CORNELL
UNIVERSITY

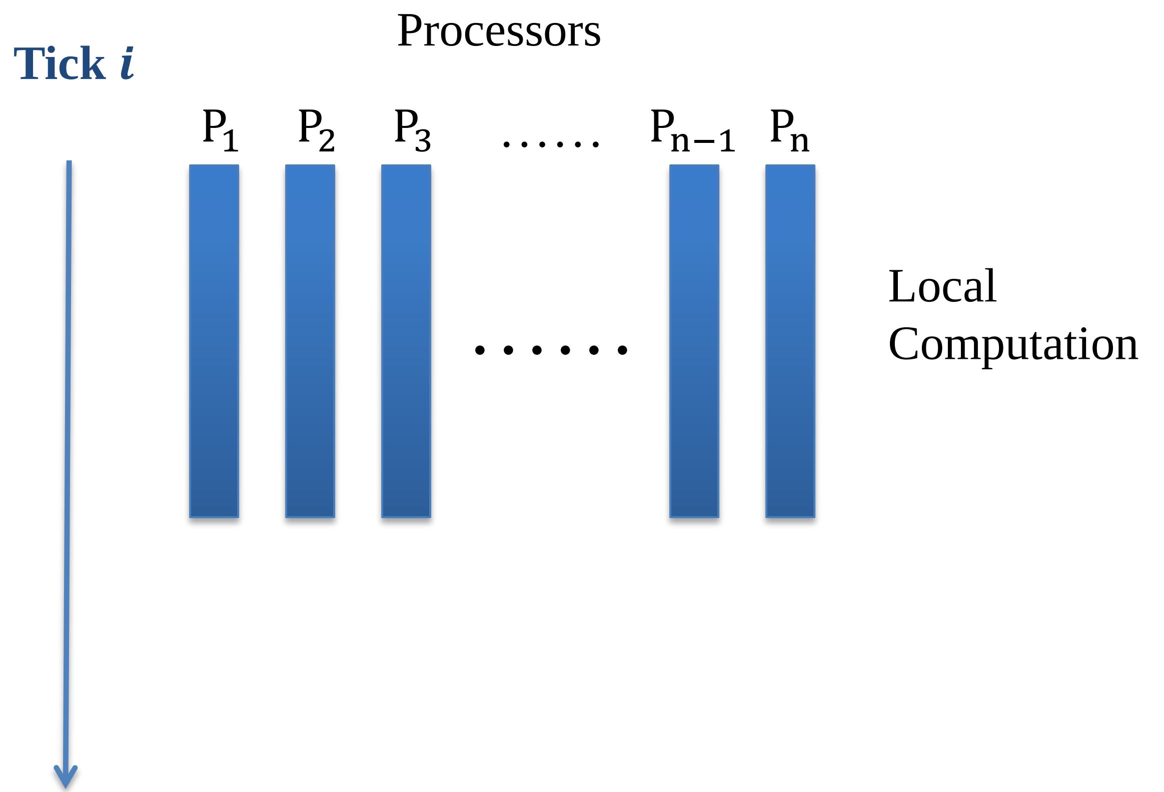
Making Time-stepped Applications Tick in the Cloud

Tao Zou, Guozhang Wang, Marcos Vaz Salles*,
David Bindel, Alan Demers,
Johannes Gehrke, Walker White

Cornell University
*University of Copenhagen (DIKU)

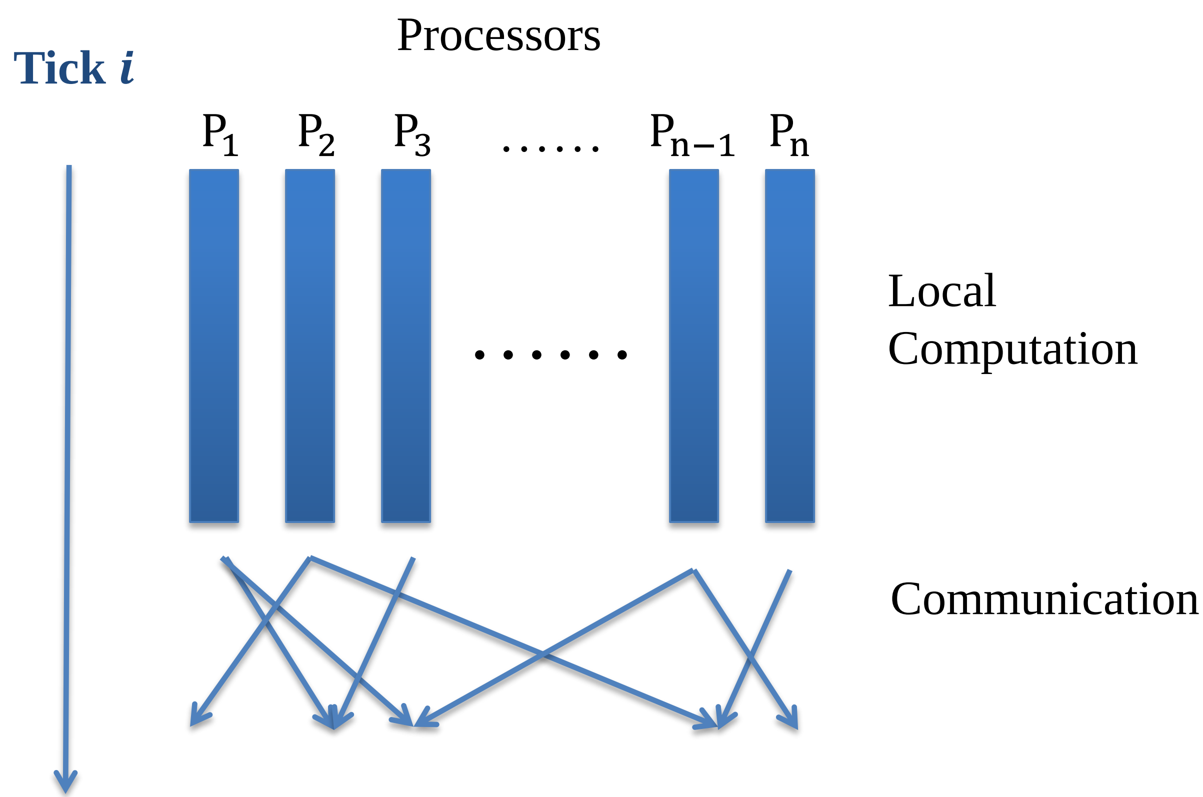
Time-Stepped Applications

- Executed with parallelism organized into logical ticks.
- Implemented using Bulk Synchronous Parallel (BSP) Model



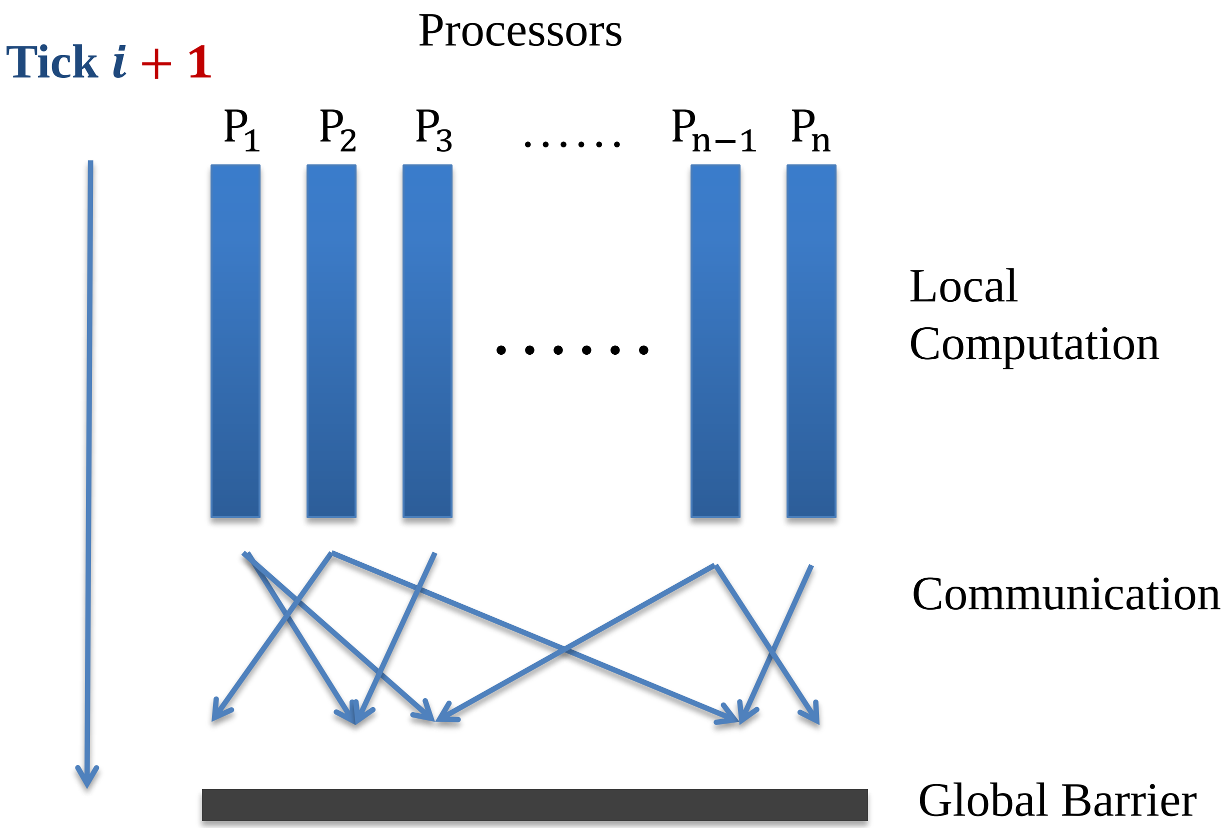
Time-Stepped Applications

- Executed with parallelism organized into logical ticks.
- Implemented using Bulk Synchronous Parallel (BSP) Model



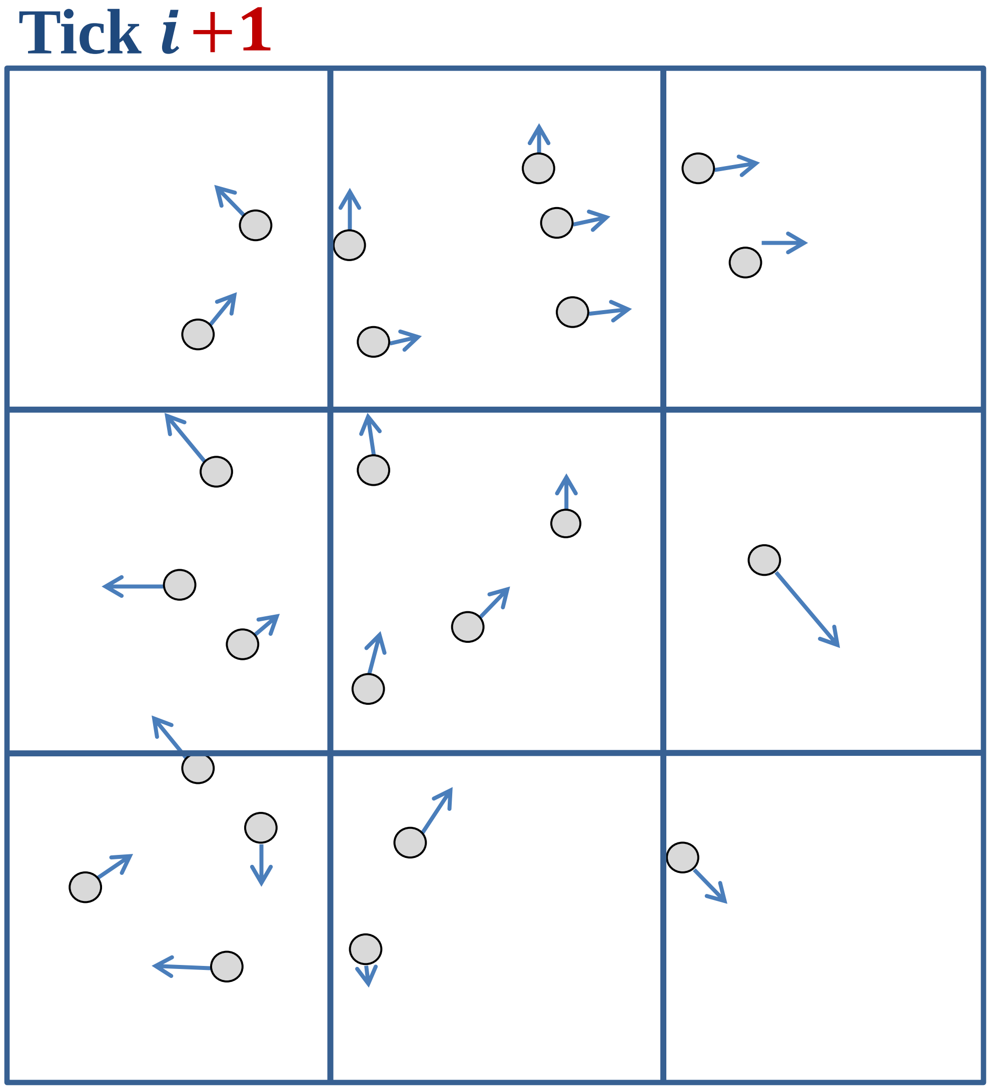
Time-Stepped Applications

- Executed with parallelism organized into logical ticks.
- Implemented using Bulk Synchronous Parallel (BSP) Model



