CORNELL
UNIVERSITY

Behavioral Simulations in MapReduce

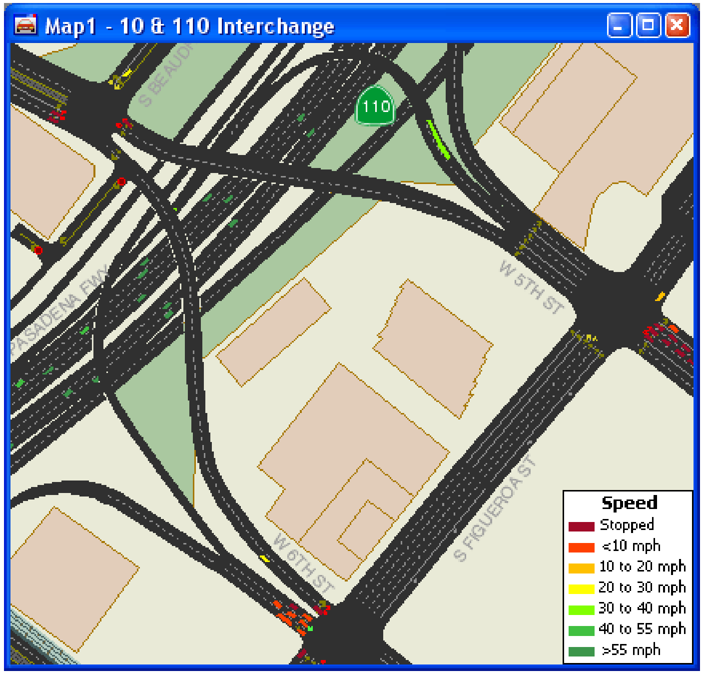
Guozhang Wang, Marcos Vaz Salles,
Benjamin Sowell, Xun Wang, Tuan Cao, Alan Demers,
Johannes Gehrke, Walker White

Cornell University

What are Behavioral Simulations?

- Simulations of individuals that interact to create emerging behavior in complex systems

- Application Areas
  - Traffic networks
  - Ecology systems
  - Sociology systems
  - etc



Why Behavioral Simulations?

- Traffic
  - Congestion cost $87.2 billion in the U.S. in 2007
  - More people killed by air pollution than accidents
  - Detailed models: micro-simulators not scale to NYC!

- Ecology
  - Hard to scale to large fish schools or locust swarms

Challenges of Behavioral Simulations

- ***Easy to program → not scalable***
  - Examples: Swarm, Mason
  - Typically one thread per agent, lots of contention
- ***Scalable → hard to program***
  - Examples: TRANSIMS, DynaMIT (traffic), GPU implementation of fish simulation (ecology)
  - Hard-coded models, compromise level of detail

Challenges of Behavioral Simulations

- ***Easy to program → not scalable***
  - Examples: Swarm, Mason
  - n
- ***S***

Can we do better?

  - Examples: TRANSIMS, DynaMIT (traffic), GPU implementation of fish simulation (ecology)
  - Hard-coded models, compromise level of detail

Our Contribution

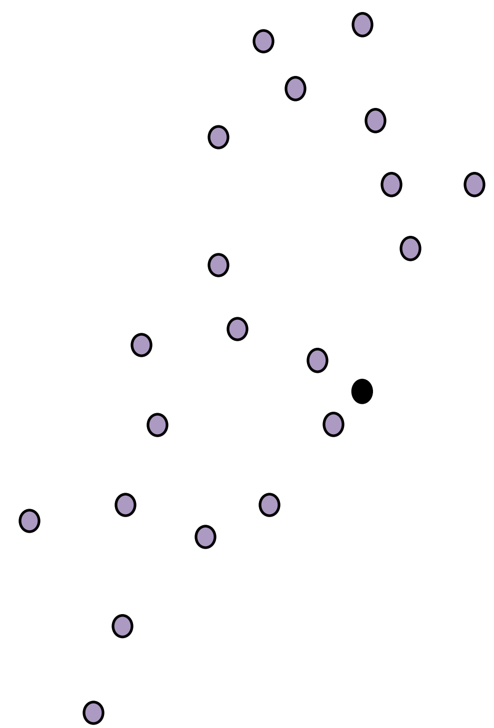
- A new simulation platform that combines:
  - Ease of programming
    - Program simulations in State-Effect pattern
    - *BRASIL*: Scripting language for domain scientists
  - Scalability
    - Execute simulations in the MapReduce model
    - *BRACE*: Special-purpose MapReduce engine

Talk Outline

- Motivation
- Ease of Programming
  - Program Simulations in State-Effect Pattern
  - BRASIL
- Scalability
  - Execute Simulations in MapReduce Model
  - BRACE
- Experiments
- Conclusion

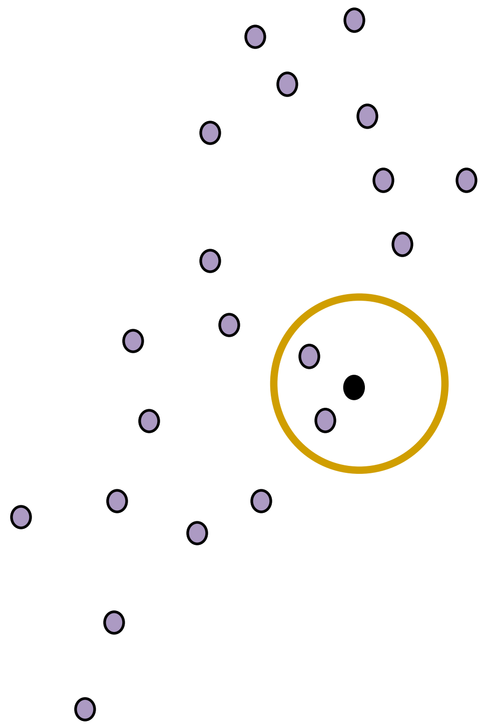
A Running Example: Fish Schools

- Adapted from Couzin et al., Nature 2005
- Fish Behavior
  - Avoidance: if too close, repel other fish
  - Attraction: if seen within range, attract other fish



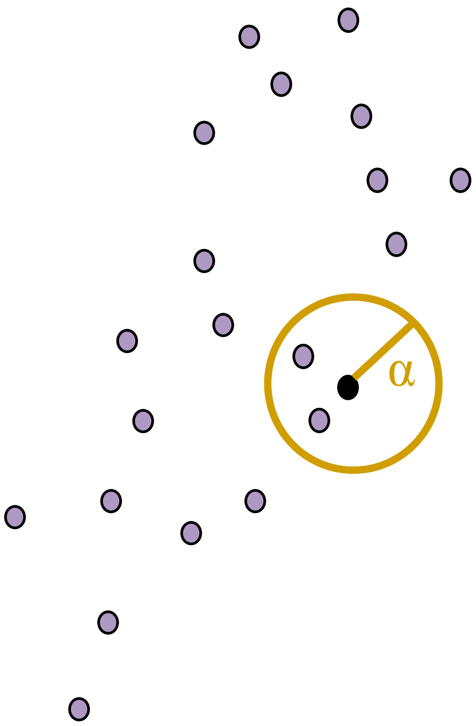
A Running Example: Fish Schools

- Adapted from Couzin et al., Nature 2005
- Fish Behavior
  - Avoidance: if too close, repel other fish
  - Attraction: if seen within range, attract other fish



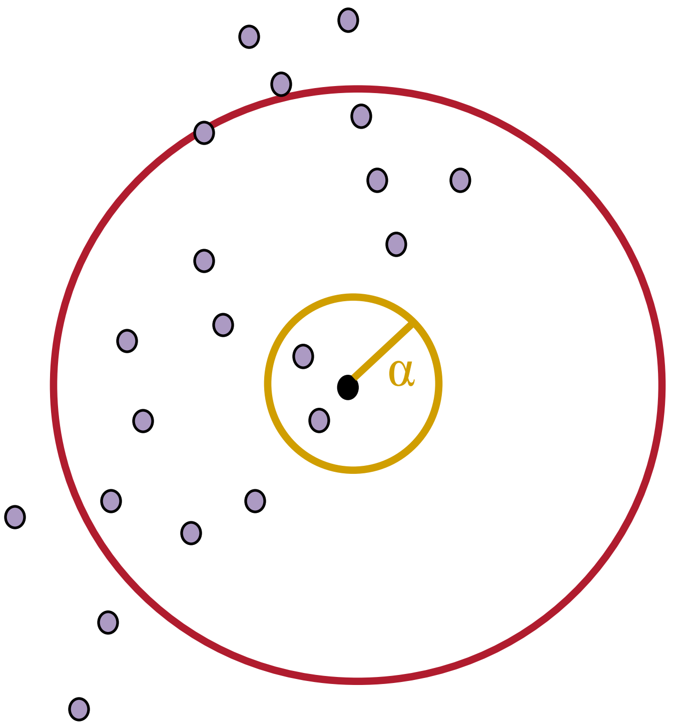
A Running Example: Fish Schools

- Adapted from Couzin et al., Nature 2005
- Fish Behavior
  - Avoidance: if too close, repel other fish
  - Attraction: if seen within range, attract other fish



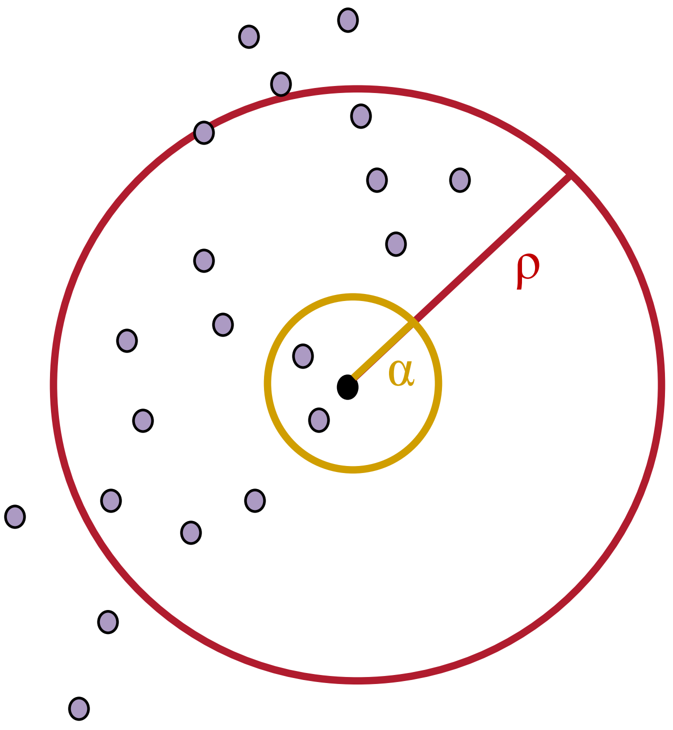
A Running Example: Fish Schools

- Adapted from Couzin et al., Nature 2005
- Fish Behavior
  - Avoidance: if too close, repel other fish
  - Attraction: if seen within range, attract other fish



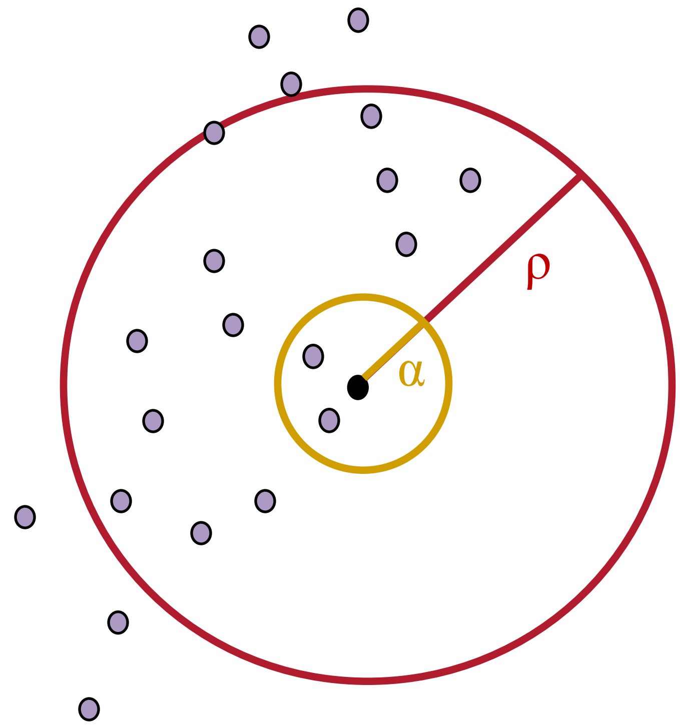
A Running Example: Fish Schools

- Adapted from Couzin et al., Nature 2005
- Fish Behavior
  - Avoidance: if too close, repel other fish
  - Attraction: if seen within range, attract other fish



