Parallel Programming
in C with MPI and OpenMP

McGraw Hill

Parallel Programming
in C with MPI and OpenMP

Michael J. Quinn

Mc
Graw
Hill

Algorithm design and basic algorithms

Slides are modified from those found in *Parallel Programming in C with MPI and OpenMP*, Michael Quinn

Outline

Outline

- Task/channel model

Outline

- Task/channel model
- Algorithm design methodology

Outline

- Task/channel model
- Algorithm design methodology
- Case studies

Task/Channel Model

Task/Channel Model

- Parallel computation = set of tasks

Task/Channel Model

- Parallel computation = set of tasks
- Task

Task/Channel Model

- Parallel computation = set of tasks
- Task
  - Program

Task/Channel Model

- Parallel computation = set of tasks
- Task
  - Program
  - Local memory

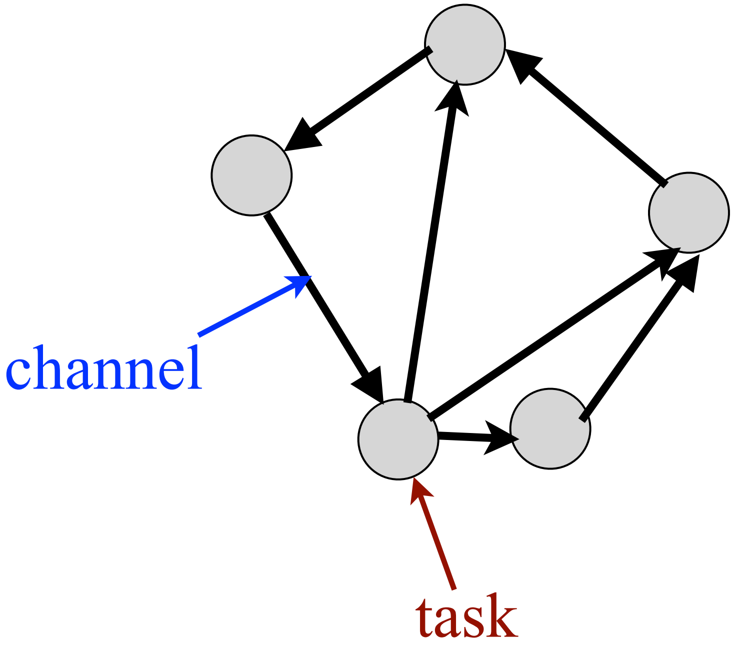
Task/Channel Model

- Parallel computation = set of tasks
- Task
  - Program
  - Local memory
  - Collection of I/O ports

Task/Channel Model

- Parallel computation = set of tasks
- Task
  - Program
  - Local memory
  - Collection of I/O ports
- Tasks interact by sending messages through channels