Running Example: Fish Simulation

- Behavioral Simulation
  - Traffic simulation
  - Simulation of groups of animals



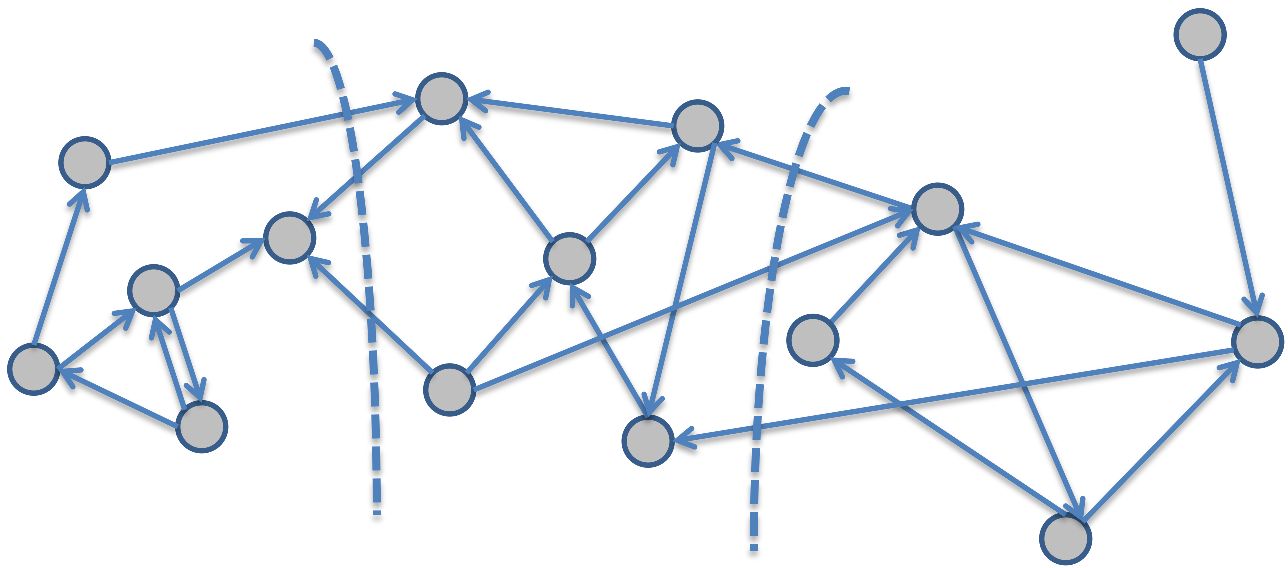
Running Example: Fish Simulation

- Behavioral Simulation
  - Traffic simulation
  - Simulation of groups of animals

Tick $i+1$

Other Time-Stepped Applications

- Iterative Graph Processing



- Matrix Computation

Other Time-Stepped Applications

- Iterative Graph Processing

- Matrix Computation

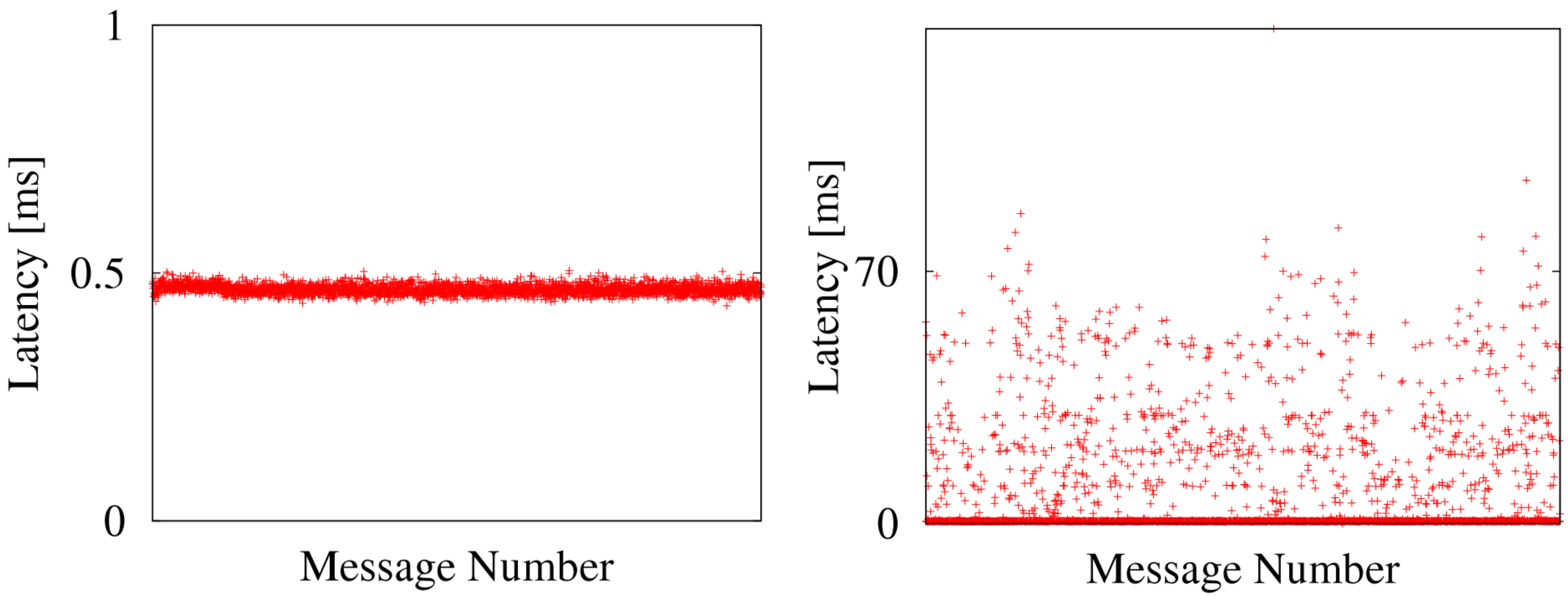
Why Run Scientific Applications in the Cloud?