A Running Example: Fish Schools

- *Time-stepping*: agents proceed in ticks
- *Concurrency*: agents are concurrent within a tick
- *Interactions*: agents continuously interact
- *Spatial Locality*: agents have limited visibility



Classic Solutions for Concurrency

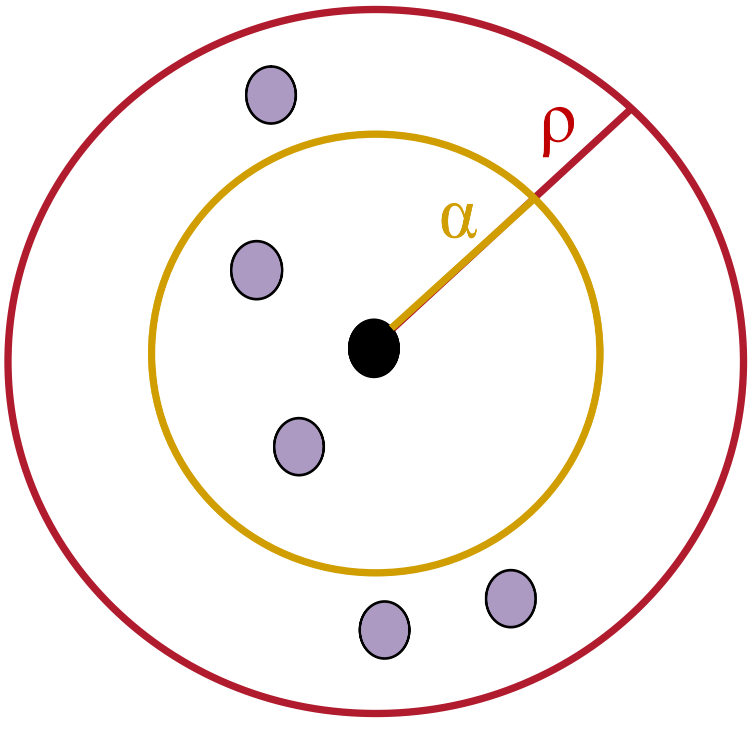
- Preempt conflicts → locking
- Rollback in case of conflicts → optimistic concurrency control
- Problems:
  - Strong iterations → many conflicts
    - Either lots of lock contention
    - Or lots of rollbacks
  - Does not scale well

State-Effect Pattern

- Programming pattern to deal with concurrency
- Follows time-stepped model
- **Core Idea:** Make all actions inside of a tick *order-independent*

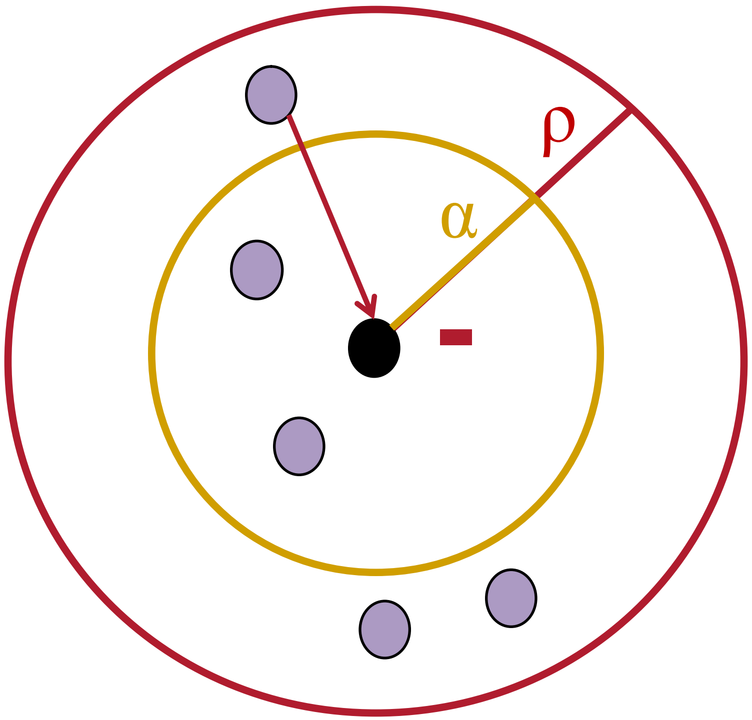
States and Effects

- States:
  - Snapshot of agents at the beginning of the tick
    - position, velocity vector
- Effects:
  - Intermediate results from interaction, used to calculate new states
    - sets of forces from other fish



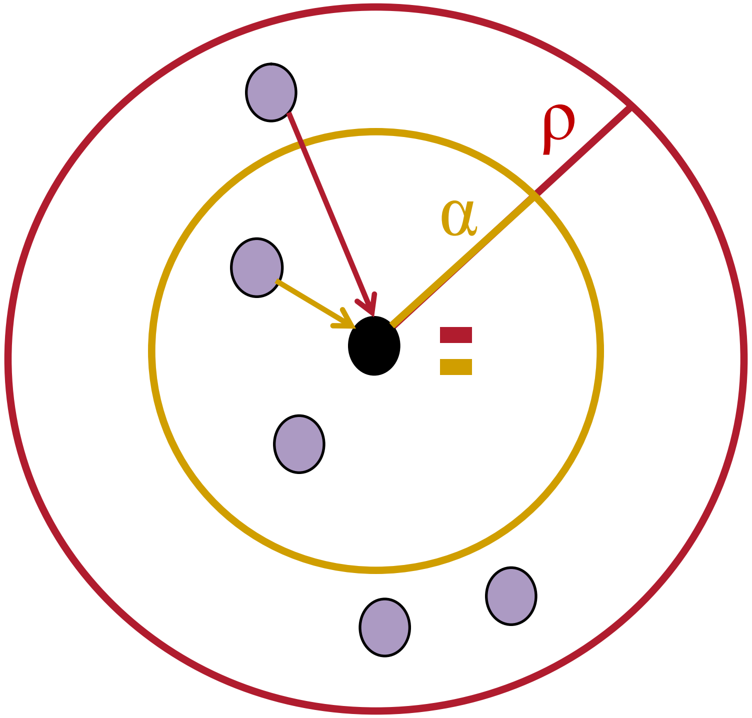
States and Effects

- States:
  - Snapshot of agents at the beginning of the tick
    - position, velocity vector
- Effects:
  - Intermediate results from interaction, used to calculate new states
    - sets of forces from other fish



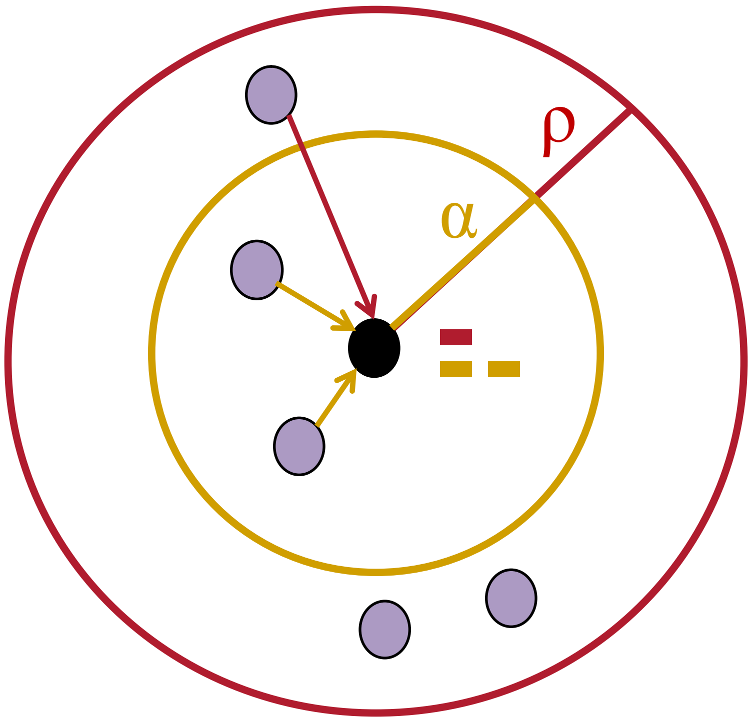
States and Effects

- States:
  - Snapshot of agents at the beginning of the tick
    - position, velocity vector
- Effects:
  - Intermediate results from interaction, used to calculate new states
    - sets of forces from other fish



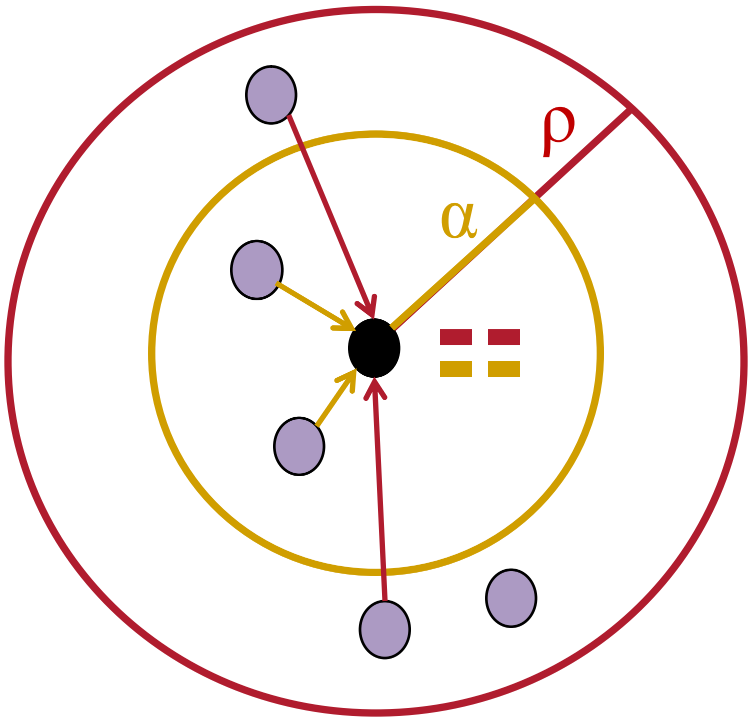
States and Effects

- States:
  - Snapshot of agents at the beginning of the tick
    - position, velocity vector
- Effects:
  - Intermediate results from interaction, used to calculate new states
    - sets of forces from other fish



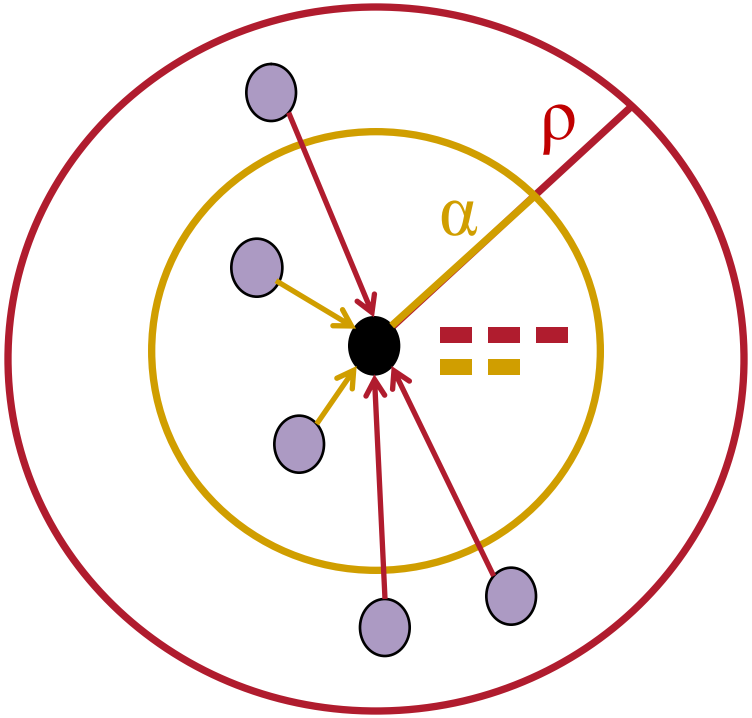
States and Effects

- States:
  - Snapshot of agents at the beginning of the tick
    - position, velocity vector
- Effects:
  - Intermediate results from interaction, used to calculate new states
    - sets of forces from other fish