Task/Channel Model



Foster’s Design Methodology

Foster's Design Methodology

- Partitioning

Foster's Design Methodology

- Partitioning
- Communication

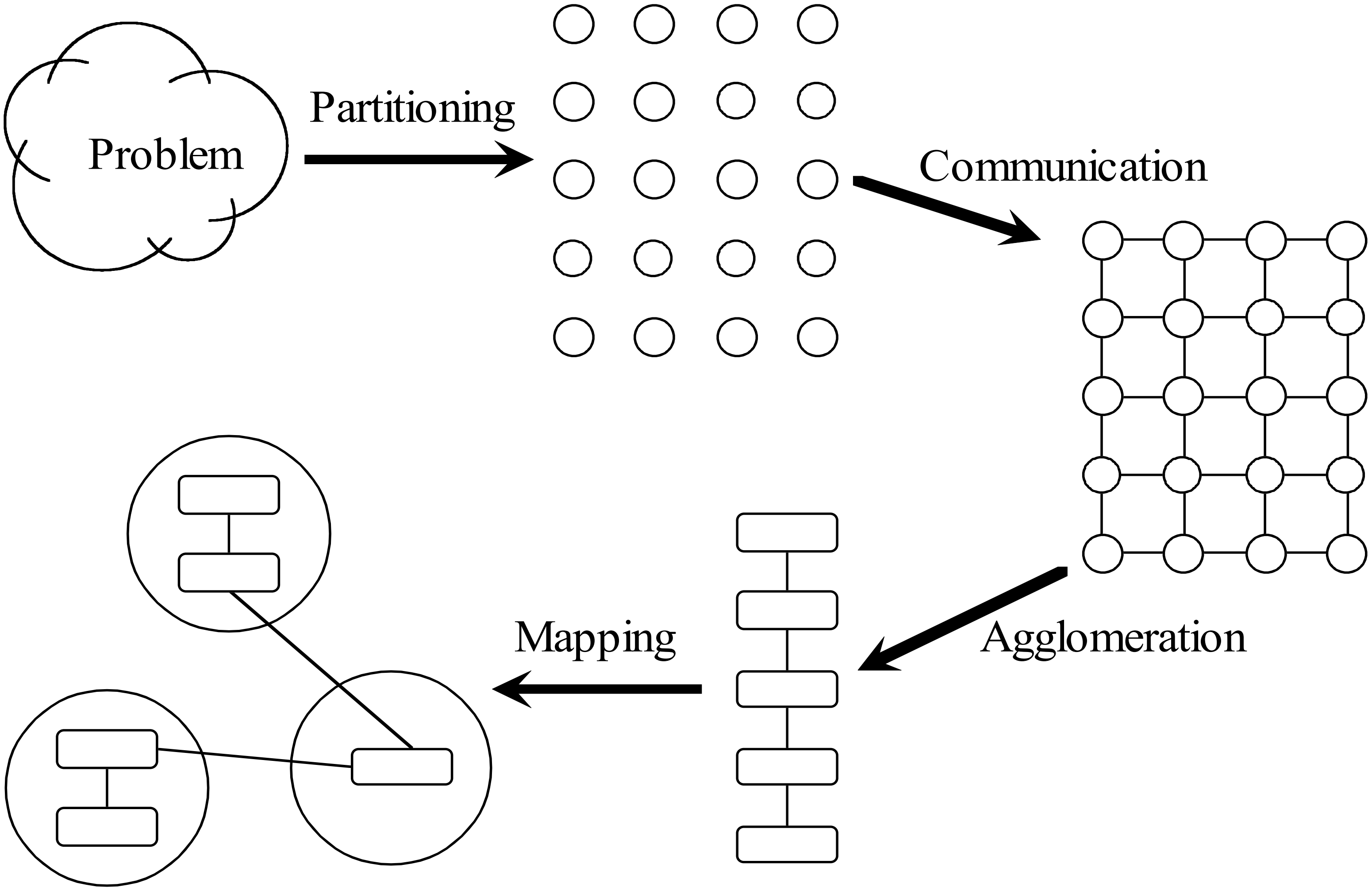
Foster's Design Methodology

- Partitioning
- Communication
- Agglomeration

Foster’s Design Methodology

- Partitioning
- Communication
- Agglomeration
- Mapping

Foster's Methodology



Partitioning

Partitioning

- Dividing computation and data into pieces

Partitioning

- Dividing computation and data into pieces
- Domain decomposition

Partitioning

- Dividing computation and data into pieces
- Domain decomposition
    - Divide data into pieces

Partitioning

- Dividing computation and data into pieces
- Domain decomposition
  - Divide data into pieces
  - Determine how to associate computations with the data

Partitioning

- Dividing computation and data into pieces
- Domain decomposition
  - Divide data into pieces
  - Determine how to associate computations with the data
- Functional decomposition