- Elasticity
- Cost Saving
- Instant Availability

Avoid jobs queuing for days

What Does Cloud Infrastructure Imply

→Unstable network latencies

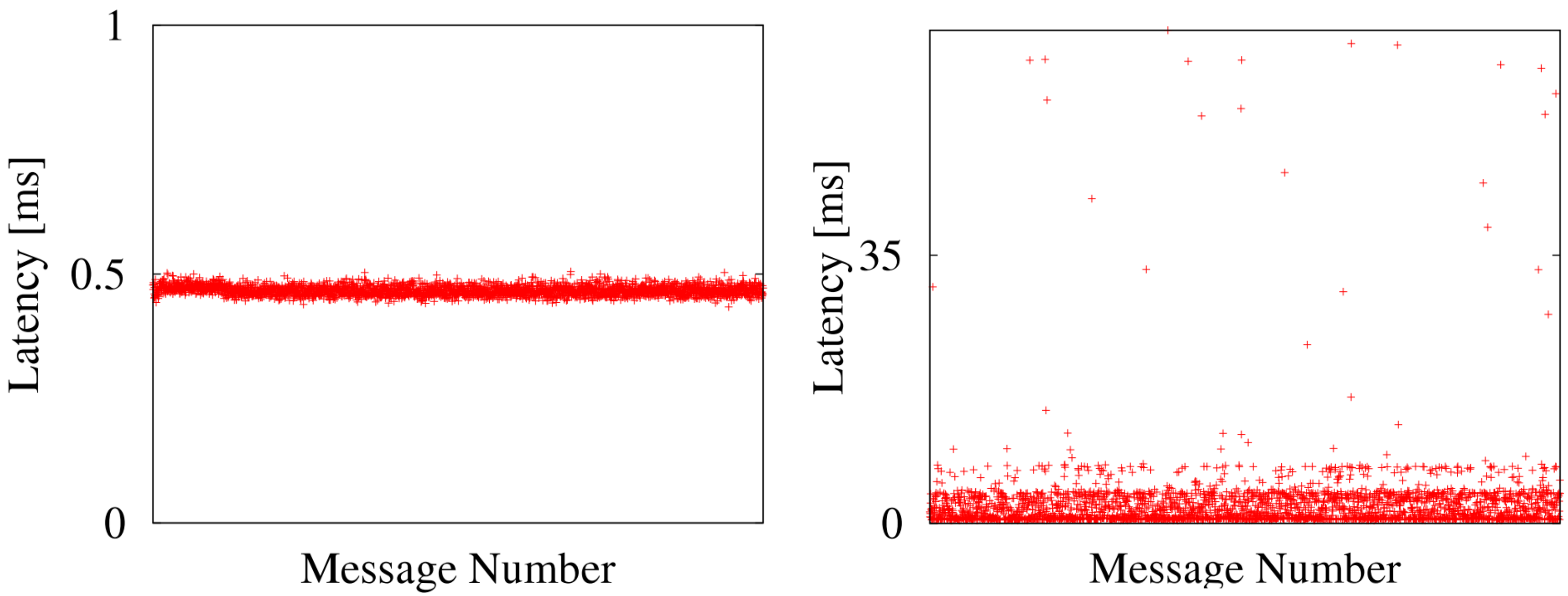


Local Cluster VS EC2 Small Instance

- Virtualization
- Lack of network performance isolation

What Does Cloud Infrastructure Imply

→Unstable network latencies

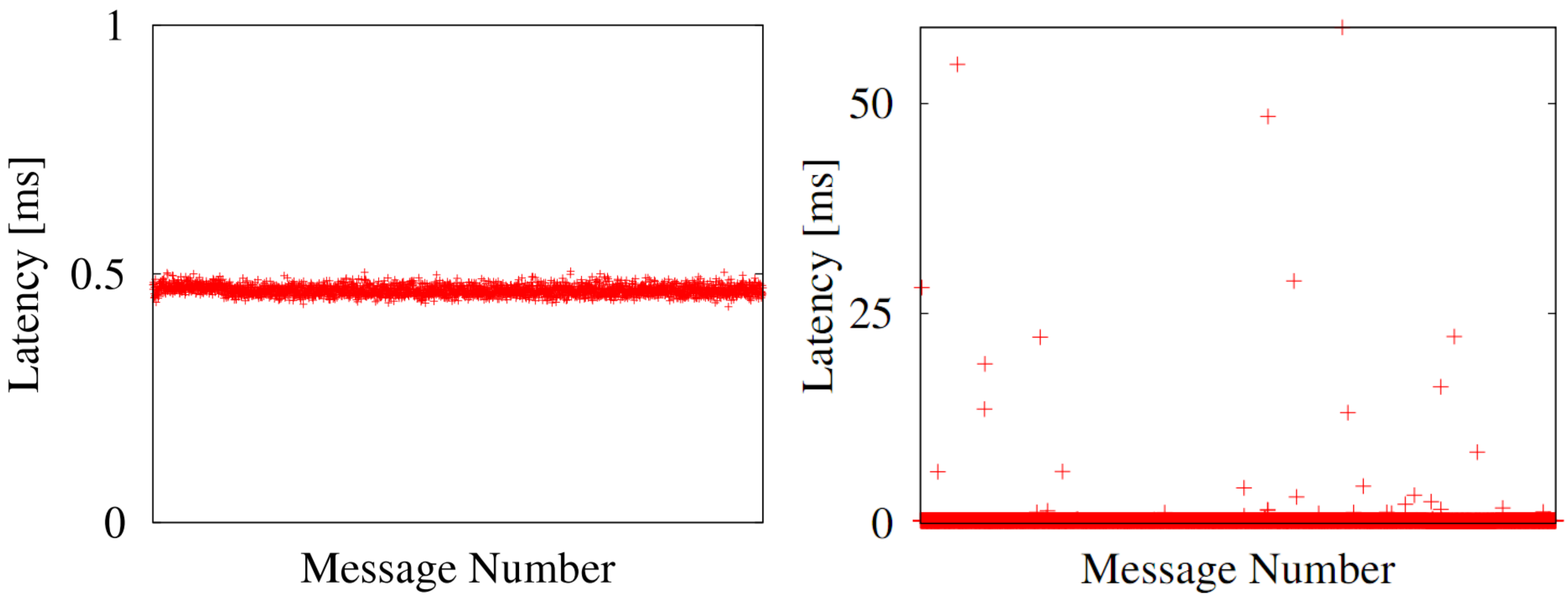


Local Cluster VS EC2 Large Instance

- Virtualization
- Lack of network performance isolation

What Does Cloud Infrastructure Imply

→Unstable network latencies

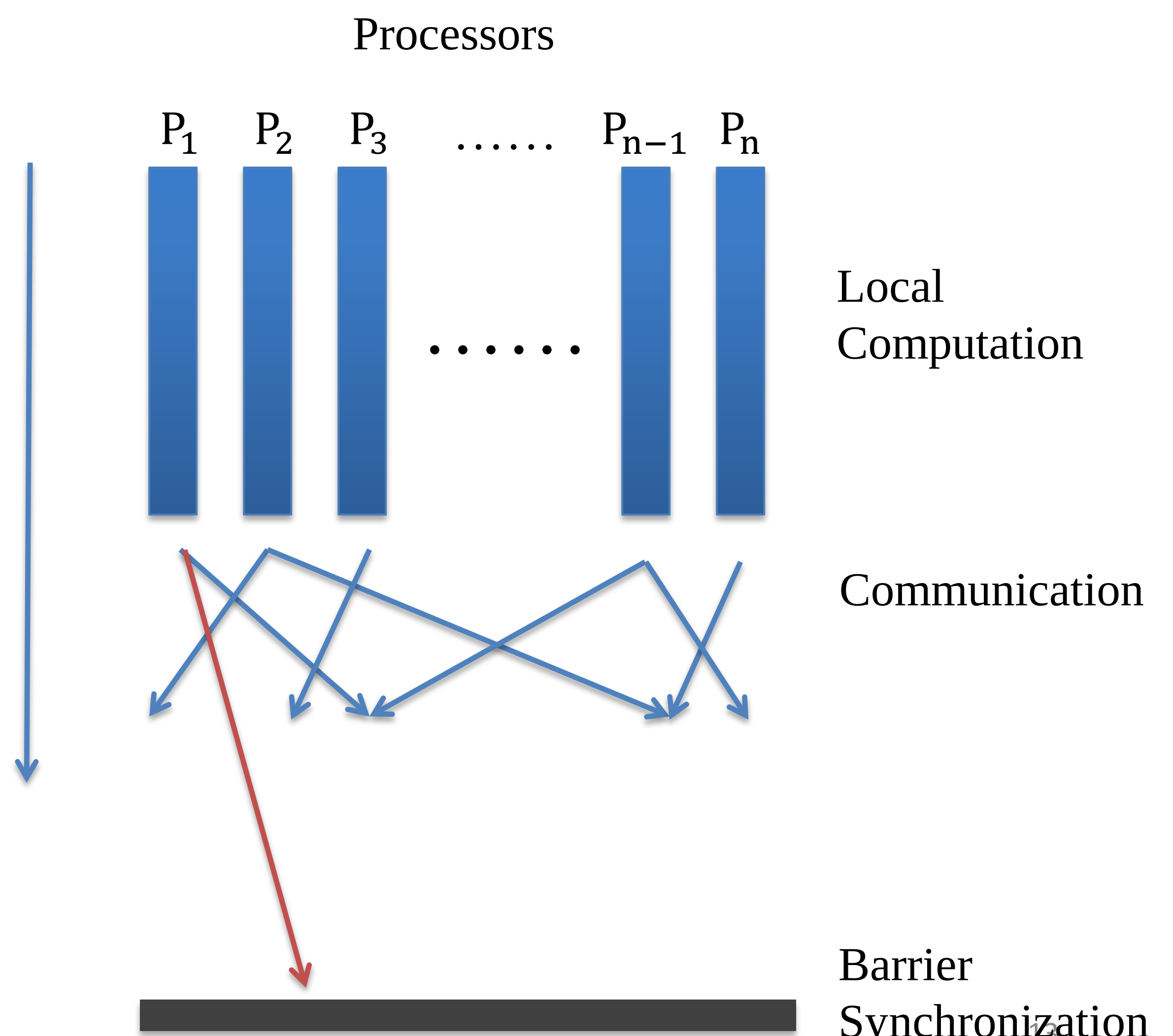


Local Cluster VS EC2 Cluster Instance

- Virtualization
- Lack of network performance isolation

Time-Stepped Applications under Latency Jitter

- Sensitive to latencies
- Remove unnecessary barriers
  - Jitter still propagates



Problem

- Time-stepped applications
- Unstable latencies
- Solution space
  - Improve the networking infrastructure
    - Recent proposals only tackle bandwidth problems
  - Make applications more resistant to unstable latencies

Talk Outline

- Motivation
- Our Approach
- Experimental Results
- Conclusions

Why not Ad-Hoc Optimizations?

- Disadvantages
  - No Generality

    Goal: Applicable to all time-stepped applications
  - No Ease of Programming

    Goal: Transparent optimization and communication
  - Error-Prone

    Goal: Correctness guarantee
- Programming Model + Jitter-tolerant Runtime

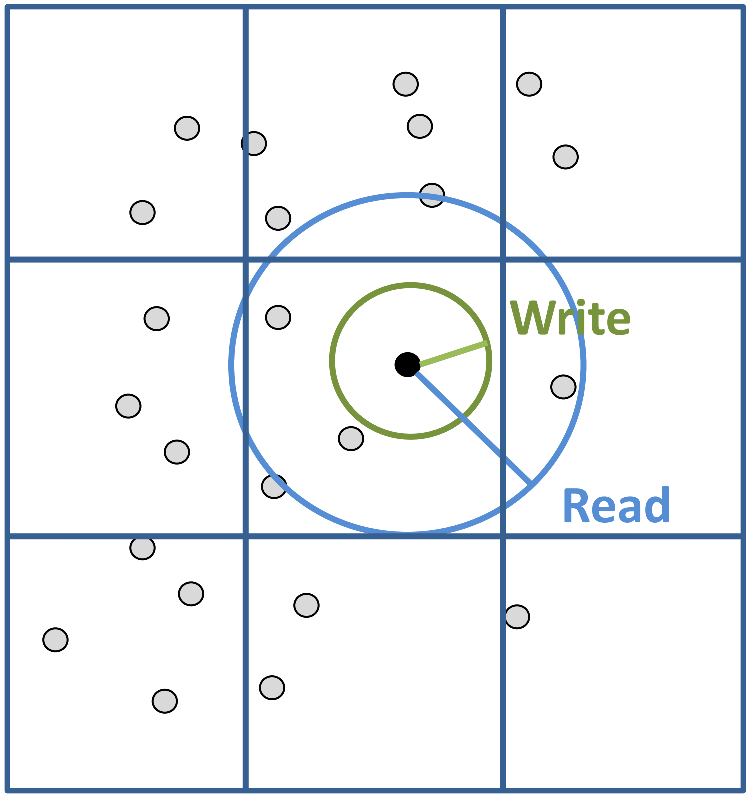
Talk Outline

- Motivation
- Our Approach
  - Programming Model
  - Jitter-tolerant Runtime
- Experimental Results
- Conclusions

Data Dependencies: What to Communicate

- Read Dependency
  - Example: How far can a fish see?
- Write Dependency
  - Example: How far can a fish move?

- Key: Modeling Dependencies



Programming Model

Modeling State

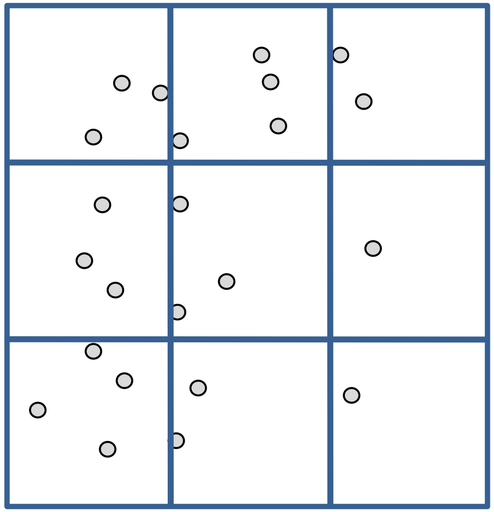
- Motivated by thinking of the applications as distributed database system
- Application state: Set of tuples
  - Fish → tuple
  - Fish school → application state
- Selection over state: Query
  - 2D range query over fish school

Programming Model

Modeling Data Parallelism

- Partition Function:

$$\mathrm{PART}(n) \rightarrow \mathrm{Q}_1, \mathrm{Q}_2, \ldots, \mathrm{Q}_\mathrm{n}$$



Programming Model

Modeling Data Parallelism

- Partition Function:

$$\mathrm{PART}(n) \rightarrow \mathrm{Q}_1, \mathrm{Q}_2, \ldots, \mathrm{Q}_\mathrm{n}$$

Q_1	Q_2	Q_3
Q_4	Q_5	Q_6
Q_7	Q_8	Q_9

Programming Model

Modeling Data Parallelism

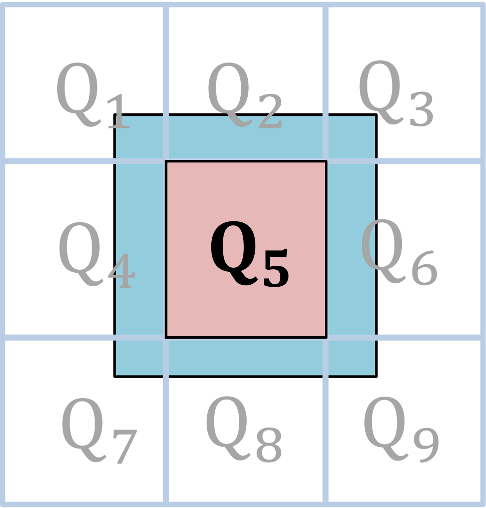
- Partition Function:

  $\mathrm{PART}(n) \rightarrow \mathrm{Q}_1, \mathrm{Q}_2, \ldots, \mathrm{Q}_\mathrm{n}$

Modeling Computation

- Parallel Computation:

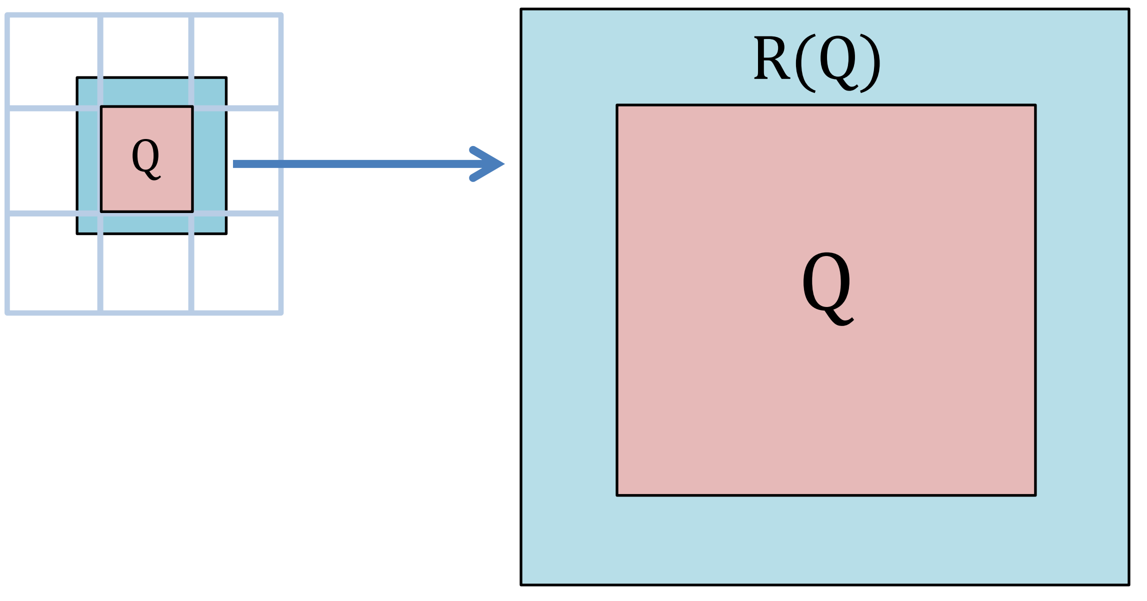
STEP(ToCompute , Context)



- Context: How large?