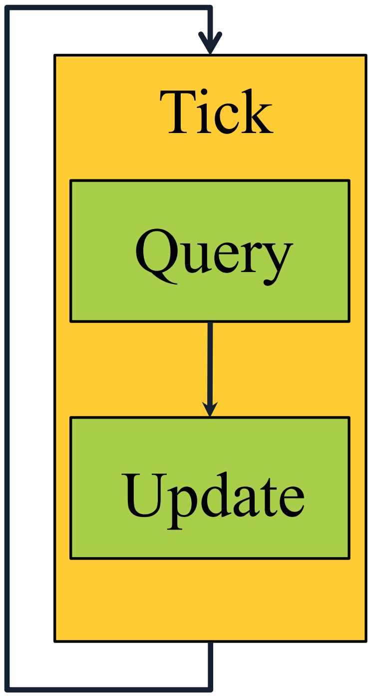
States and Effects

- States:
  - Snapshot of agents at the beginning of the tick
    - position, velocity vector
- Effects:
  - Intermediate results from interaction, used to calculate new states
    - sets of forces from other fish



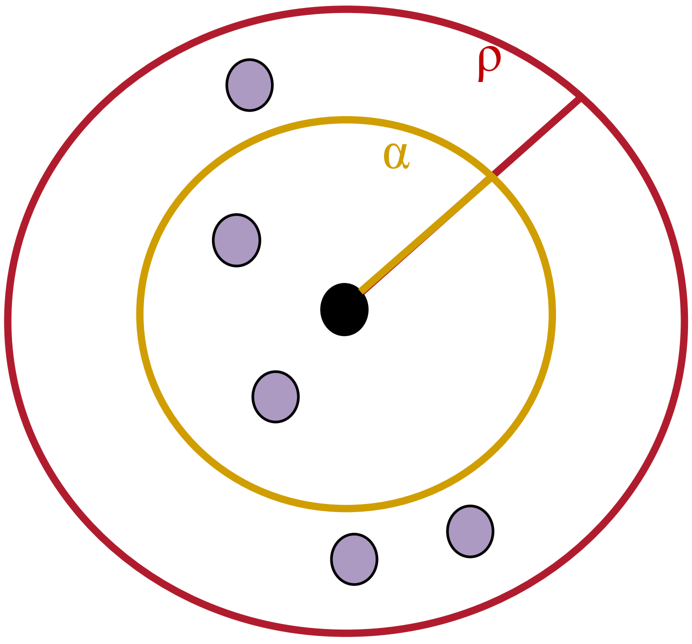
Two Phases of a Tick

- Query: capture agent interaction
  - Read states → write effects
  - Each effect set is associated with *combinator* function
  - Effect writes are *order-independent*
- Update: refresh world for next tick
  - Read effects → write states
  - Reads and writes are totally local
  - State writes are *order-independent*



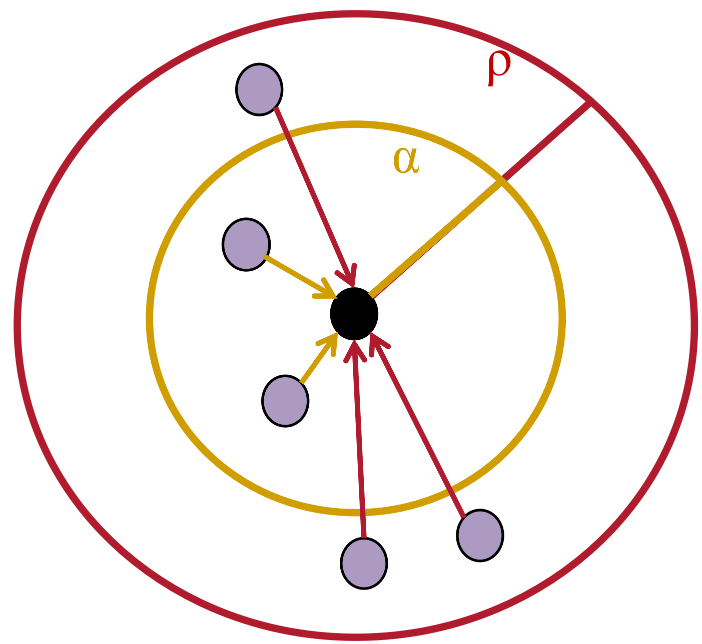
A Tick in State-Effect

- Query
  - For fish f in visibility α:
    - Write repulsion to f’s effects
  - For fish f in visibility ρ:
    - Write attraction to f’s effects
- Update
  - new velocity = combined repulsion + combined attraction + old velocity
  - new position = old position + old velocity



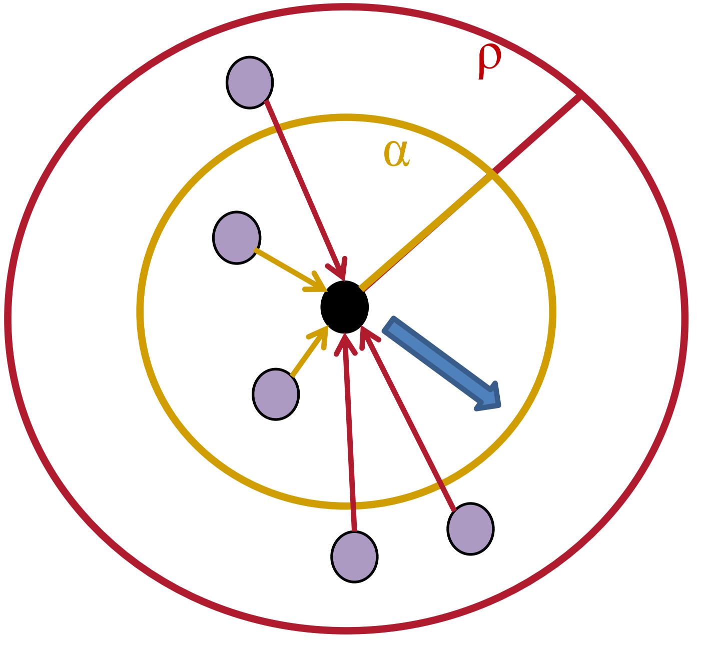
A Tick in State-Effect

- Query
  - For fish f in visibility α:
    - Write repulsion to f's effects
  - For fish f in visibility ρ:
    - Write attraction to f's effects
- Update
  - new velocity = combined repulsion + combined attraction + old velocity
  - new position = old position + old velocity



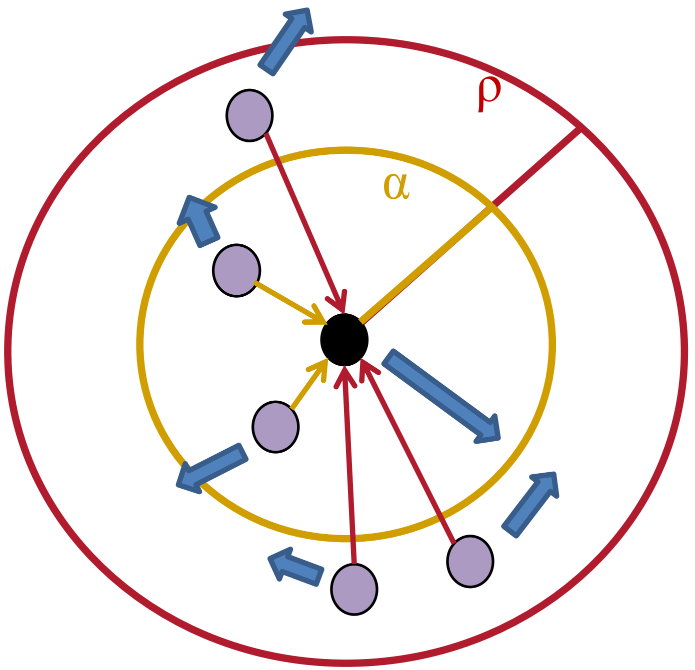
A Tick in State-Effect

- Query
  - For fish f in visibility α:
    - Write repulsion to f's effects
  - For fish f in visibility ρ:
    - Write attraction to f's effects
- Update
  - new velocity = combined repulsion + combined attraction + old velocity
  - new position = old position + old velocity



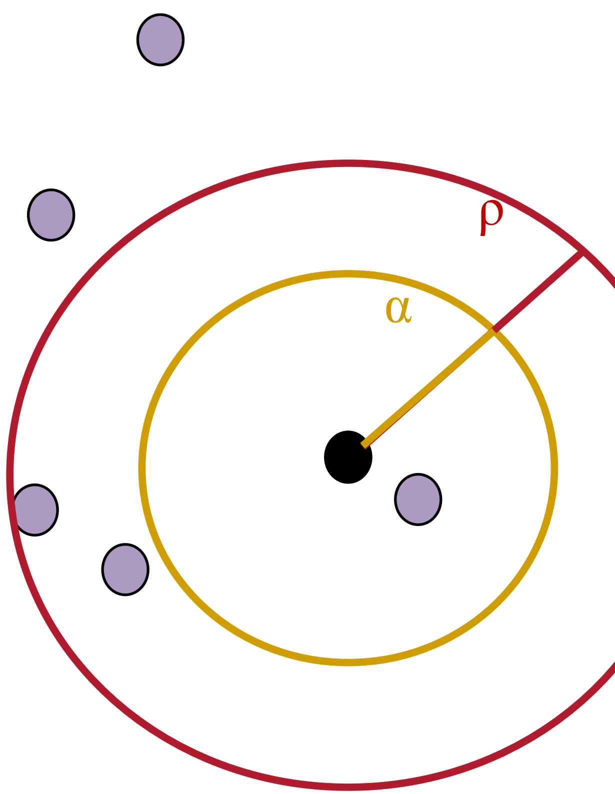
A Tick in State-Effect

- Query
  - For fish f in visibility α:
    - Write repulsion to f's effects
  - For fish f in visibility ρ:
    - Write attraction to f's effects
- Update
  - new velocity = combined repulsion + combined attraction + old velocity
  - new position = old position + old velocity



Fish in State-Effect

- Query
  - For fish f in visibility α:
    - Write repulsion to f's effects
  - For fish f in visibility ρ:
    - Write attraction to f's effects
- Update
  - new velocity = combined repulsion + combined attraction + old velocity
  - new position = old position + old velocity



BRASIL (**B**ig **R**ed **A**gent **SI**mulation **L**anguage)

- High-level language for domain scientists
- Object-oriented style
- Programs specify behavior logic of individual agents

Fish in BRASIL

```
class Fish {
  // The fish location & velocity (x)
  public state float x  : x + vx; #range[-1,1];
  public state float vx : vx + rand() + avoidx / count * vx;
  // Used to update our velocity (x)
  private effect float avoidx : sum;
  private effect int   count  : sum;
  /** The query-phase for this fish. */
  public void run() {
    // Use "forces" to repel fish too close
    foreach(Fish p : Extent<Fish>) {
      p.avoidx <- 1 / abs(x - p.x);
      ...
      p.count  <- 1;
    }}}
```

Fish in BRASIL

- Syntax enforces state-effect pattern
- Translates to Monad Algebra
  - Can reuse classic DB optimization techniques

```
public void run() {
  // Use "forces" to repel fish too close
  foreach(Fish p : Extent<Fish>) {
     p.avoidx <- 1 / abs(x - p.x);
     ...
     p.count  <- 1;
  }}}
```

Fish in BRASIL

- Syntax enforces state-effect pattern
- Translates to Monad Algebra
  - Can reuse classic DB optimization techniques

$$\mathrm{P} = \langle 1{:}\Pi_1 \circ \Pi_p, 2{:}\Pi_2\rangle \circ \mathrm{PAIRWITH}_2 \circ \sigma_{\Pi_1=\Pi_2 \circ \Pi_{key}} \circ \mathrm{GET} \circ \Pi_x$$

$$\mathrm{E}_1 = \langle 1{:}\Pi_1 \circ \Pi_p, 2{:}\rho(\mathrm{avoidx}), 3{:}1 \;/\; (\Pi_1 \circ \Pi_x - \mathrm{P})\rangle$$

$$\mathrm{E}_2 = \langle 1{:}\Pi_1 \circ \Pi_p, 2{:}\rho(\mathrm{count}), 3{:}1\rangle$$

$$\mathrm{B} = \langle 1{:}\Pi_1, 2{:}\Pi_2, 3{:}\Pi_2 \oplus (\mathrm{E}_1 \circ \mathrm{SNG}) \oplus (\mathrm{E}_2 \circ \mathrm{SNG})\rangle$$

$$\mathrm{F} = \langle 1{:}\Pi_1, 2{:}\Pi_2, 3{:}\langle 1{:}\Pi_1 \circ x_p(\Pi_2) \circ \mathrm{PAIRWITH}_p, \; 2{:}\Pi_2, \; 3{:}\Pi_3\rangle \circ \mathrm{FLATMAP}(\mathrm{B} \circ \Pi_3)\rangle$$

