_2 \to F + C_3 \qquad B + C_3 \to B + C_2$$

$$\vdots$$



C_k appears after expected time $\approx n^{k-1}$

reverse-biased random walk

E[time for $L_2 + R \to L_3$] $\leq n$

Probability 1 computation

Probability 1 computation

- Errr... isn't that stable computation?

Probability 1 computation

- Errr... isn't that stable computation?
- With finite state space (e.g. population protocols), yes.

Probability 1 computation

- Errr... isn't that stable computation?
- With finite state space (e.g. population protocols), yes.

Consider...

$$Y \xrightarrow{2} 2Y$$

$$Y \xrightarrow{1}$$

$$\xrightarrow{1} Y$$

Probability 1 computation

- Errr... isn't that stable computation?
- With finite state space (e.g. population protocols), yes.

Consider... $Y \xrightarrow{2} 2Y$

$Y \xrightarrow{1}$

initial state $\{1Y, 1N\}$

Probability 1 computation

- Errr... isn't that stable computation?
- With finite state space (e.g. population protocols), yes.

Consider... $Y \xrightarrow{2} 2Y$, $Y \xrightarrow{1}$ initial state $\{1Y, 1N\}$

Theorem: All (Turing) computable predicates can be computed by a CRN with probability 1.

(Cummings, Doty, Soloveichik, DNA 2014)