Modeling Dependencies: R

- Read Dependency: R(Q)

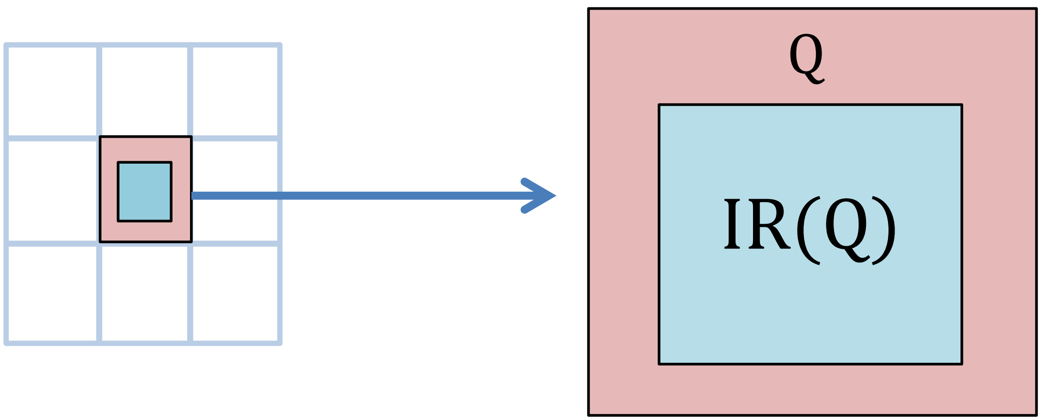


  – Contains all necessary tuples in context to compute Q

STEP(Q, R(Q))

Modeling Dependencies: IR

- $\mathrm{STEP}\left(\quad ? \quad , \quad \mathrm{Q} \quad \right)$
- Inverse Read Dependency: $\mathrm{IR(Q)}$



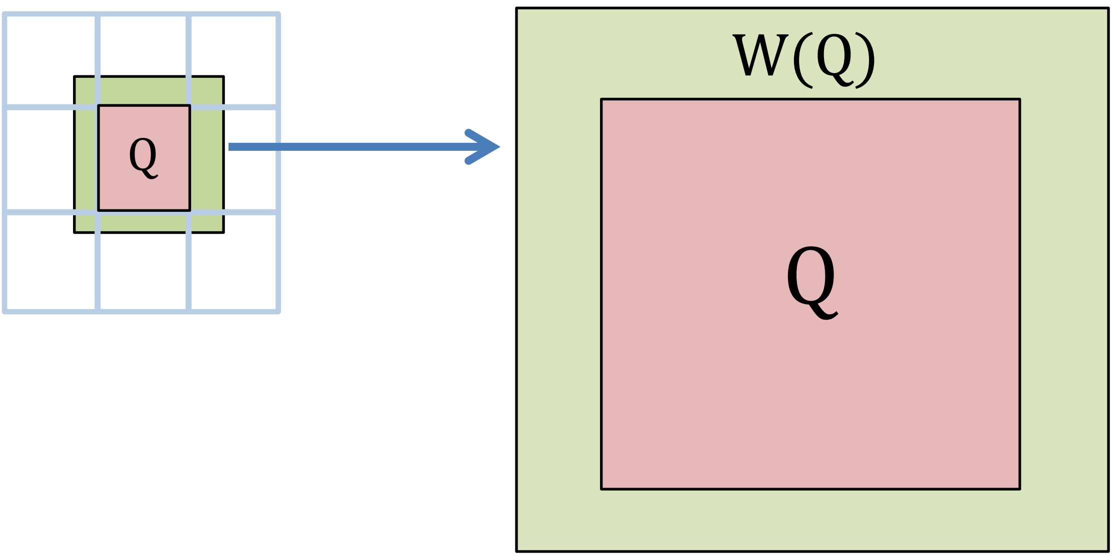
  - Contains all tuples that can be computed with Q as context

$$\mathrm{STEP}(\,\mathrm{IR(Q)},\ \mathrm{Q})$$

  - $\mathrm{IR} \approx \mathrm{R}^{-1}$

Modeling Dependencies: W

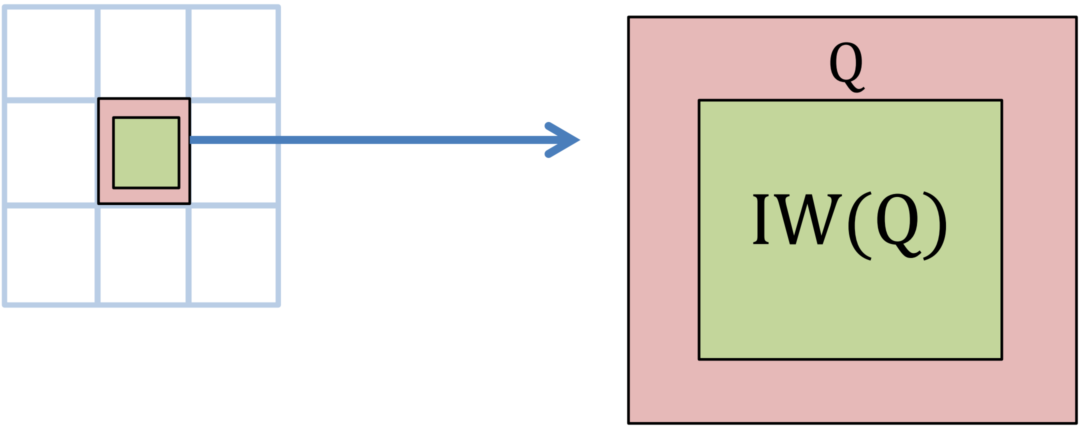
- Write Dependency: W(Q)



  – Contains all tuples generated by computing Q

Modeling Dependencies: IW

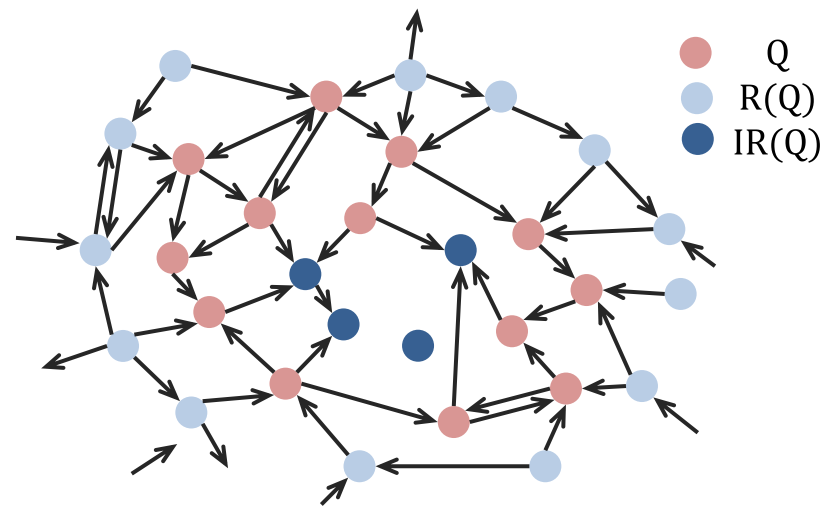
- Inverse Write Dependency: IW(Q)



  – Contains all tuples in the next tick after computing Q
  – $\mathrm{IW} \approx \mathrm{W}^{-1}$

Programming Model: All together

- PART – data parallelism
- STEP – computation
- R, IR – read dependencies
- W, IW– write dependencies



PageRank

- Remarks:
  - Users inherently think in terms of dependencies
  - Not limited to spatial properties

Talk Outline

- Motivation
- Our Approach
  - Programming Model
  - Jitter-tolerant Runtime
- Experimental Results
- Conclusions

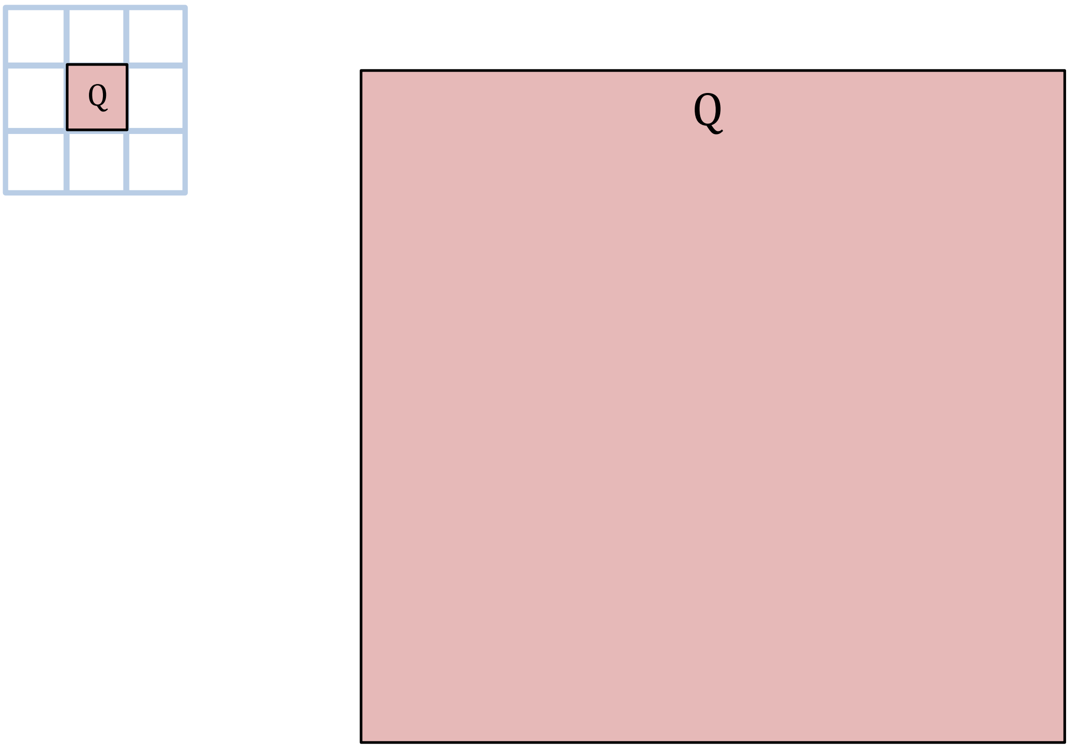
Jitter-tolerant Runtime

- Input: Functions defined in programming model
- Output: Parallel computation results
- Requirement:

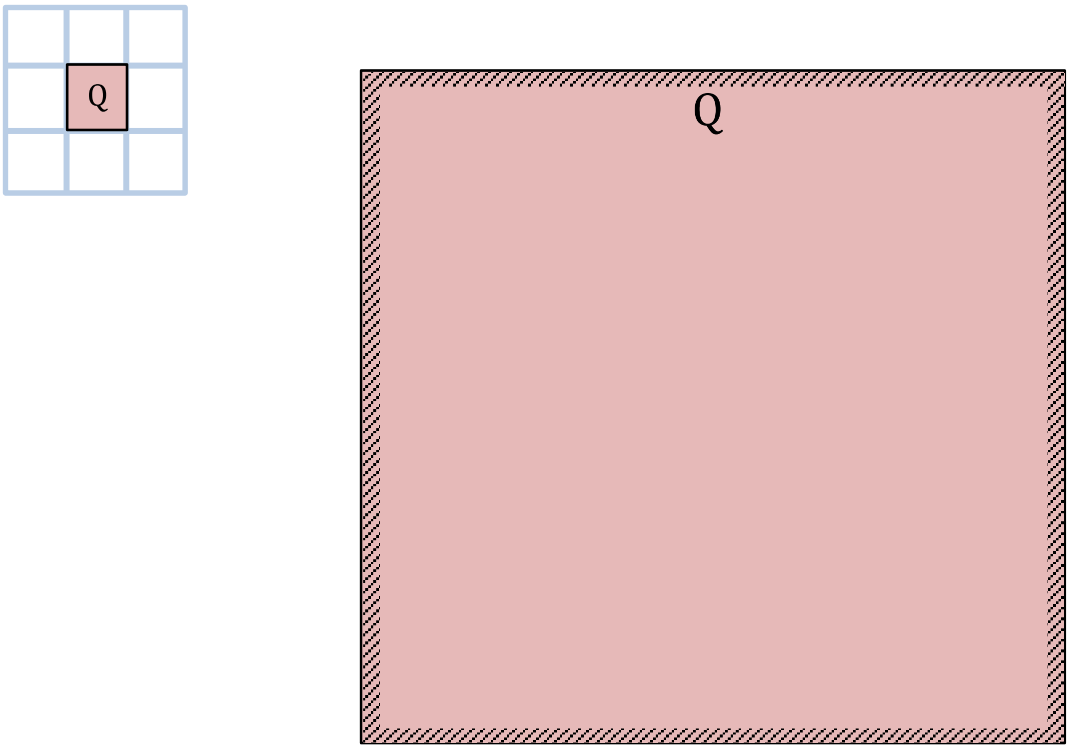
  Efficiency and Correctness

Runtime Dependency Scheduling

Tick *t*



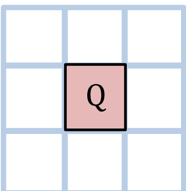
Runtime Dependency Scheduling

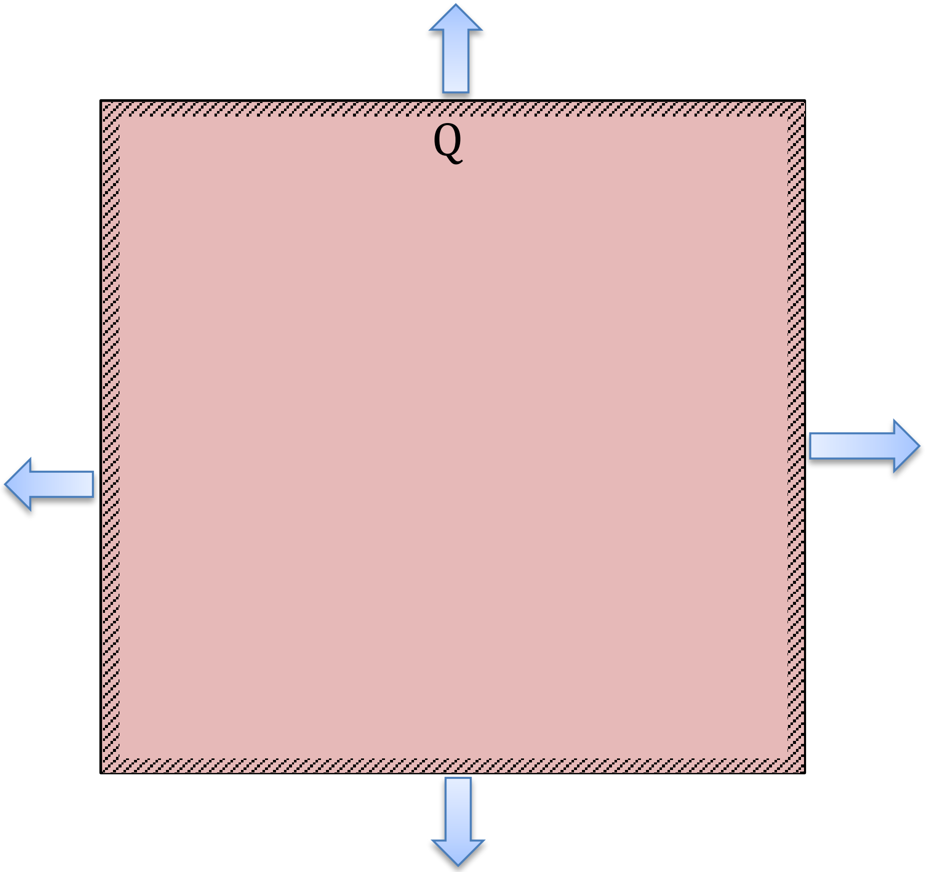


Tick *t*

Compute Q

Runtime Dependency Scheduling



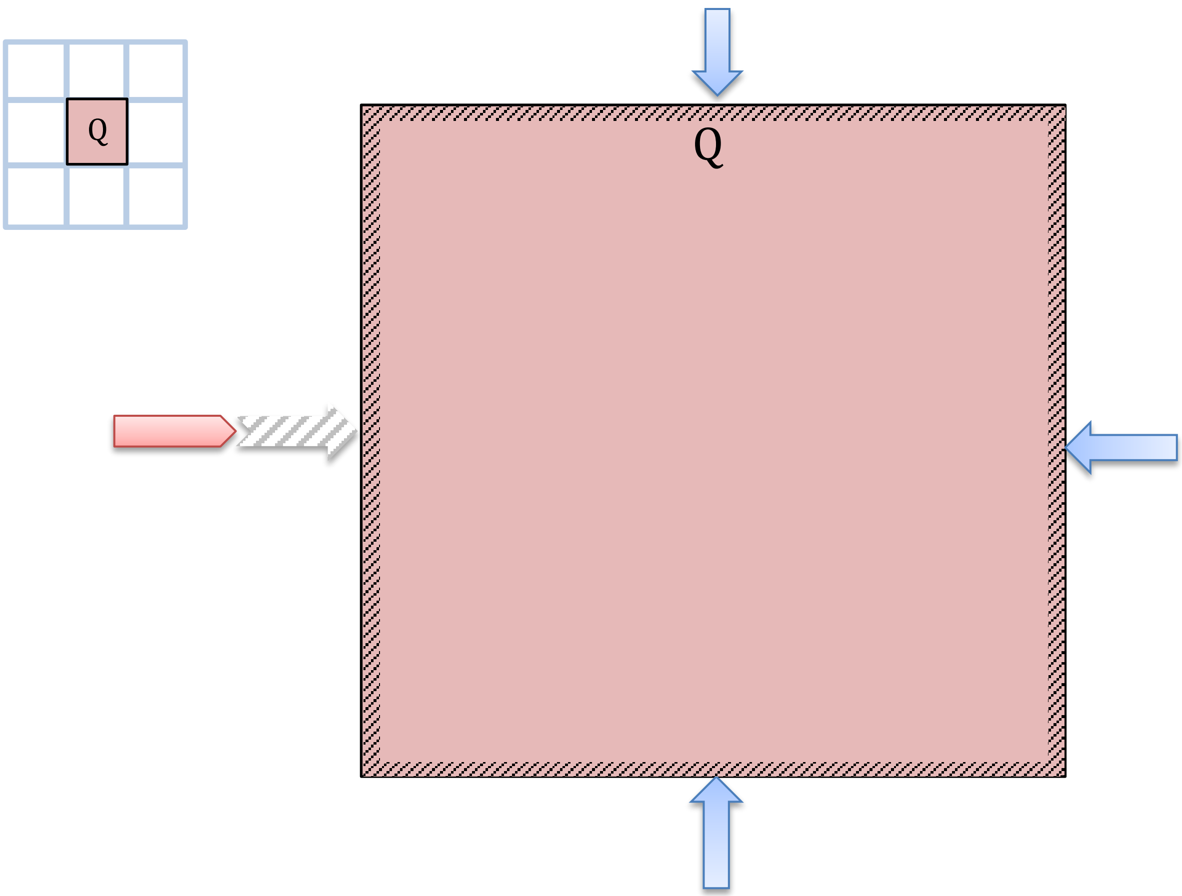


Tick *t*

Compute Q

Send out updates

Runtime Dependency Scheduling



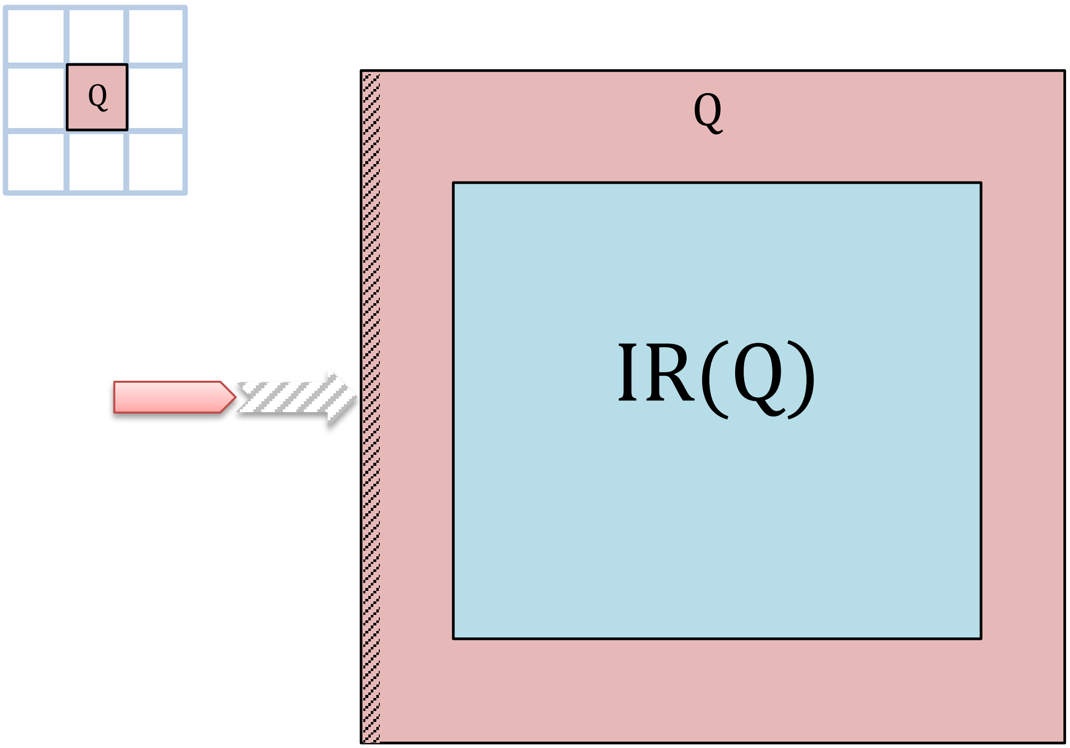
Tick *t*

Compute Q

Send out updates

Wait for messages

Runtime Dependency Scheduling



Tick t

Compute Q

Send out updates

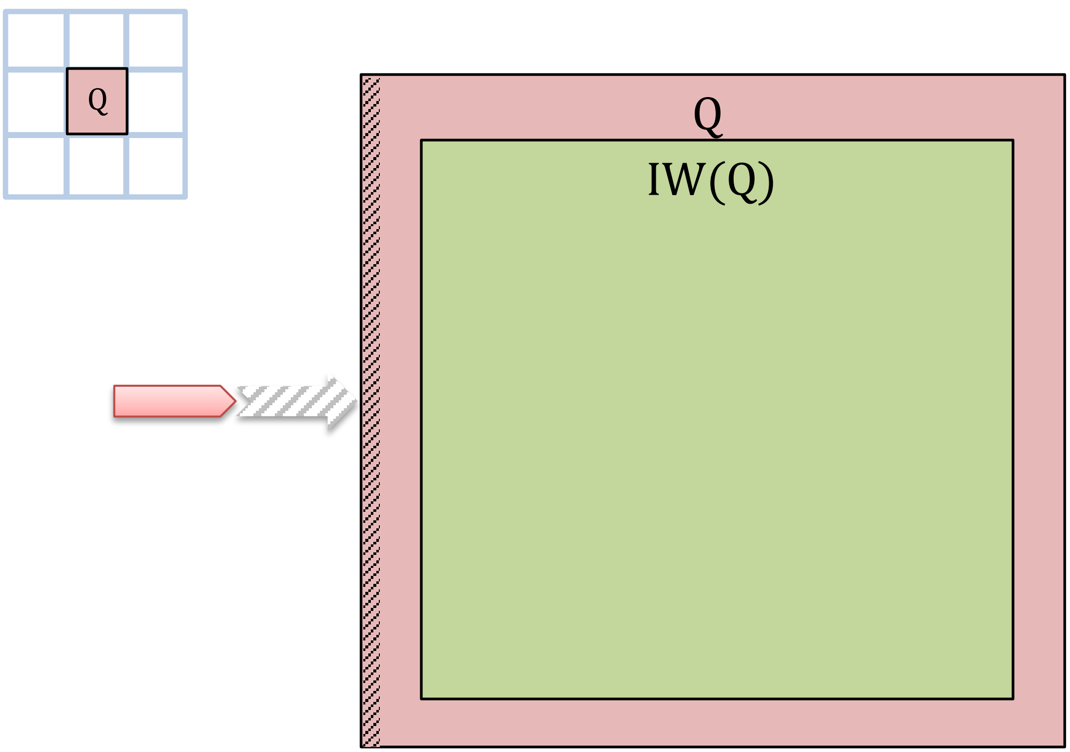
Wait for messages

Tick $t+1$

Compute IR(Q) ?

No. Incoming message may contain updates to Q.