- Details of translation in our VLDB 2010 paper

Talk Outline

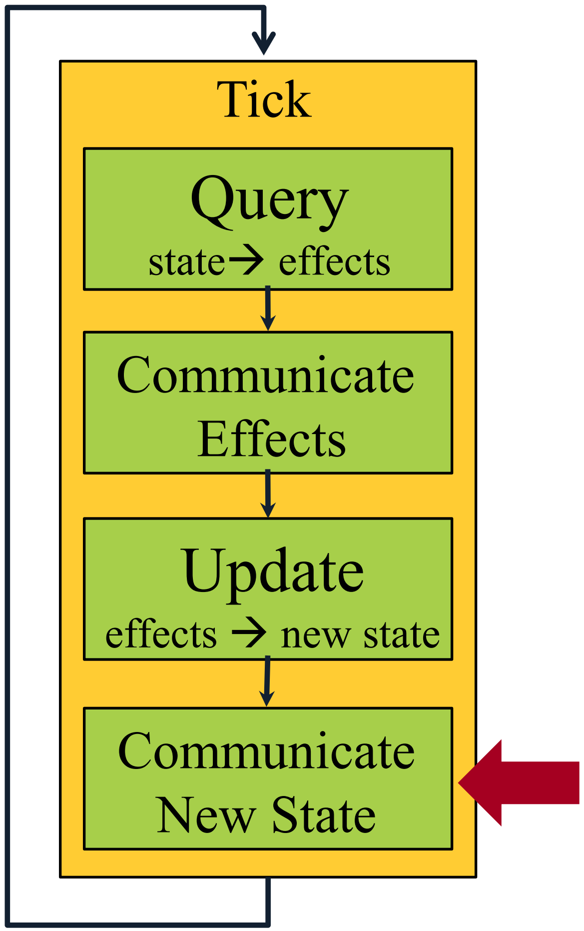
- Motivation
- Ease of Programming
  - Program Simulations in State-Effect Pattern
  - BRASIL
- Scalability
  - Execute Simulations in MapReduce Model
  - BRACE
- Experiments
- Conclusion

How to Scale to Millions of Fish?

- Use multiple nodes in a cluster of machines for large simulation scenarios
- Need to efficiently parallelize computations of state-effect pattern

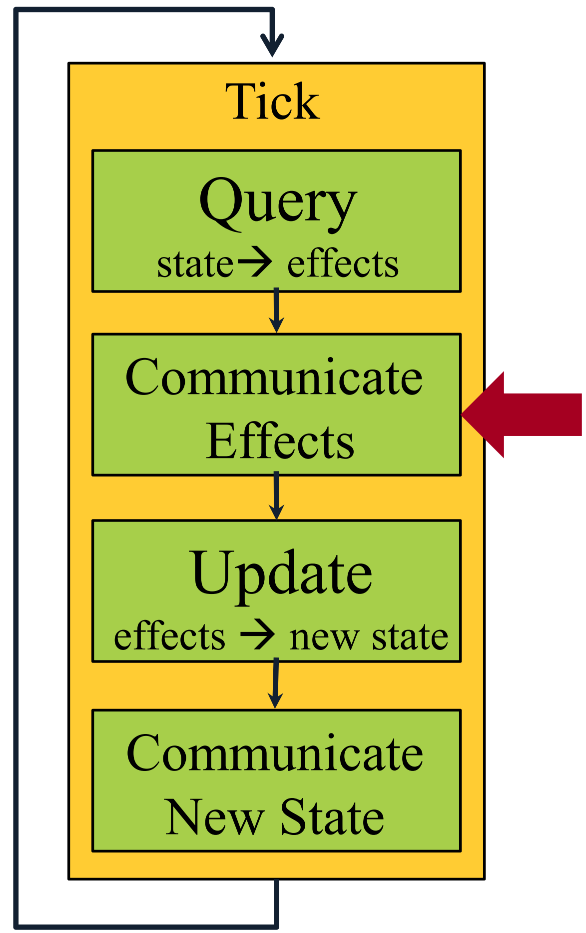
State-Effect Revisited

- Agent partitioning with replications across nodes
- Communicate new states before next tick's query phase

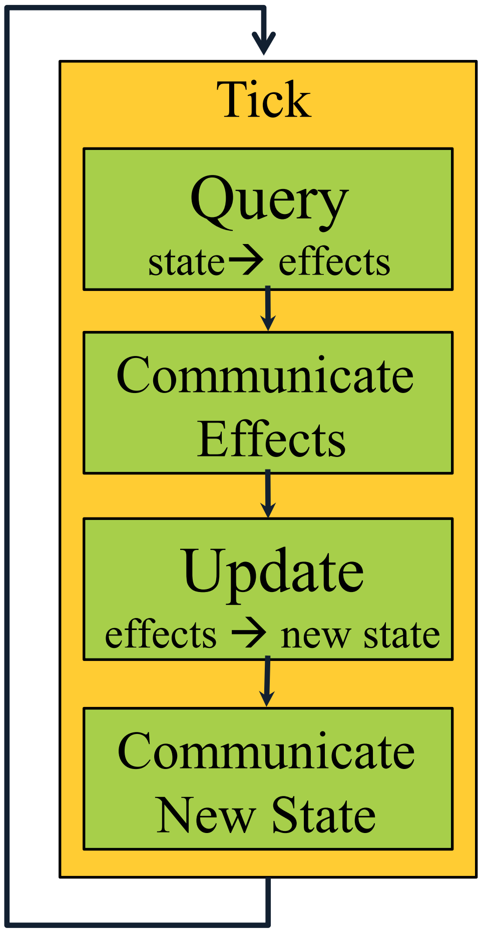


State-Effect Revisited

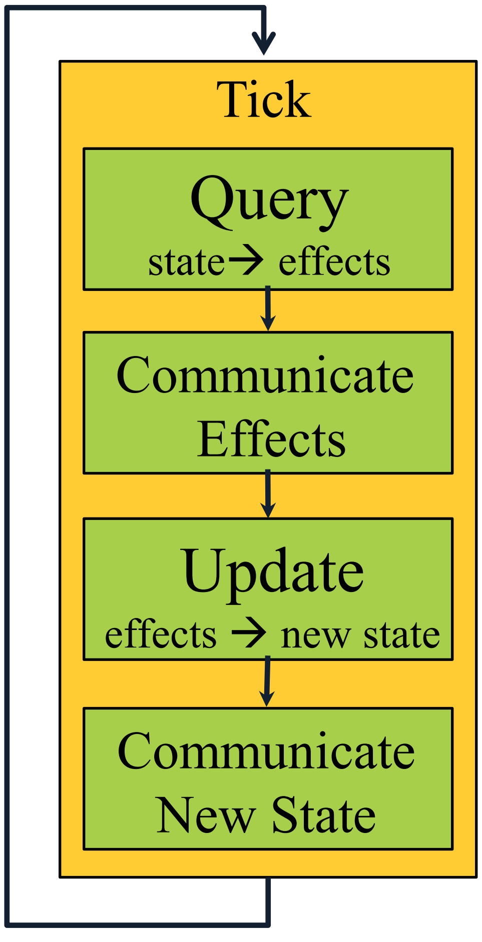
- Agent partitioning with replications across nodes
- Communicate new states before next tick's query phase
- Communicate effect assignments before update phase