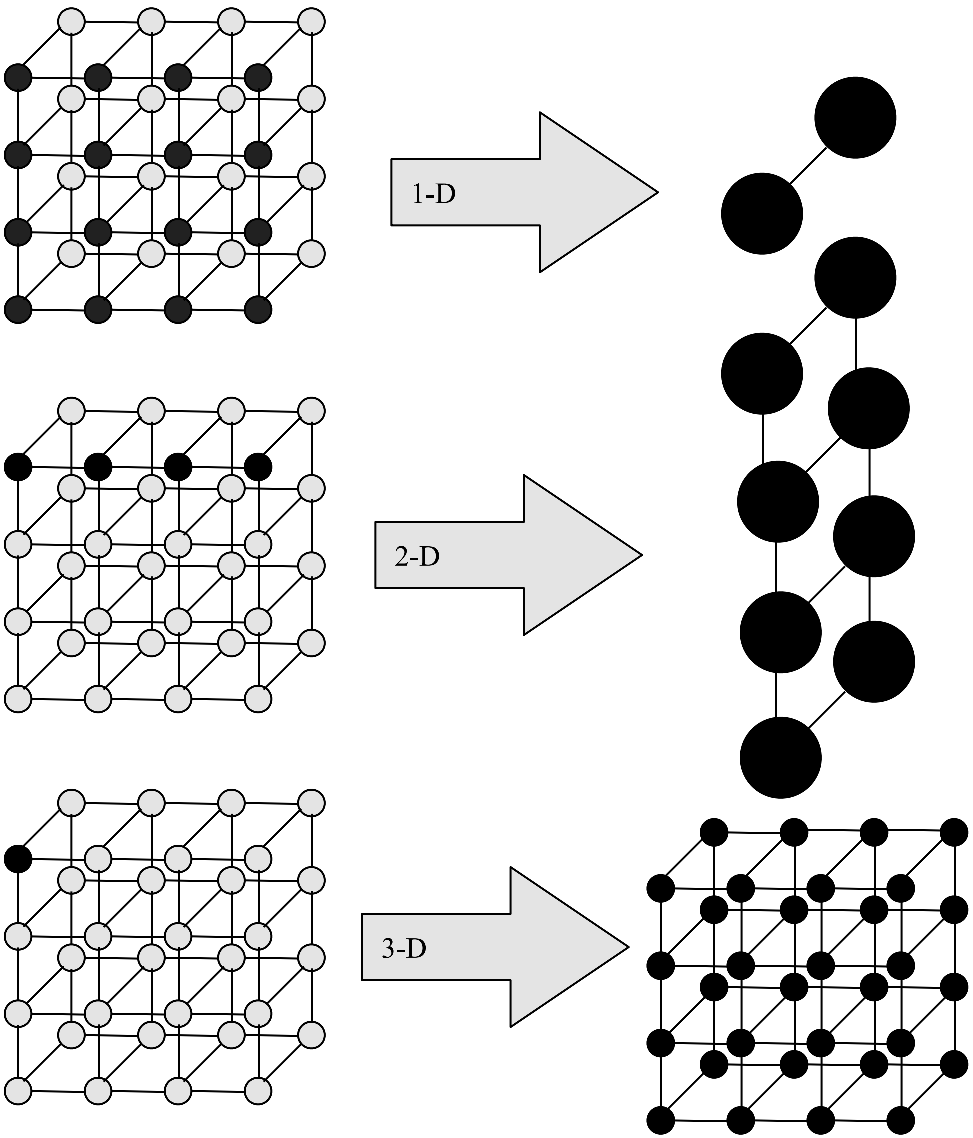
Partitioning

- Dividing computation and data into pieces
- Domain decomposition
  - Divide data into pieces
  - Determine how to associate computations with the data
- Functional decomposition
  - Divide computation into pieces

Partitioning

- Dividing computation and data into pieces
- Domain decomposition
  - Divide data into pieces
  - Determine how to associate computations with the data
- Functional decomposition
  - Divide computation into pieces
  - Determine how to associate data with the computations