Runtime Dependency Scheduling



Tick t

Compute Q

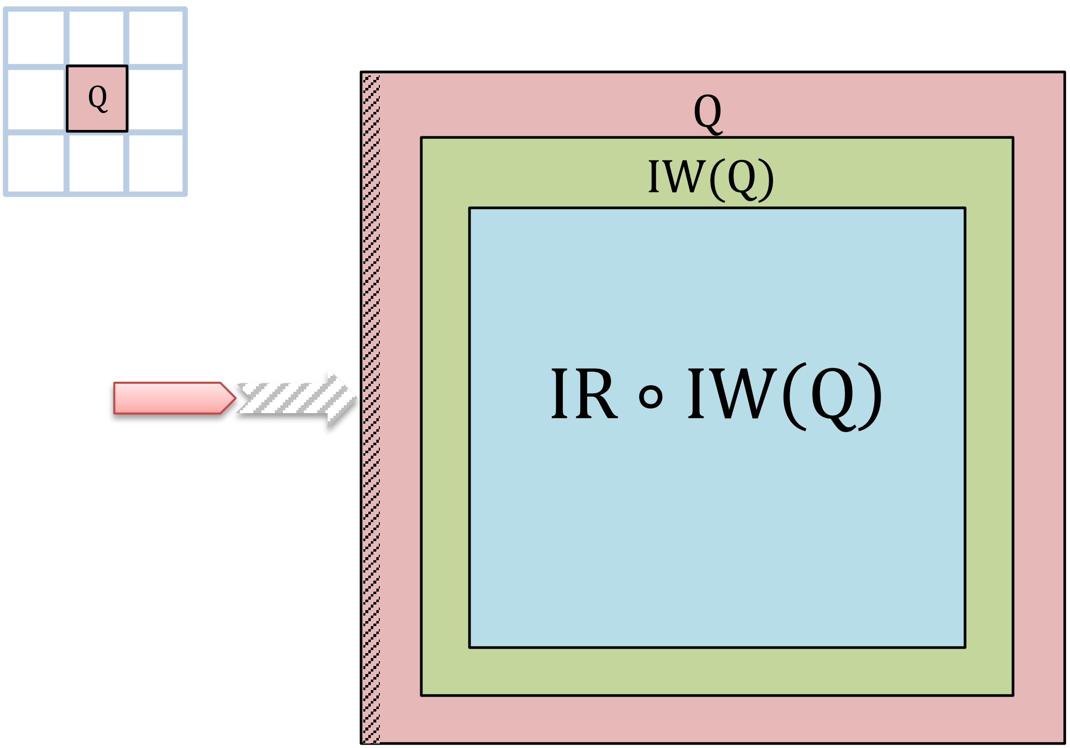
Send out updates

Wait for messages

Tick $t + 1$

IW(Q) is not influenced by the messages

Runtime Dependency Scheduling



Tick t

Compute Q

Send out updates

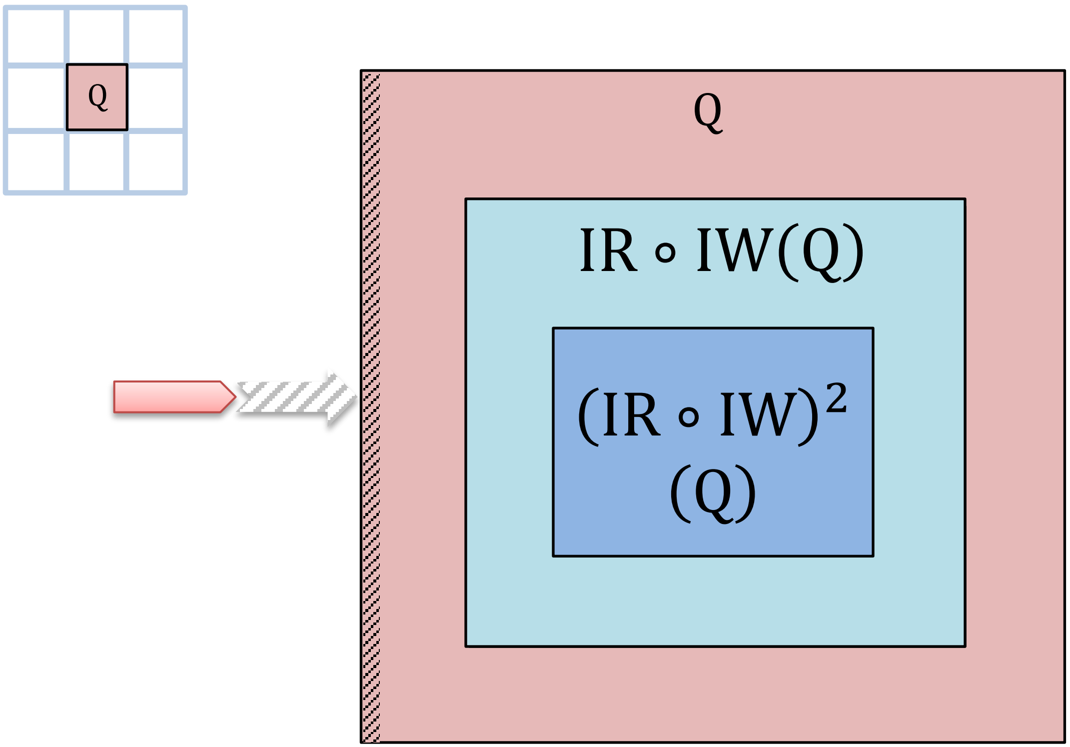
Wait for messages

Tick $t + 1$

Compute IR ∘ IW(Q)

IW(Q) is not influenced by the messages

Runtime Dependency Scheduling



Tick t

Compute Q

Send out updates

Wait for messages

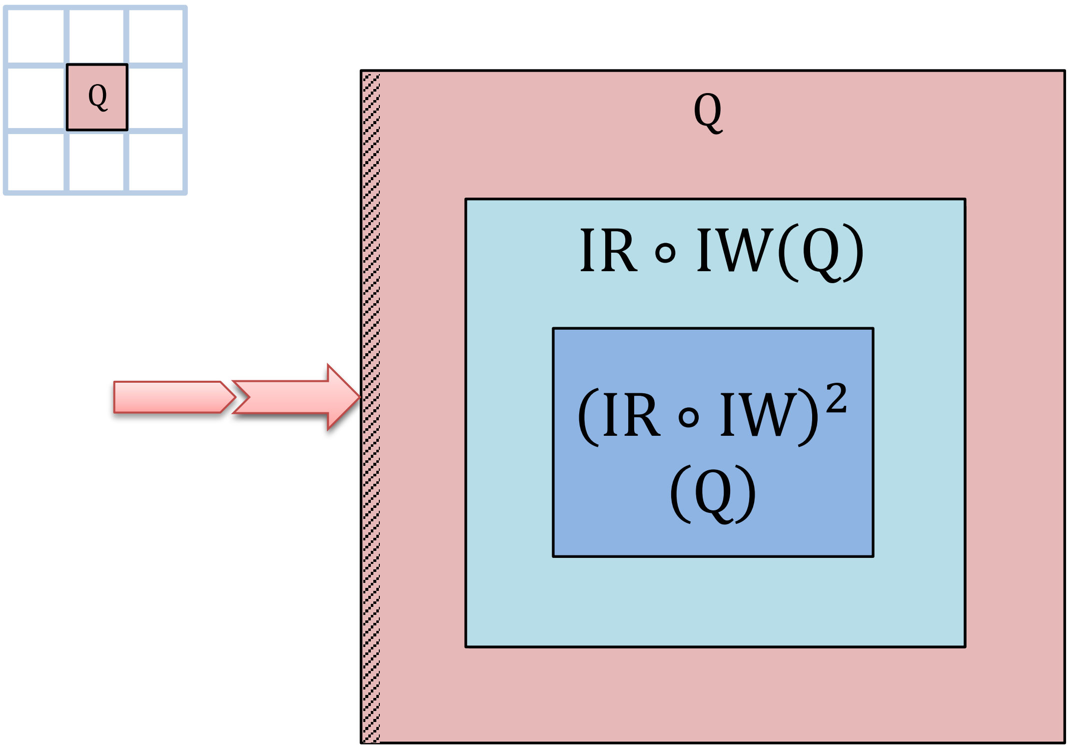
Tick $t + 1$

Compute IR ∘ IW(Q)

Tick $t + 2$

Compute $(\mathrm{IR} \circ \mathrm{IW})^2(\mathrm{Q})$

Runtime Dependency Scheduling



Tick t

Compute Q

Send out updates

All message received

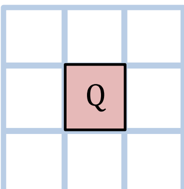
Tick $t+1$

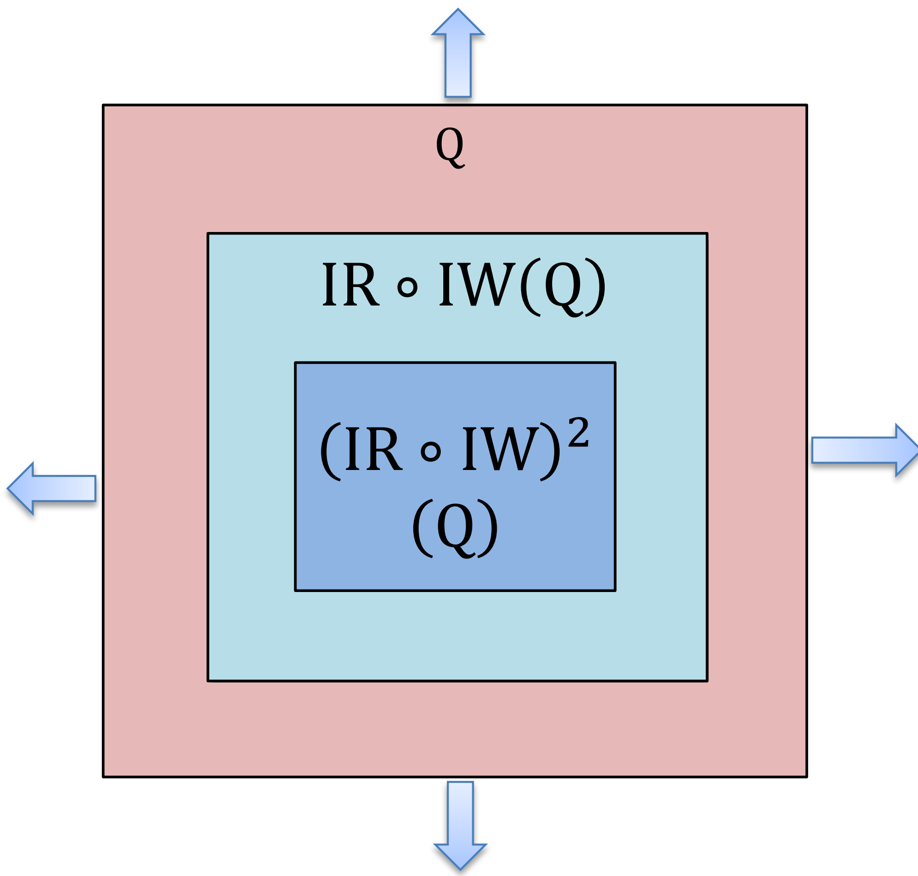
Compute $IR \circ IW(Q)$

Tick $t+2$

Compute $(IR \circ IW)^2(Q)$

Runtime
Dependency Scheduling





Tick t

Compute Q

Send out updates

All message received

Tick $t + 1$

Compute $\mathrm{IR} \circ \mathrm{IW}(\mathrm{Q})$

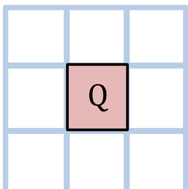
Compute $\mathrm{Q} - \mathrm{IR} \circ \mathrm{IW}(\mathrm{Q})$