From State-Effect to Map-Reduce



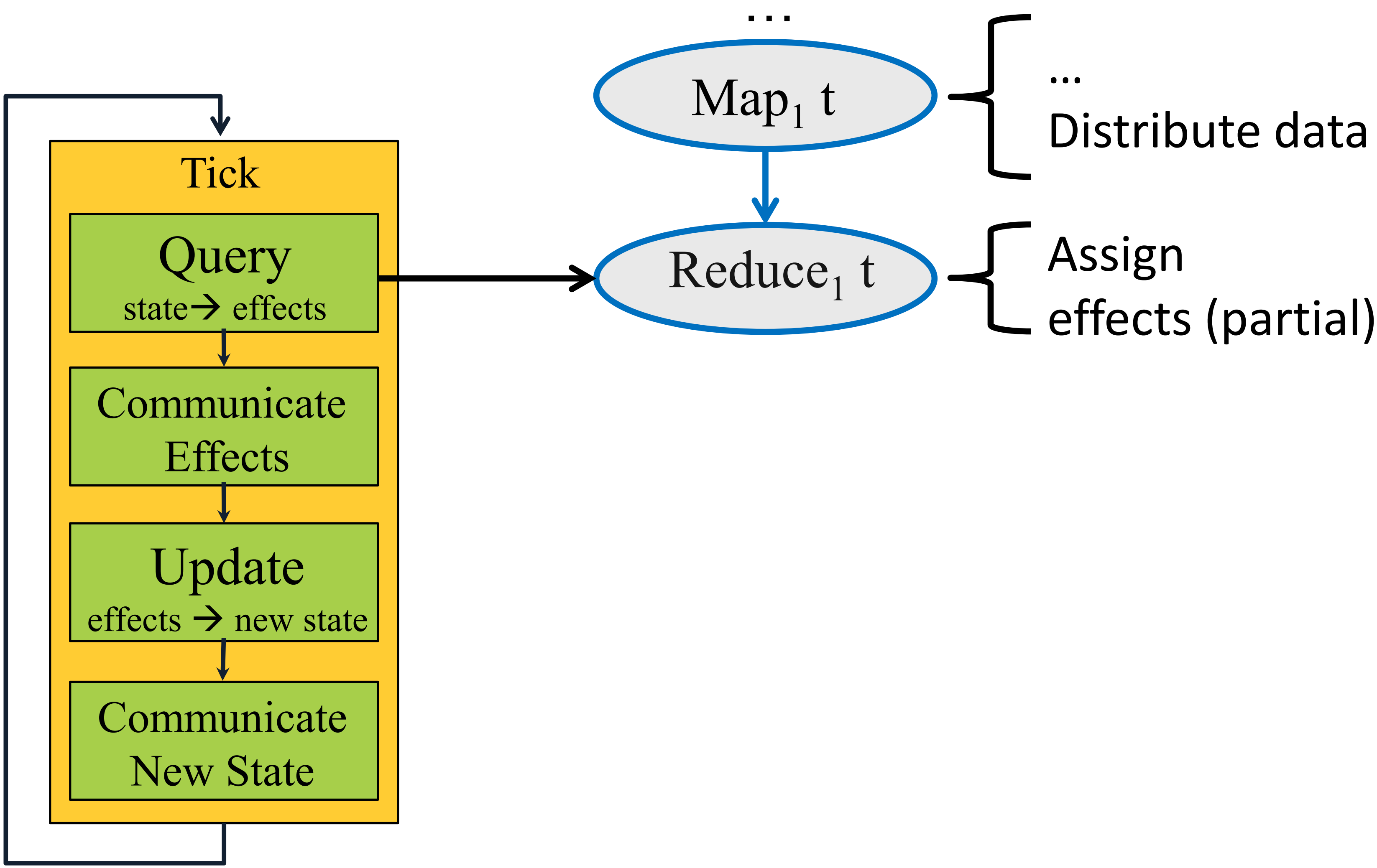
From State-Effect to Map-Reduce



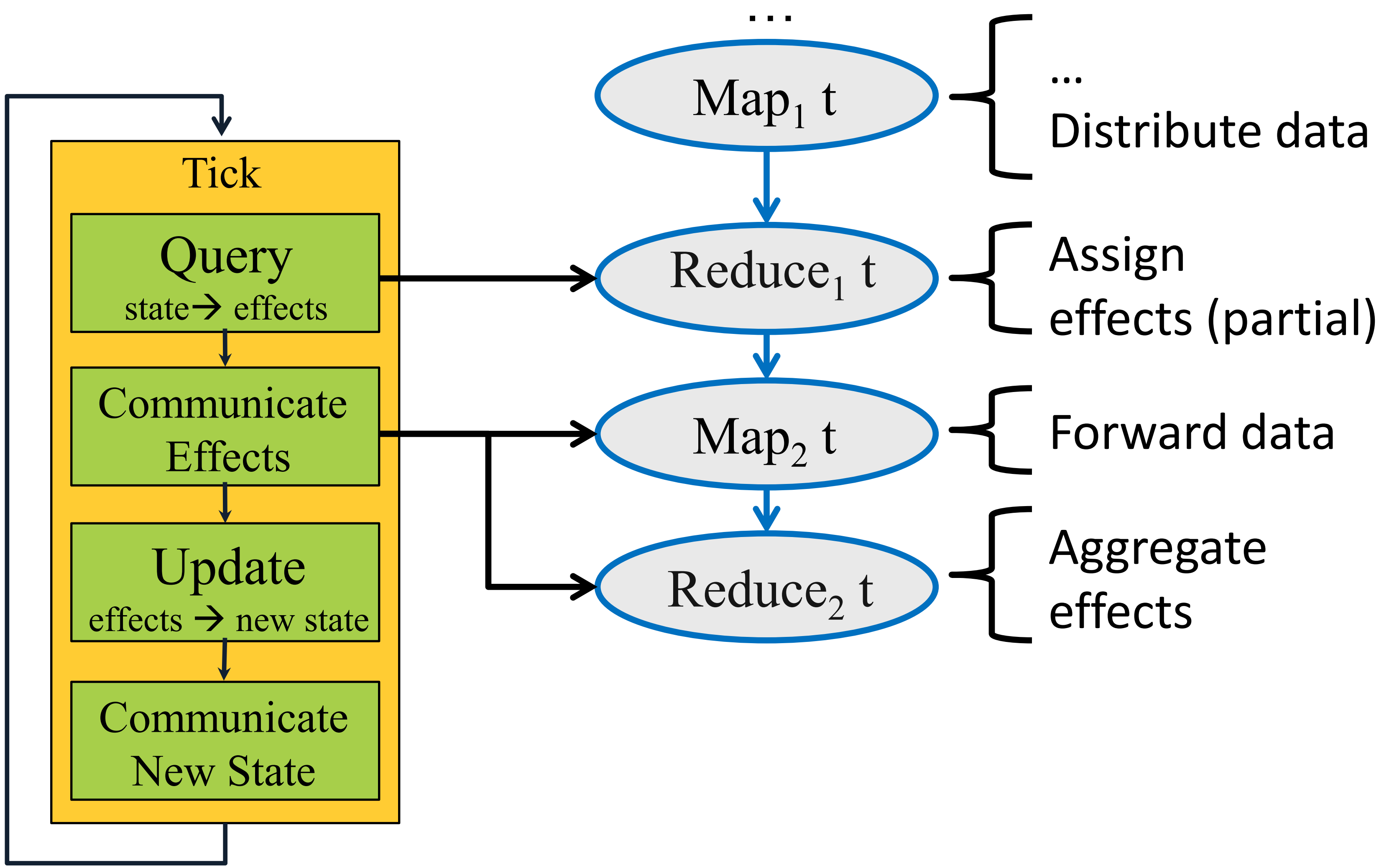
Map_1 t

- …
- Distribute data

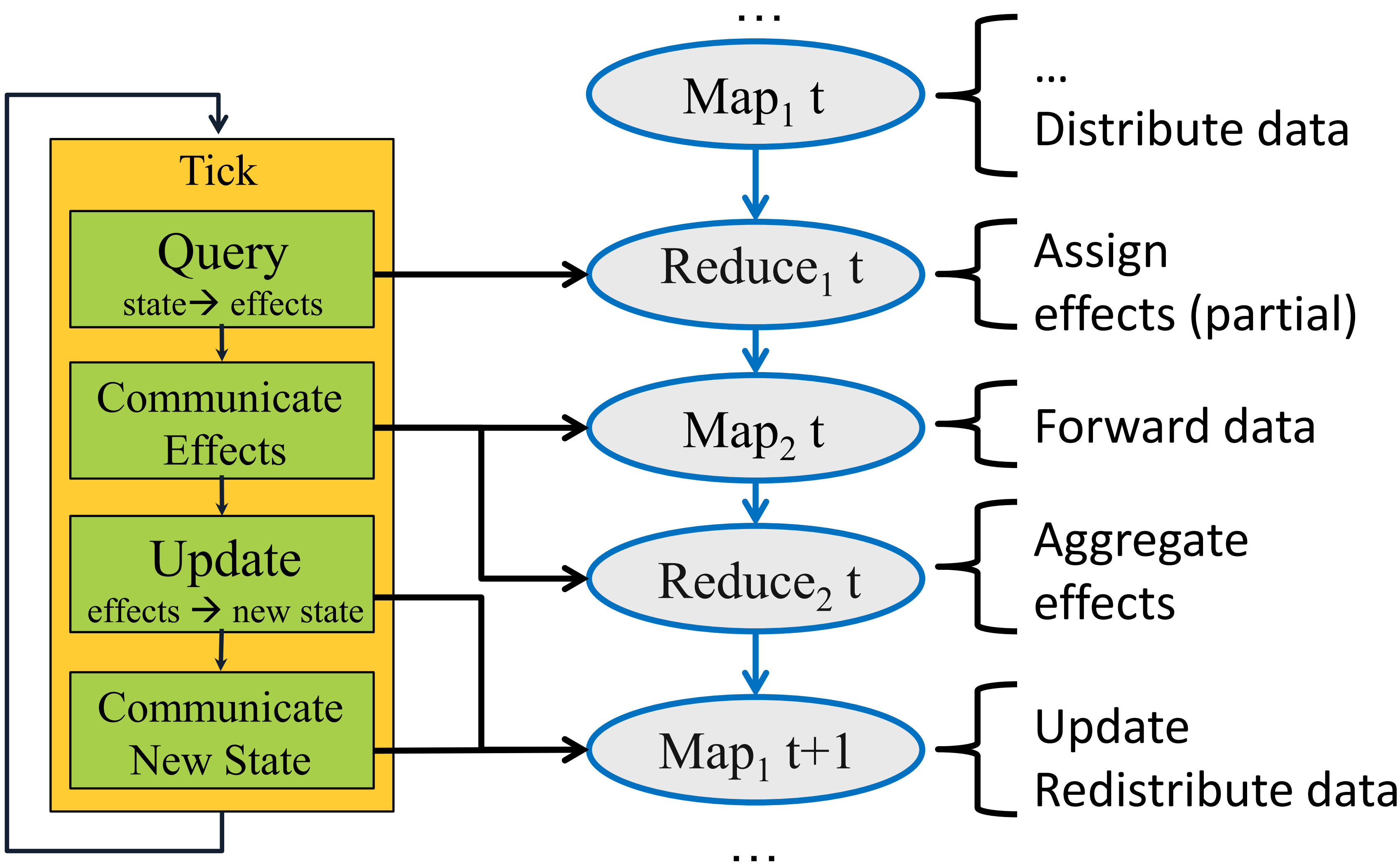
From State-Effect to Map-Reduce



From State-Effect to Map-Reduce



From State-Effect to Map-Reduce

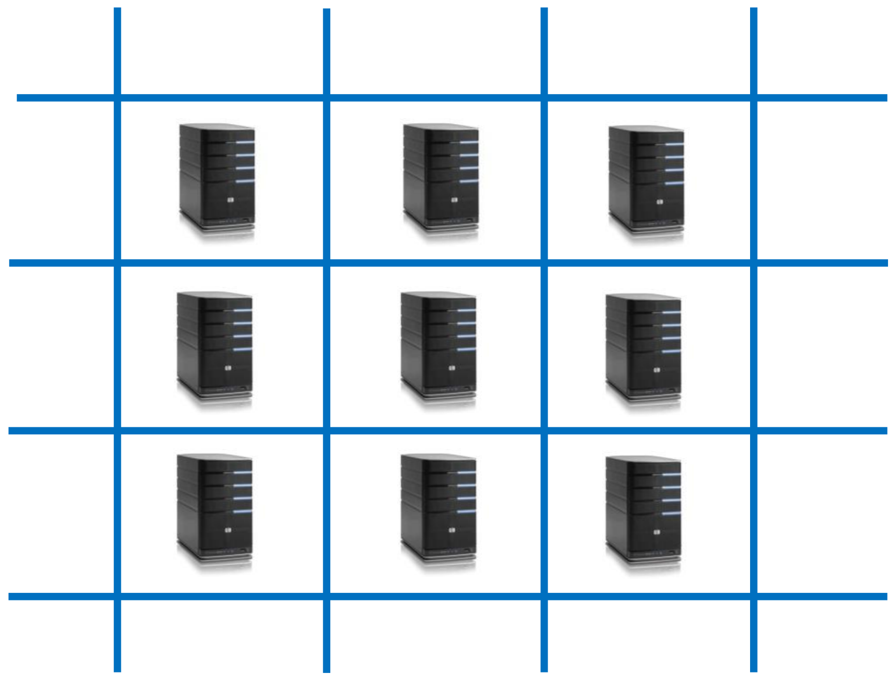


BRACE (**B**ig **R**ed **A**gent **C**omputation **E**ngine)

- Special-purpose MapReduce engine for behavioral simulations
- Basic Optimizations
  - Keep data in main memory
  - Do Not checkpoint every iteration
- Optimizations based on *Spatial Properties:*
  - Collocate tasks
  - Minimize communication overhead

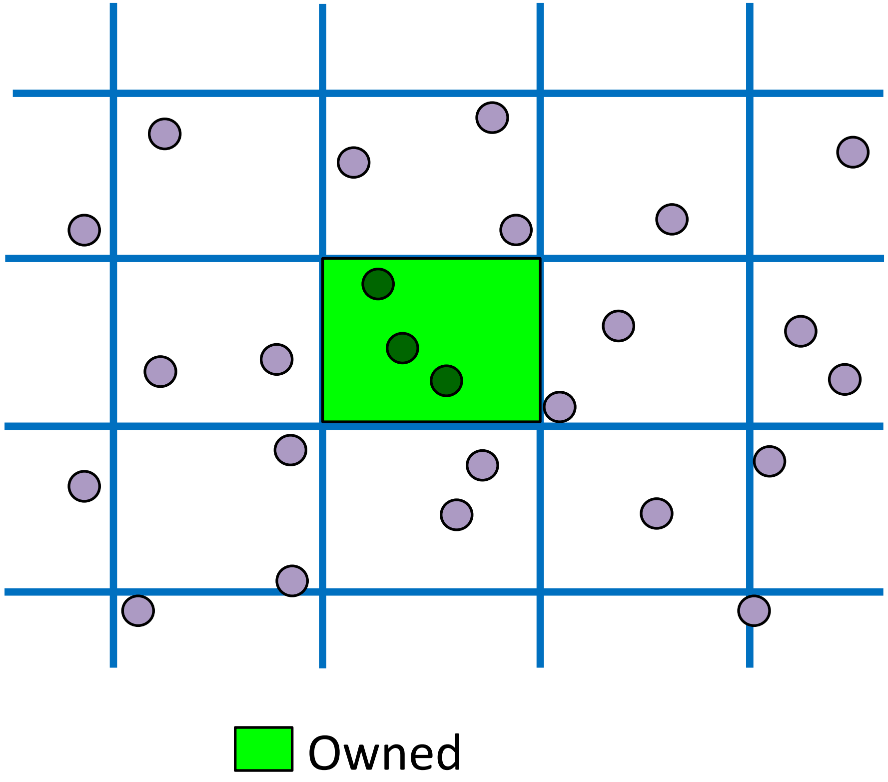
Spatial Partitioning

- Partition simulation space into regions, each handled by a separate node



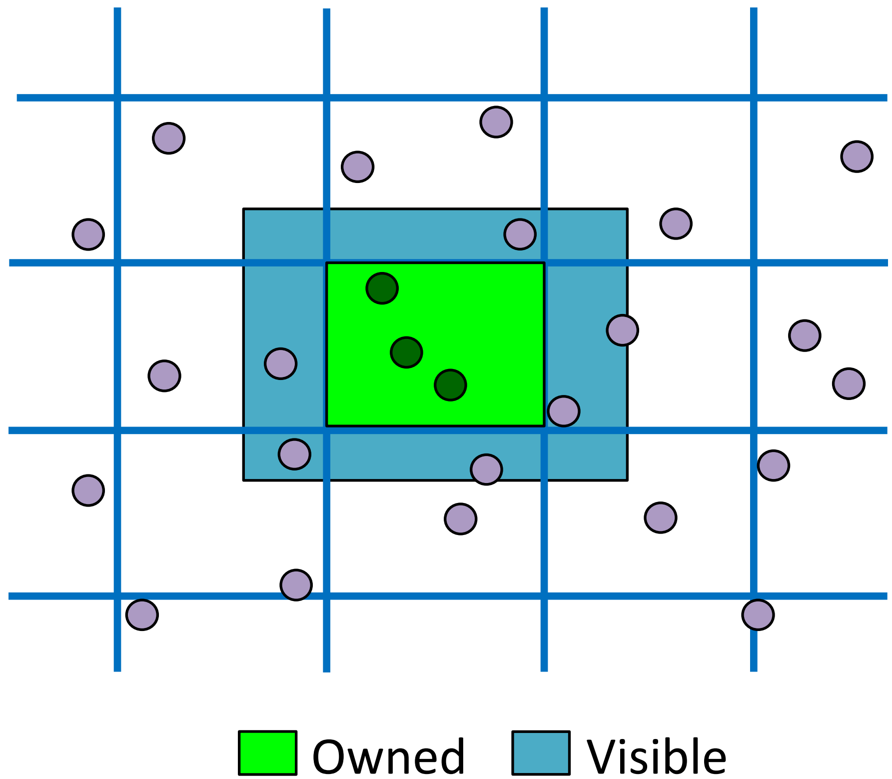
Communication Between Partitions

- *Owned Region*: agents in it are owned by the node



Communication Between Partitions

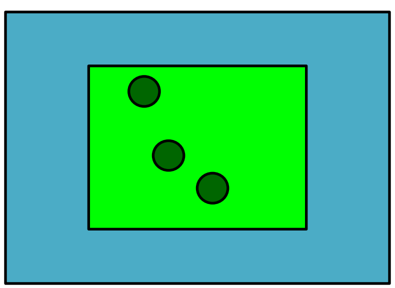
- *Visible Region*: agents in it are not owned, but need to be seen by the node