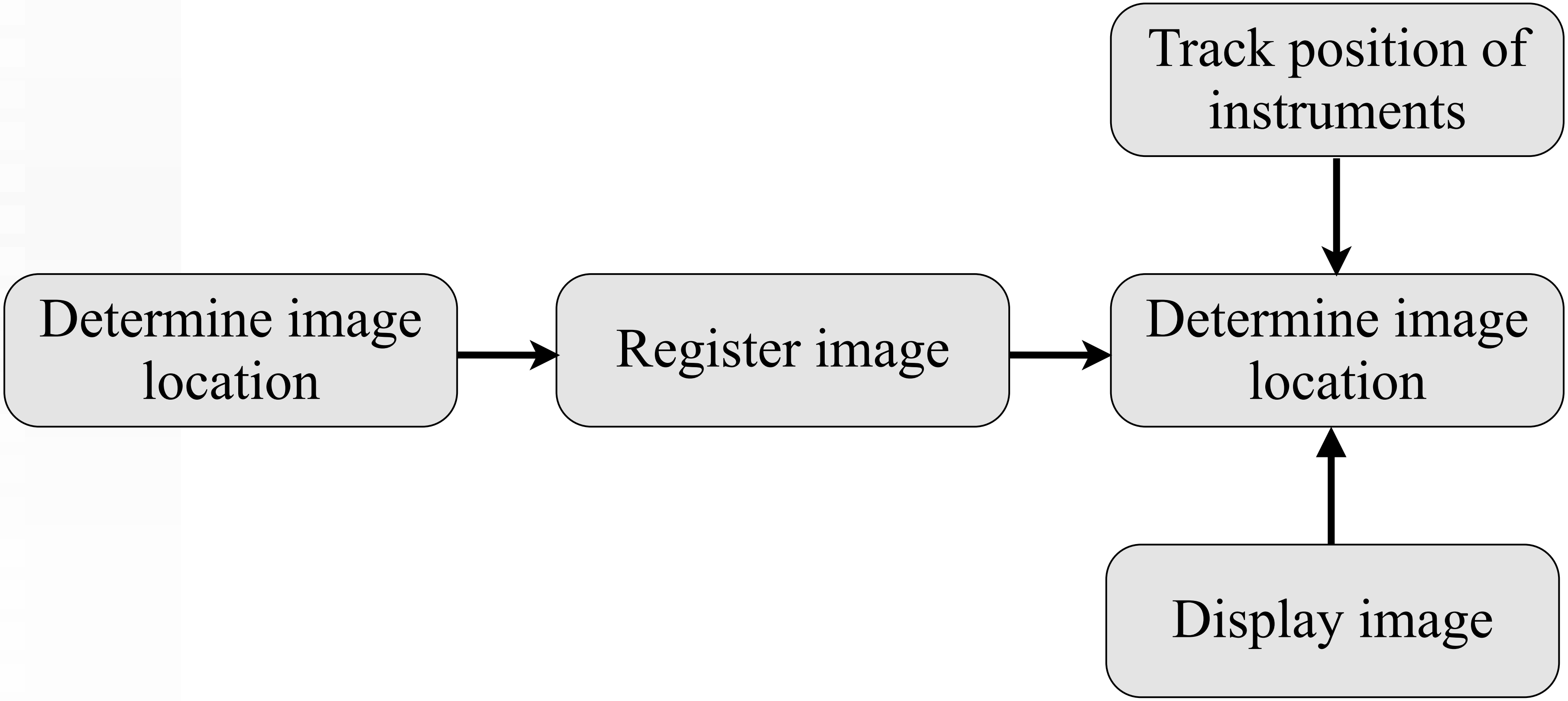
Example Domain Decompositions



Primitive tasks is the number of scope, or order of magnitude, of the parallelism.

1-D has, in the example, n-way ||ism along the n-faces, 2-D has n^2 ||ism along the faces, and 3-way has n^3 || ism along the faces.