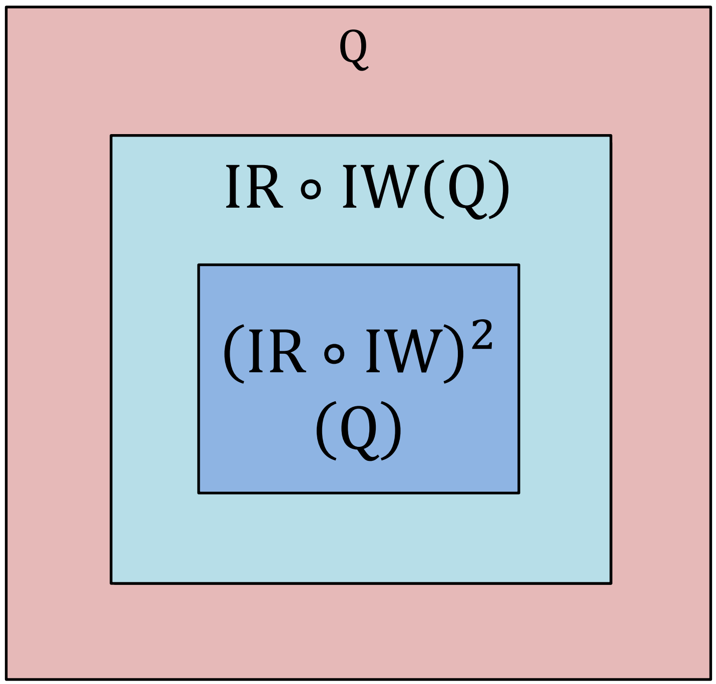
Send out updates

Tick $t + 2$

Compute $(\mathrm{IR} \circ \mathrm{IW})^2(\mathrm{Q})$

Runtime Dependency Scheduling





Tick t

Compute Q

Send out updates

All message received

Tick $t + 1$

Compute $IR \circ IW(Q)$

Compute $Q - IR \circ IW(Q)$

Send out updates

……

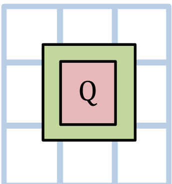
Tick $t + 2$

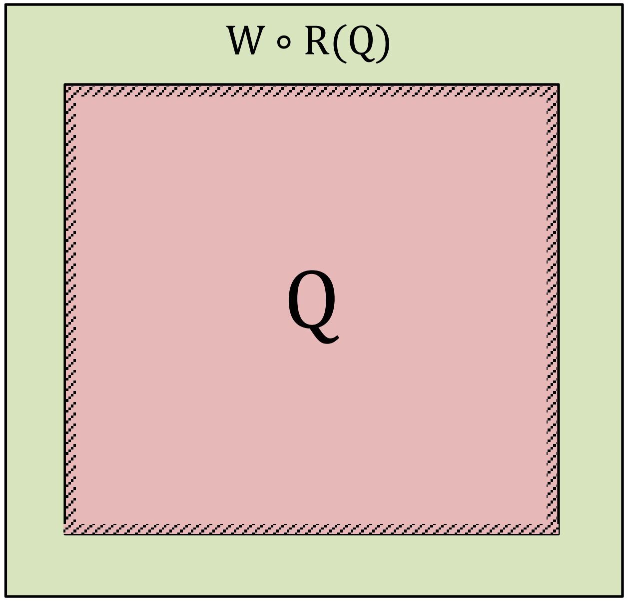
Compute $(IR \circ IW)^2(Q)$

……

Intuition: schedule computation for future ticks when delayed

Runtime
Computational Replication



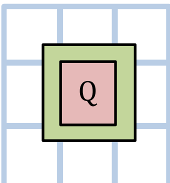


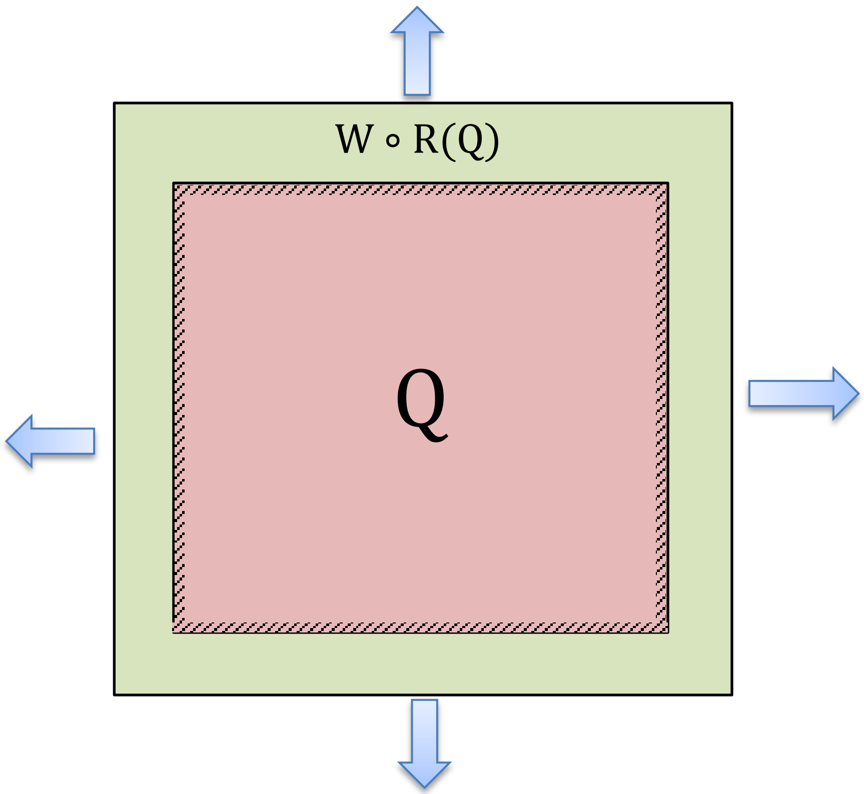
Tick *t*

Compute Q

Tick *t* + 1

Runtime
Computational Replication





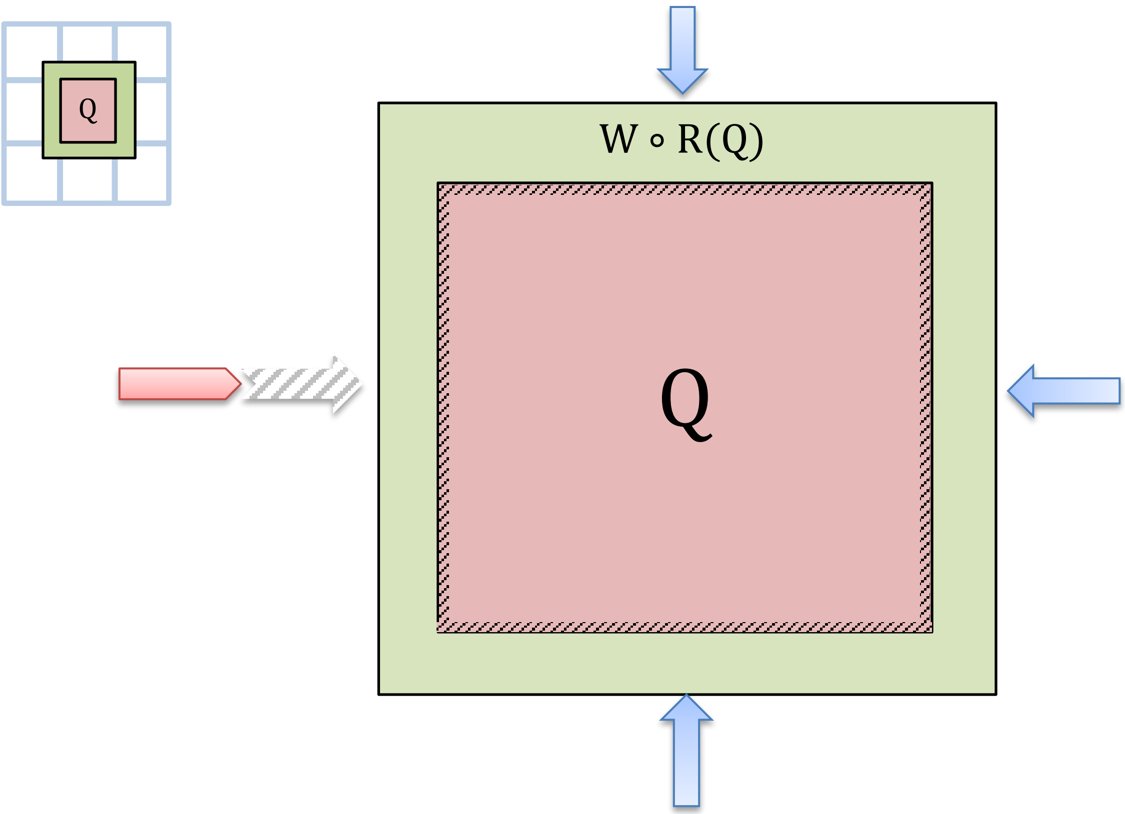
Tick t

Compute Q

Send out updates

Tick $t + 1$

Runtime
Computational Replication



Tick t

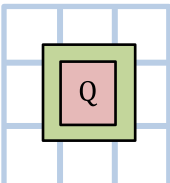
Compute Q

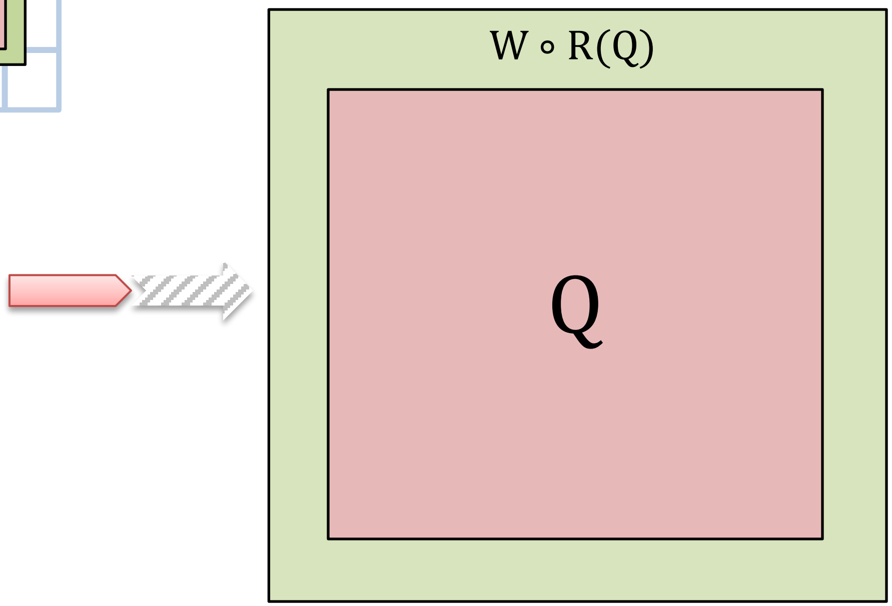
Send out updates

Wait for messages

Tick $t + 1$

Runtime
Computational Replication





Tick t

Compute Q

Send out updates

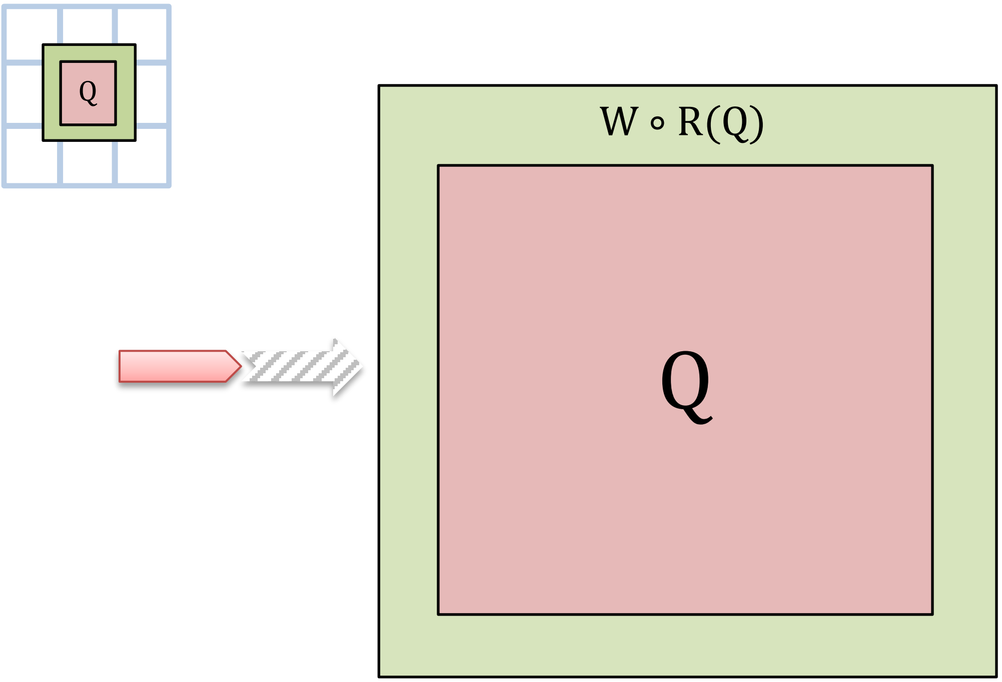
Wait for messages

Compute $W \circ R(Q) - Q$

Tick $t+1$

Compute Q

Runtime
Computational Replication



Tick t

Compute Q

Send out updates

Wait for messages

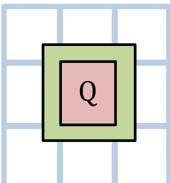
Compute $\mathrm{W} \circ \mathrm{R(Q)} - \mathrm{Q}$

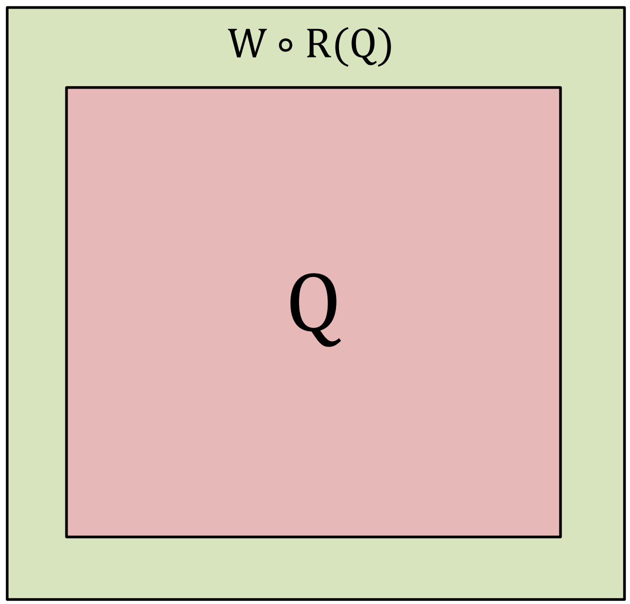
Tick $t + 1$

Compute Q

……

Runtime Computational Replication





Tick t

Compute Q

Send out updates

Wait for messages

Compute $\mathrm{W} \circ \mathrm{R}(\mathrm{Q}) - \mathrm{Q}$

Tick $t + 1$

Compute Q

……

- Intuition: enlarge region to compute contents of delayed messages.
- $\mathrm{W} \circ \mathrm{R}(\mathrm{Q}), (\mathrm{W} \circ \mathrm{R})^2(\mathrm{Q}), \ldots, (\mathrm{W} \circ \mathrm{R})^m(\mathrm{Q})$

Our Approach: Summary

- Programming model captures
  - Application state
  - Computation logic
  - Data dependencies
- Jitter-tolerant runtime
  - Dependency scheduling
  - Computational replication

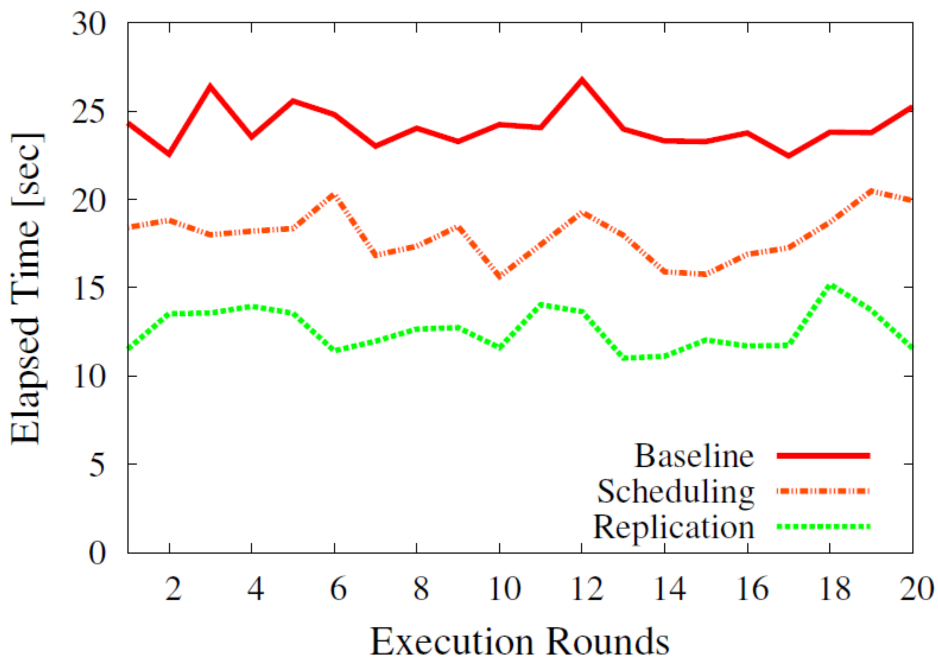
Talk Outline

- Motivation
- Our Approach
  - Programming Model
  - Jitter-tolerant Runtime
- Experimental Results
- Conclusions