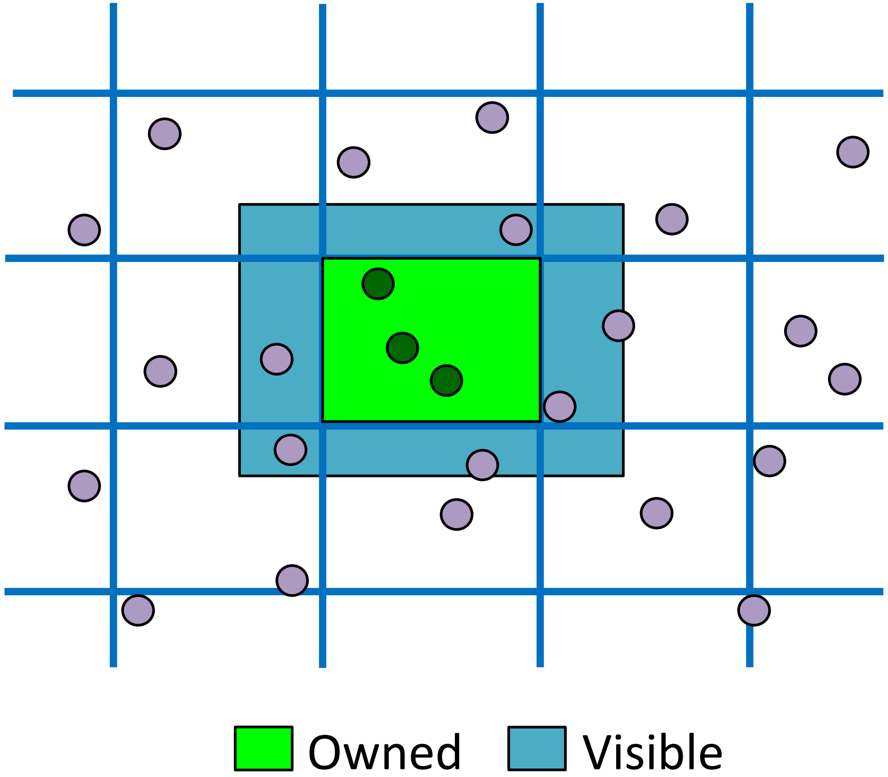


Communication Between Partitions

- *Visible Region*: agents in it are not owned, but need to be seen by the node

State Communication

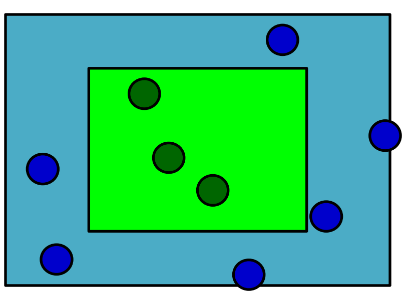


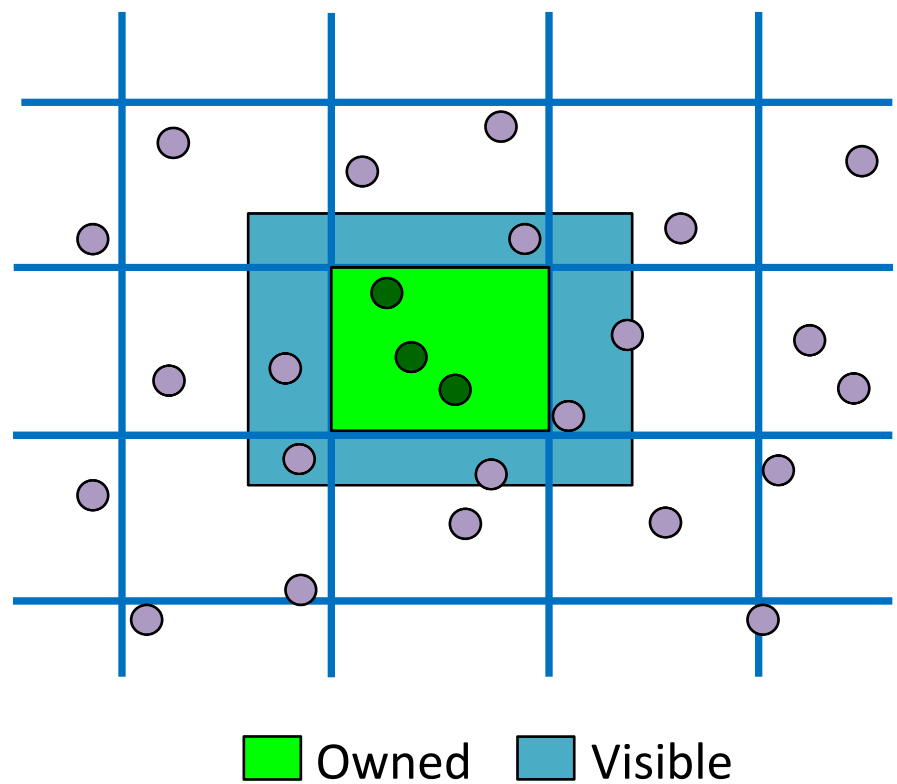


Communication Between Partitions

- *Visible Region*: agents in it are not owned, but need to be seen by the node

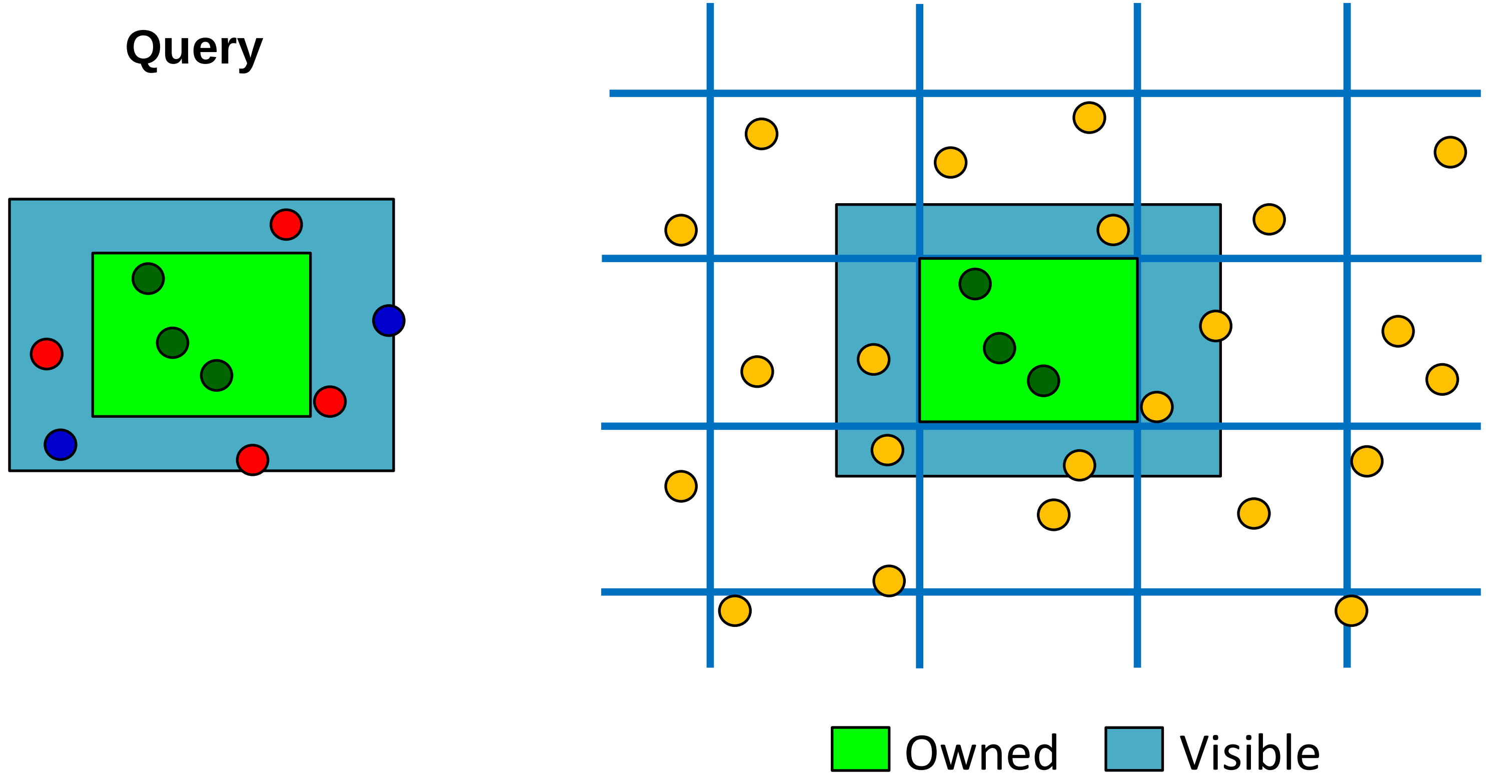
State Communication





Communication Between Partitions

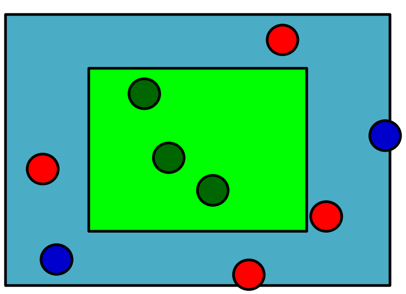
- *Visible Region*: agents in it are not owned, but need to be seen by the node