Example Functional Decomposition



Types of parallelism

Types of parallelism

- Numerical algorithms often have data-parallelism

Types of parallelism

- Numerical algorithms often have data-parallelism
- Non-numerical algorithms often have functional parallelism.

Types of parallelism

- Numerical algorithms often have data-parallelism
- Non-numerical algorithms often have functional parallelism.
- Many algorithms, especially complex numerical algorithms, have both, e.g., data parallelism within an function, many functions that can be done in parallel.

Types of parallelism

- Numerical algorithms often have data-parallelism
- Non-numerical algorithms often have functional parallelism.
- Many algorithms, especially complex numerical algorithms, have both, e.g., data parallelism within an function, many functions that can be done in parallel.
- Functional parallelism often scales worse with increasing data size (concurrency-limited in isoefficiency terms)

Partitioning Checklist

Partitioning Checklist

- At least 10x more primitive tasks than processors in target computer

Partitioning Checklist

- At least 10x more primitive tasks than processors in target computer
- Minimize redundant computations and redundant data storage

Partitioning Checklist

- At least 10x more primitive tasks than processors in target computer
- Minimize redundant computations and redundant data storage
- Primitive tasks roughly the same size

Partitioning Checklist

- At least 10x more primitive tasks than processors in target computer
- Minimize redundant computations and redundant data storage
- Primitive tasks roughly the same size
- Number of tasks an increasing function of problem size

Communication

Communication

- Determine values passed among tasks

Communication

- Determine values passed among tasks
- Local communication

Communication

- Determine values passed among tasks
- Local communication
    - Task needs values from a small number of other tasks

Communication

- Determine values passed among tasks
- Local communication
  - Task needs values from a small number of other tasks
  - Create channels illustrating data flow

Communication

- Determine values passed among tasks
- Local communication
  - Task needs values from a small number of other tasks
  - Create channels illustrating data flow
- Global communication

Communication

- Determine values passed among tasks
- Local communication
  - Task needs values from a small number of other tasks
  - Create channels illustrating data flow
- Global communication
  - Significant number of tasks contribute data to perform a computation

Communication

- Determine values passed among tasks
- Local communication
  - Task needs values from a small number of other tasks
  - Create channels illustrating data flow
- Global communication
  - Significant number of tasks contribute data to perform a computation
  - Don’t create channels for them early in design

Communication Checklist

Communication Checklist

- Communication operations balanced among tasks

Communication Checklist

- Communication operations balanced among tasks
- Each task communicates with only small group of neighbors

Communication Checklist

- Communication operations balanced among tasks
- Each task communicates with only small group of neighbors
- Tasks can perform communications concurrently

Communication Checklist

- Communication operations balanced among tasks
- Each task communicates with only small group of neighbors
- Tasks can perform communications concurrently
- Task can perform computations concurrently

Agglomeration

Agglomeration

- Grouping tasks into larger tasks

Agglomeration

- Grouping tasks into larger tasks
- Goals

Agglomeration

- Grouping tasks into larger tasks
- Goals
  - Improve performance

Agglomeration

- Grouping tasks into larger tasks
- Goals
  - Improve performance
  - Maintain scalability of program