Experimental Setup

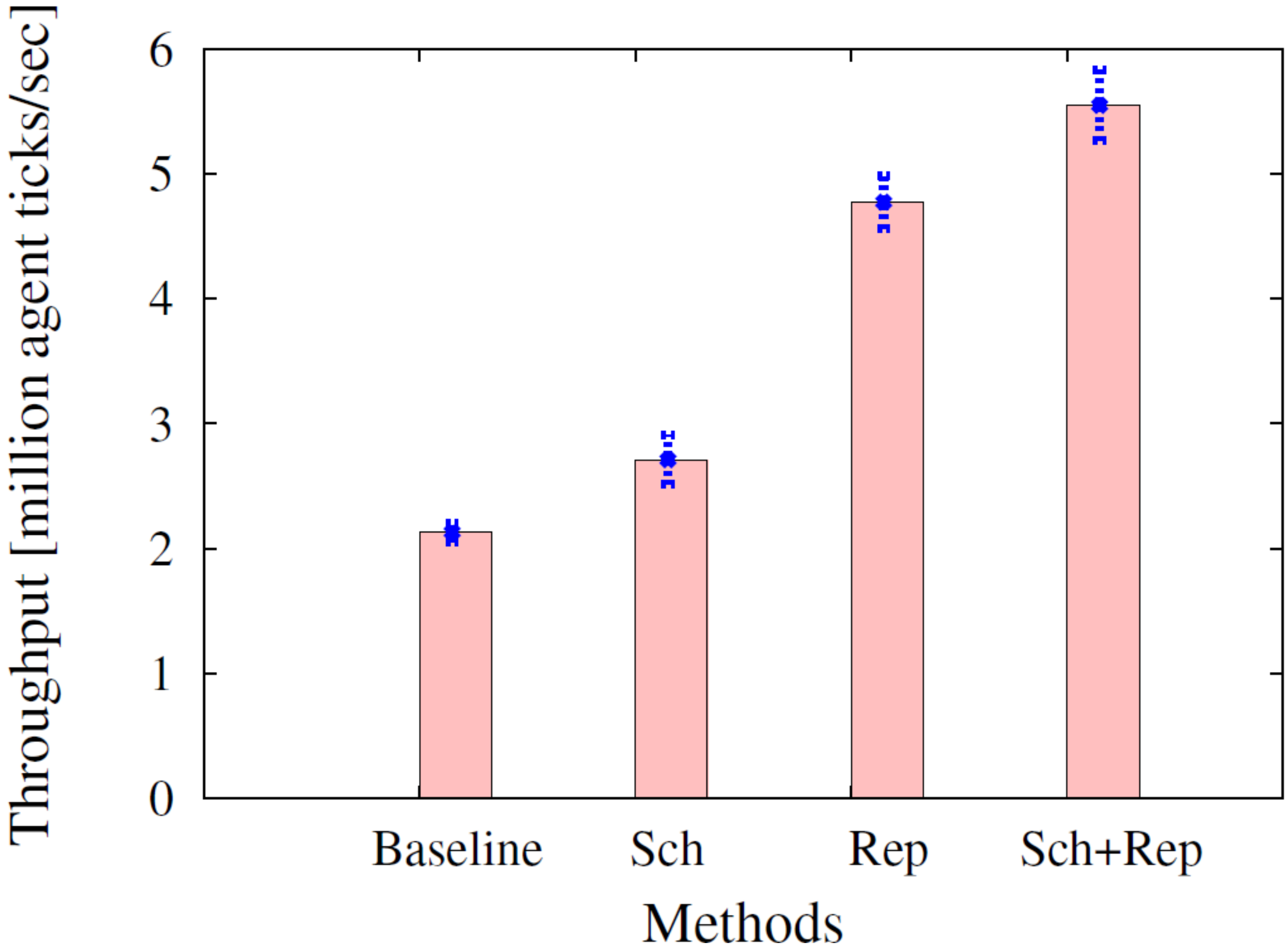
- A prototype framework
  - Jitter-tolerant runtime
    - MPI for communication
  - Three different applications
    - A fish school behavioral simulation
    - A linear solver using the Jacobi method
    - A message-passing algorithm computes PageRank
- Hardware Setup
  - Up to 100 EC2 large instances (m1.large)
    - 2.26GHz Xeon cores with 6MB cache
    - 7.5GB main memory

Methodology



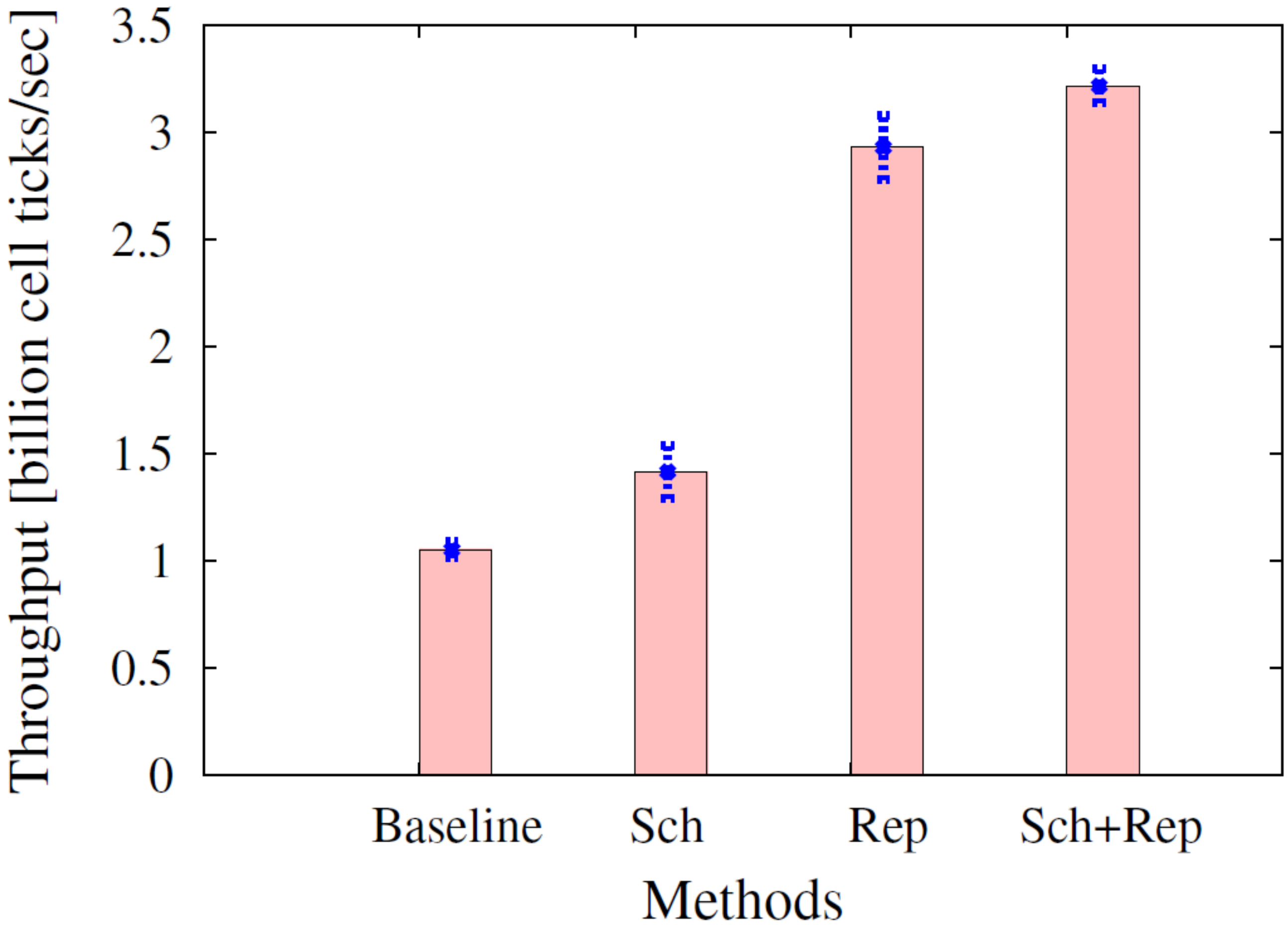
- Observation: Temporal variation in network performance
- Solution
  - Execute all settings in rounds of fixed order
  - At least 20 consecutive executions of these rounds

Effect of Optimization: Fish Sim



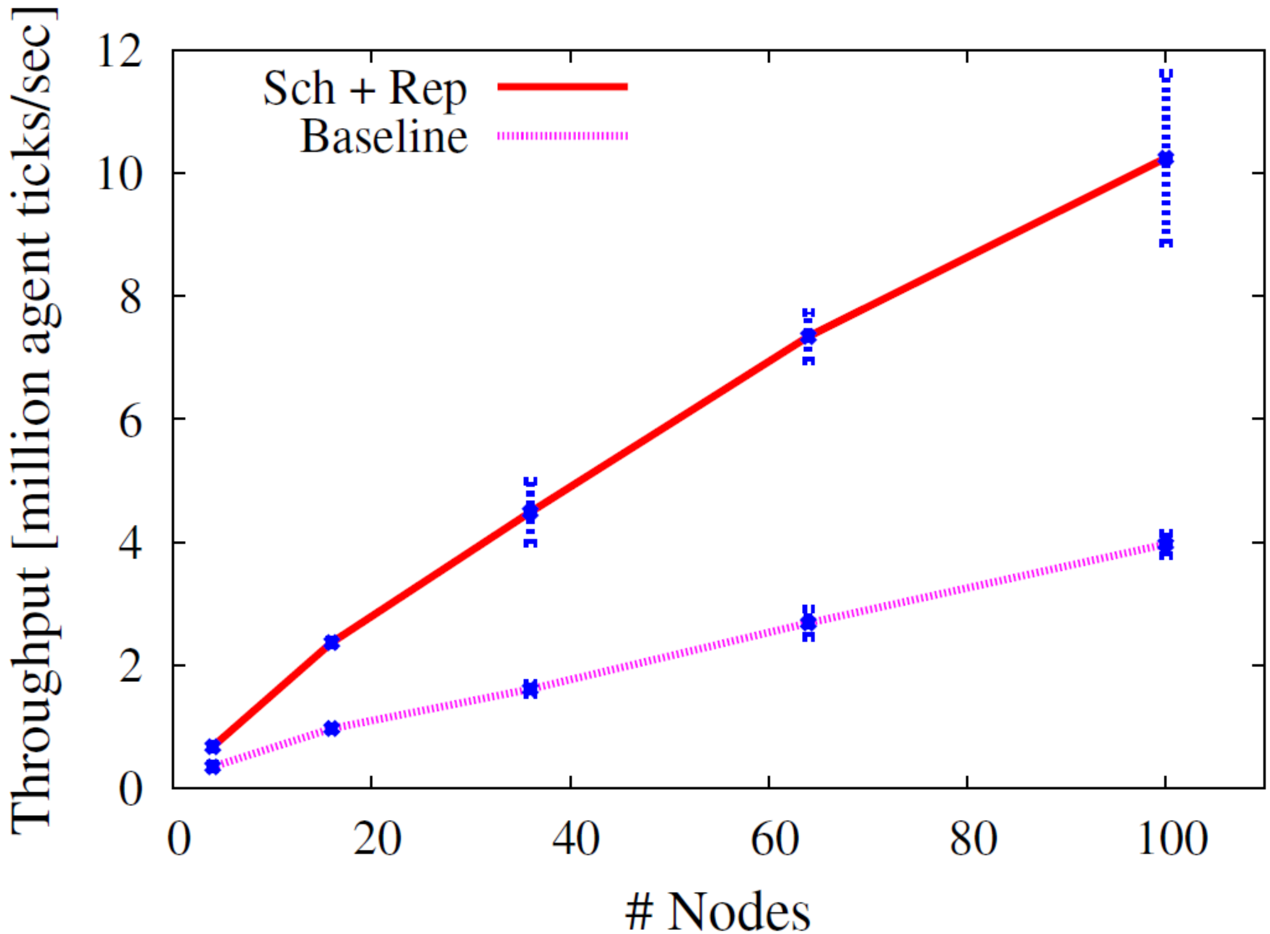
- **Baseline**: Local Synchronization; **Sch**: Dependency Scheduling; **Rep**: Computational Replication

Effect of Optimization: Jacobi



- **Baseline**: Local Synchronization; **Sch**: Dependency Scheduling; **Rep**: Computational Replication

Scalability: Fish Simulation



- **Baseline**: Local Synchronization; **Sch**: Dependency Scheduling; **Rep**: Computational Replication

Conclusions

- Latency jitter is a key characteristic of today's cloud environments.
- Programming model + jitter-tolerant runtime
  - Good performance under latency jitter
  - Ease of programming
  - Correctness
- We have released our framework as a public Amazon AMI: http://www.cs.cornell.edu/bigreddata/games/.
- Our framework will be used this fall in CS 5220 (Applications of Parallel Computers) at Cornell.