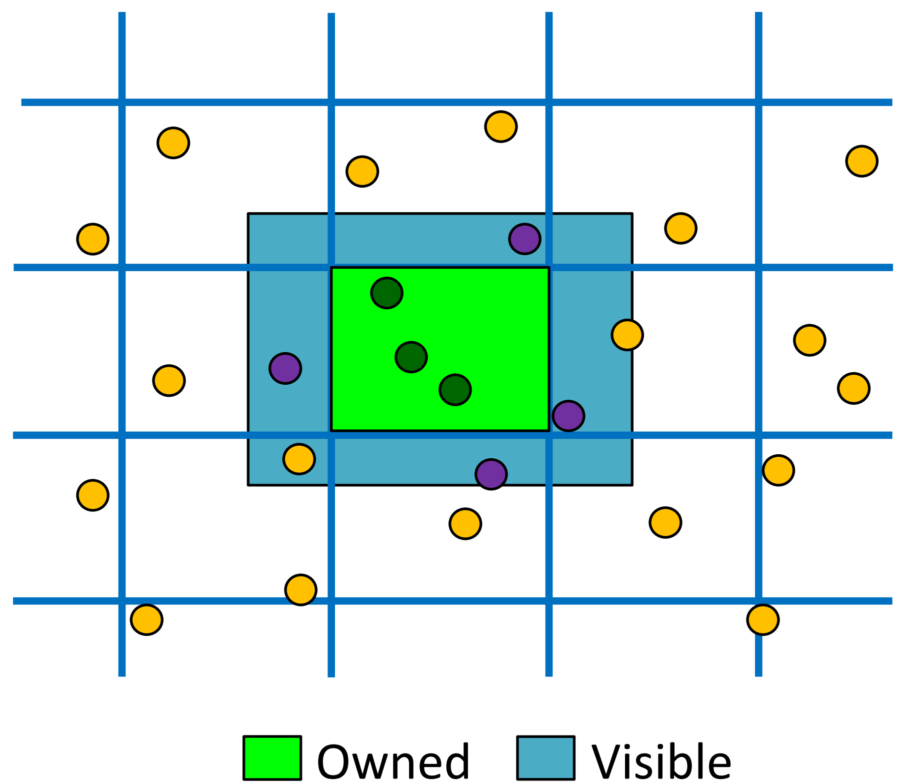


Communication Between Partitions

- *Visible Region*: agents in it are not owned, but need to be seen by the node

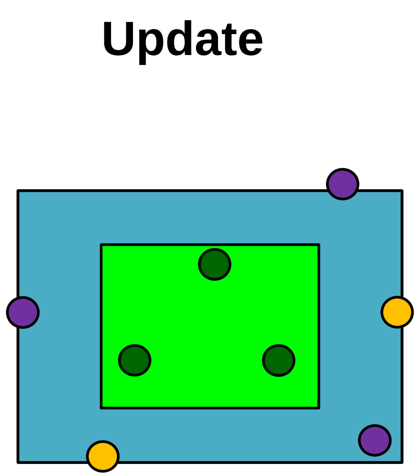
Effect communication

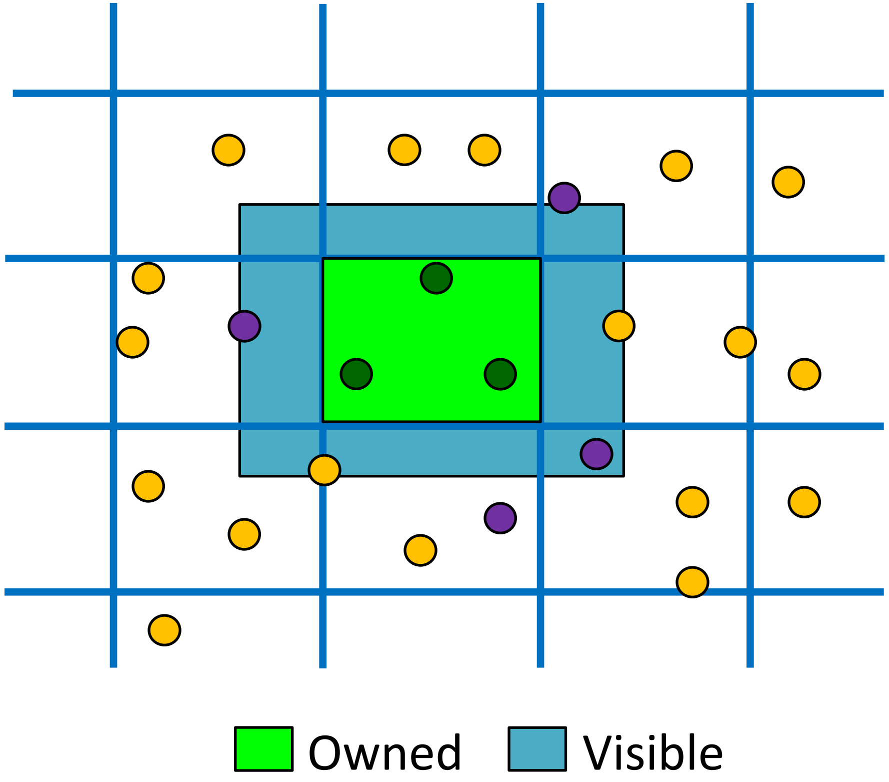




Communication Between Partitions

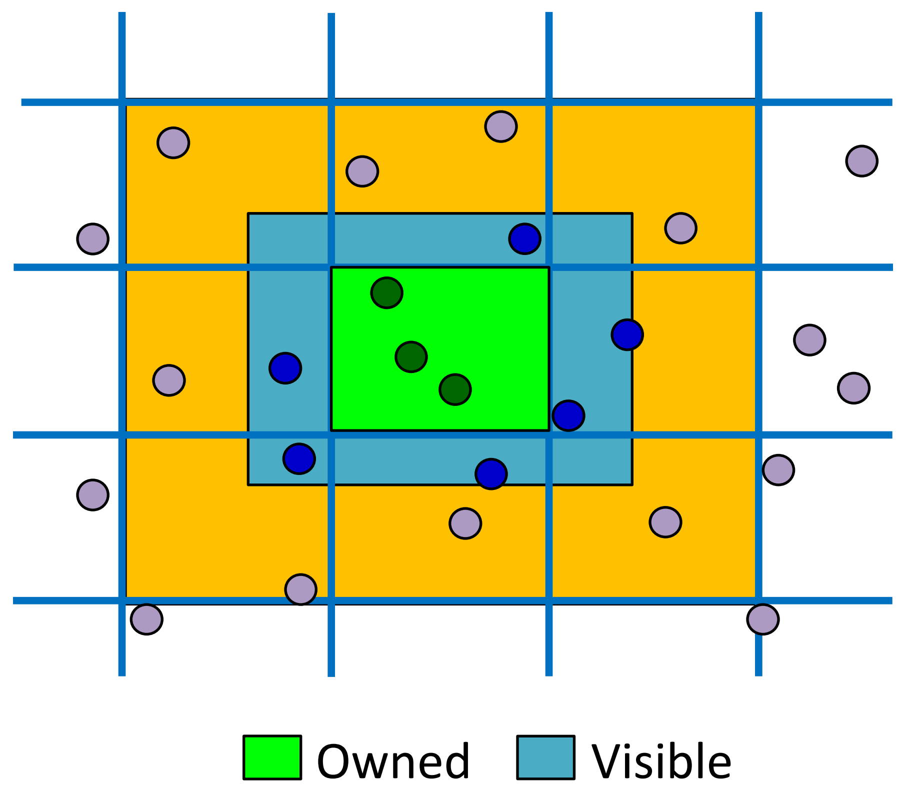
- *Visible Region*: agents in it are not owned, but need to be seen by the node





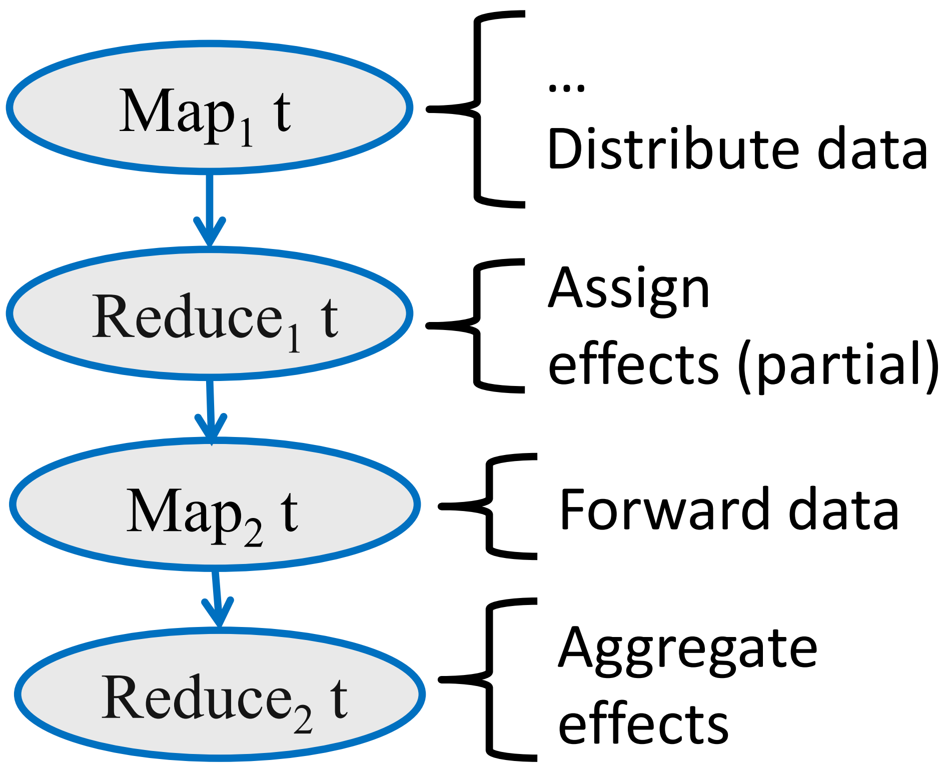
Communication Between Partitions

- *Visible Region*: agents in it are not owned, but need to be seen by the node
- Only need to communicate with neighbors to
  – refresh states
  – forward assigned effects



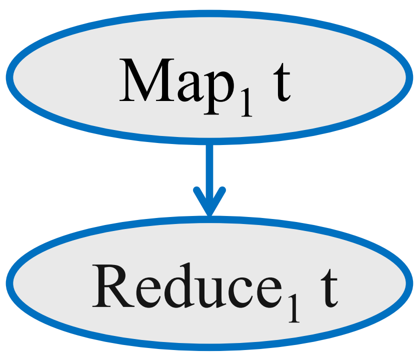
Effect Inversion

- In case of **local** effects only, can save one round of communication in each tick



Effect Inversion

- In case of **local** effects only, can save one round of communication in each tick



…
Distribute data

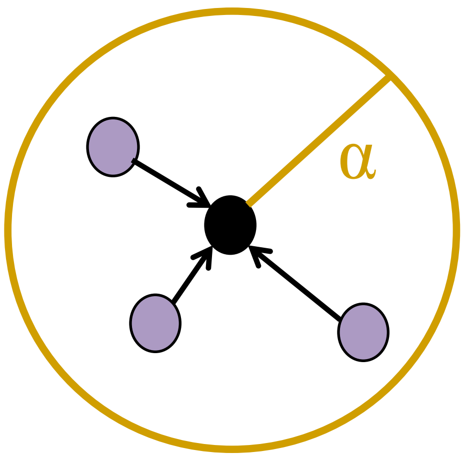
Assign and
Aggregate effects

Do not have non-local effects

Effect Inversion Is Always Possible

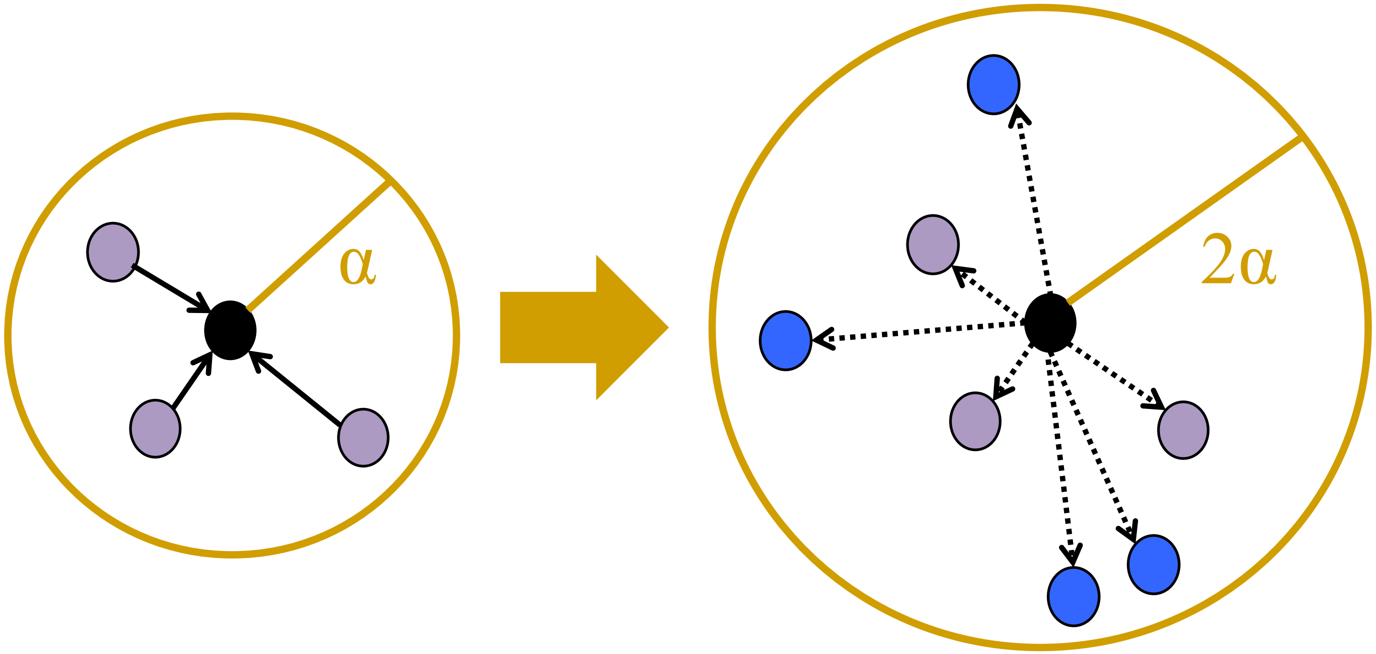
- **Theorem**: Every behavioral simulation written in BRASIL that uses non-local effects can be rewritten to an equivalent simulation that uses local effects only
  - Proof in the VLDB 2010 paper