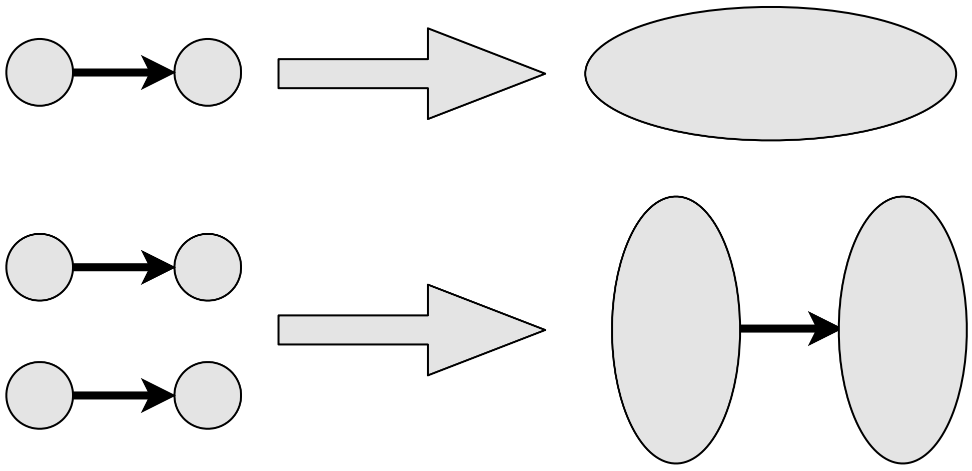
Agglomeration

- Grouping tasks into larger tasks
- Goals
  - Improve performance
  - Maintain scalability of program
  - Simplify programming

Agglomeration

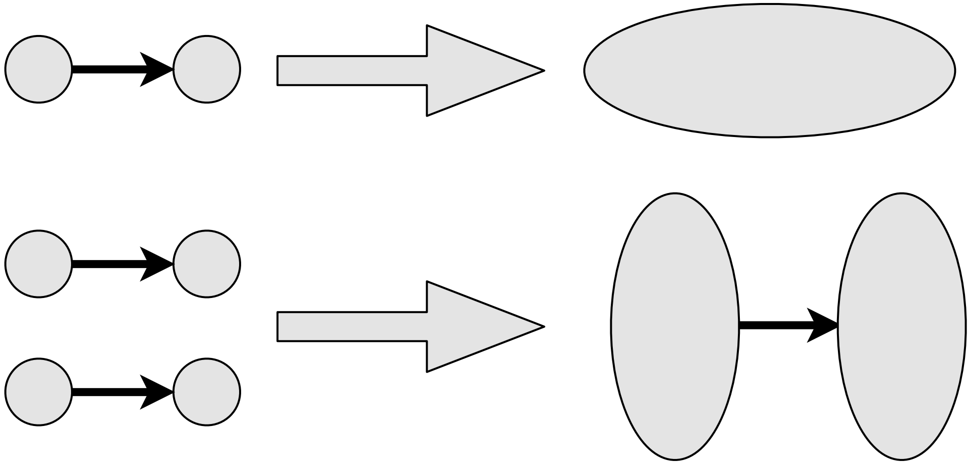
- Grouping tasks into larger tasks
- Goals
  - Improve performance
  - Maintain scalability of program
  - Simplify programming
- In MPI programming, goal often to create one agglomerated task per processor

Agglomeration Can Improve Performance



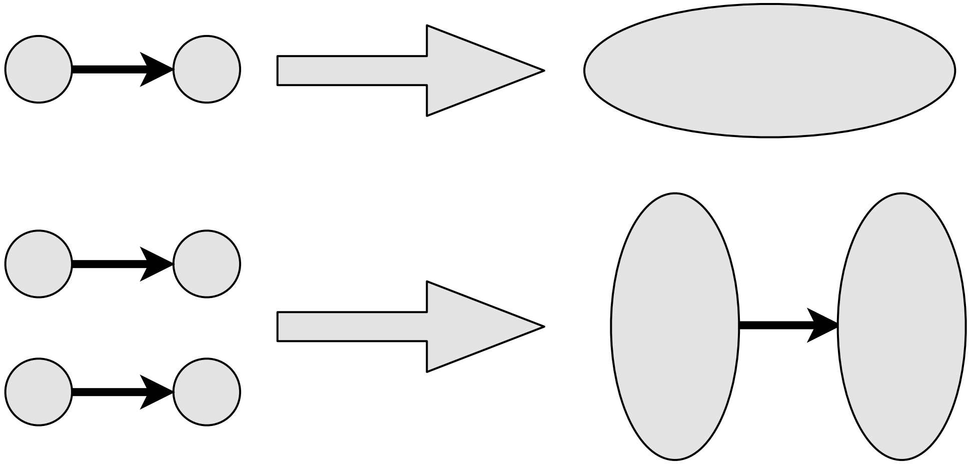
Agglomeration Can Improve Performance

- Eliminate communication between primitive tasks agglomerated into consolidated task