Intuition of Effect Inversion Theorem



Non-local
Effect Writes

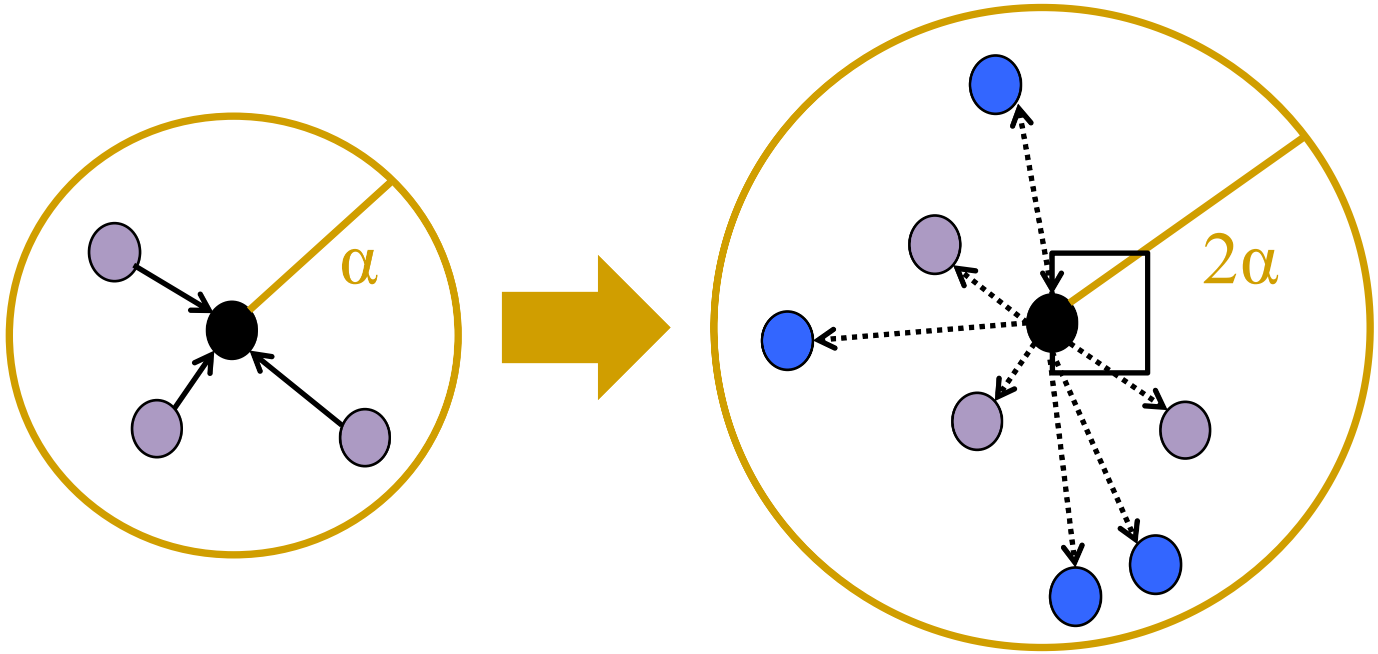
Intuition of Effect Inversion Theorem



Non-local
Effect Writes

Non-local
State Reads

Intuition of Effect Inversion Theorem



Non-local
Effect Writes

Non-local
State Reads
+
Local
Effect Writes

Talk Outline

- Motivation
- Ease of Programming
  - Program Simulations in Time-stepped Pattern
  - BRASIL
- Scalability
  - Execute Simulations in Dataflow Model
  - BRACE
- Experiments
- Conclusion

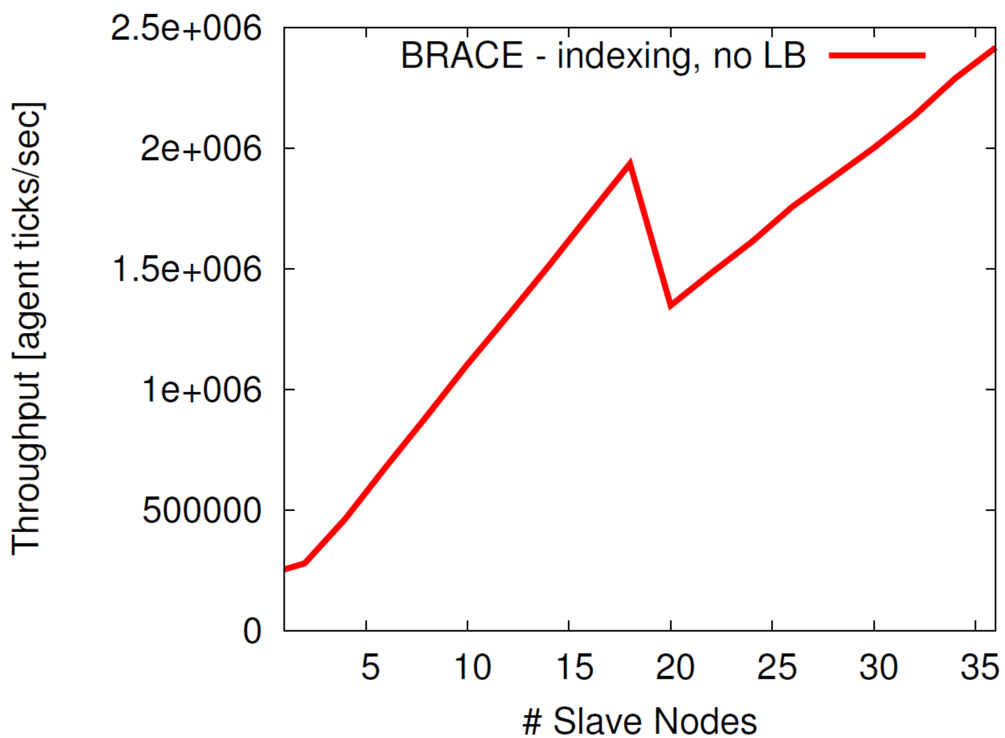
Experimental Setup

- BRACE prototype
  - Grid partitioning
  - KD-Tree spatial indexing, rebuild every tick
  - Basic load balancing
  - No checkpointing
- Hardware: Cornell WebLab Cluster (60 nodes, 2xQuadCore Xeon 2.66GHz, 4MB cache, 16GB RAM)

Implemented Simulations

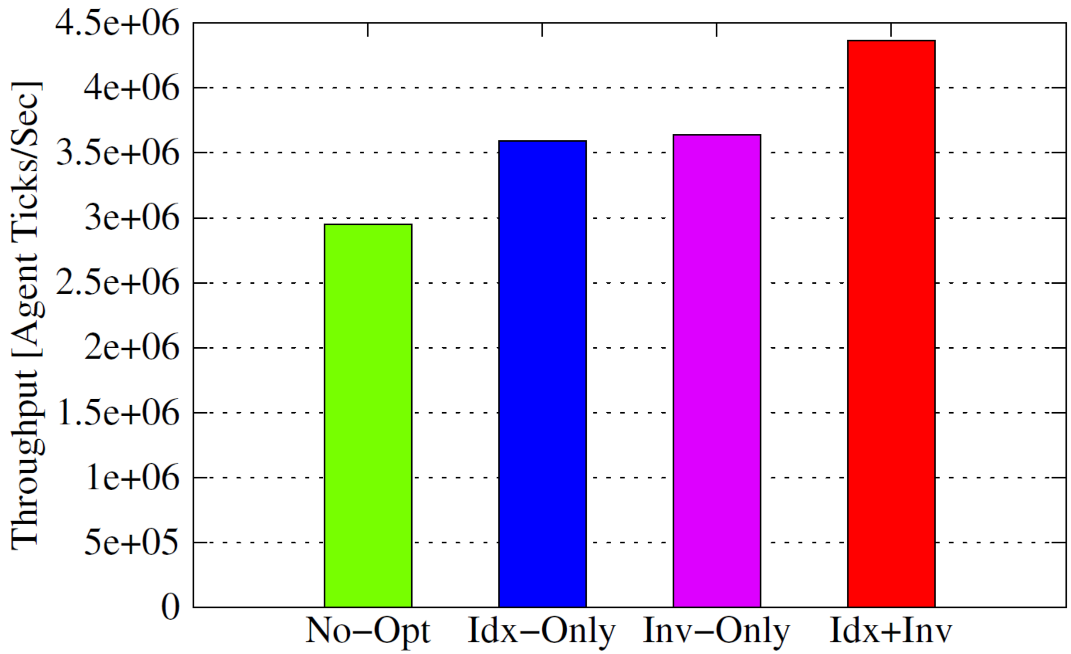
- Traffic Simulation
  - Best-effort reimplementation of MITSIM lane changing and car following
  - Large segment of highway
- Bacteria Simulation
  - Simple artificial society simulation
- Fish School Simulation
  - Model of collective animal motion by Couzin et al., Nature, 2005

Scalability: Traffic



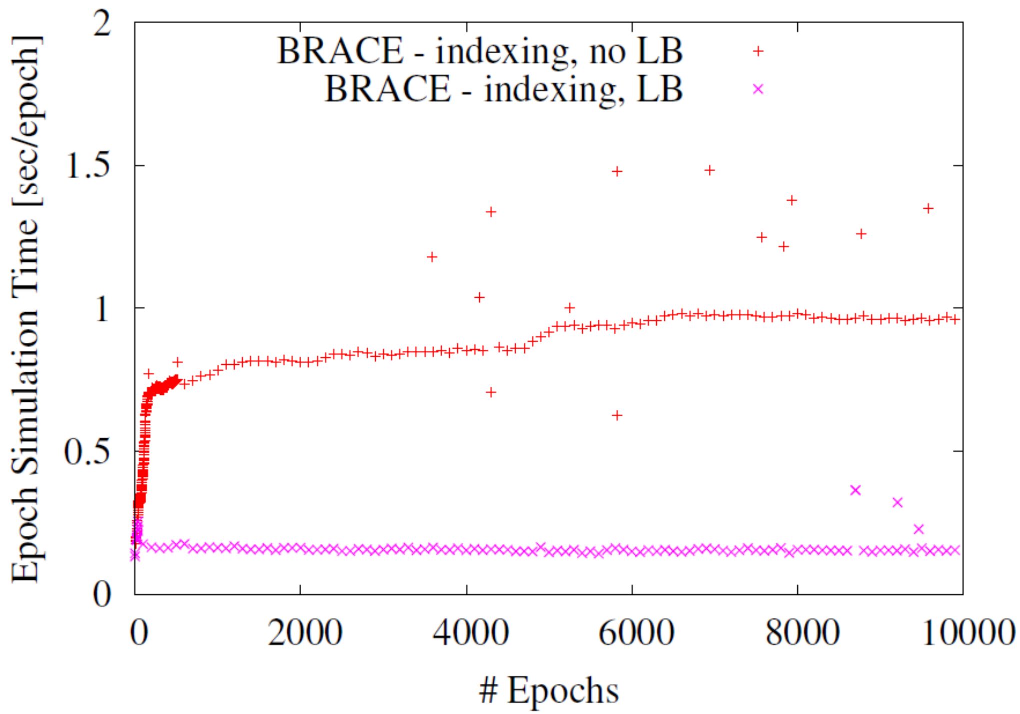
- Scale up the size of the highway with the number of the nodes
- Notch consequence of multi-switch architecture

Optimization: Bacteria



- 16-node with indexing and effect inversion
- 10,000 epochs of bacteria simulation

Load Balancing: Fish



- 16-node with load balancing turned on
- Fish simulation of two independent schools that swim in opposite directions

Conclusions

Thank you!

- Behavioral Simulations can have huge impact, but need to be run at large-scale
- New programming environment for behavioral simulations
  - *Easy to program:* Simulations in the state-effect pattern → BRASIL
  - *Scalable:* State-effect pattern in special-purpose MapReduce Engine → BRACE
- We are moving to simulate NYC ! ☺