Agglomeration Can Improve Performance

- Eliminate communication between primitive tasks agglomerated into consolidated task
- Combine groups of sending and receiving tasks



Agglomeration Checklist

Agglomeration Checklist

- Locality of parallel algorithm has increased

Agglomeration Checklist

- Locality of parallel algorithm has increased
- Replicated computations take less time than communications they replace

Agglomeration Checklist

- Locality of parallel algorithm has increased
- Replicated computations take less time than communications they replace
- Data replication doesn’t affect scalability

Agglomeration Checklist

- Locality of parallel algorithm has increased
- Replicated computations take less time than communications they replace
- Data replication doesn’t affect scalability
- Agglomerated tasks have similar computational and communications costs

Agglomeration Checklist

- Locality of parallel algorithm has increased
- Replicated computations take less time than communications they replace
- Data replication doesn’t affect scalability
- Agglomerated tasks have similar computational and communications costs
- Number of tasks increases with problem size

Agglomeration Checklist

- Locality of parallel algorithm has increased
- Replicated computations take less time than communications they replace
- Data replication doesn't affect scalability
- Agglomerated tasks have similar computational and communications costs
- Number of tasks increases with problem size
- Number of tasks suitable for likely target systems

Agglomeration Checklist

- Locality of parallel algorithm has increased
- Replicated computations take less time than communications they replace
- Data replication doesn’t affect scalability
- Agglomerated tasks have similar computational and communications costs
- Number of tasks increases with problem size
- Number of tasks suitable for likely target systems
- Tradeoff between agglomeration and code modifications costs is reasonable

Mapping

Mapping

- Process of assigning tasks to processors

Mapping

- Process of assigning tasks to processors
- Shared memory system: mapping done by operating system

Mapping

- Process of assigning tasks to processors
- Shared memory system: mapping done by operating system
- Distributed memory system: mapping done by user

Mapping

- Process of assigning tasks to processors
- Shared memory system: mapping done by operating system
- Distributed memory system: mapping done by user
- Conflicting goals of mapping

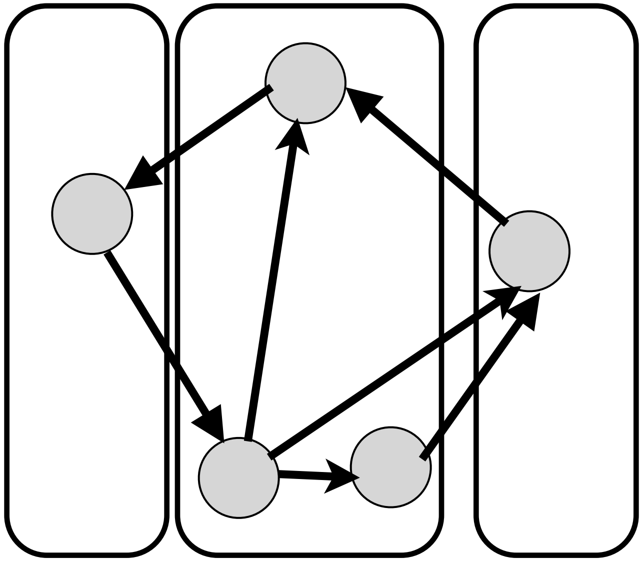
Mapping

- Process of assigning tasks to processors
- Shared memory system: mapping done by operating system
- Distributed memory system: mapping done by user
- Conflicting goals of mapping
  - Maximize processor utilization

Mapping

- Process of assigning tasks to processors
- Shared memory system: mapping done by operating system
- Distributed memory system: mapping done by user
- Conflicting goals of mapping
    - Maximize processor utilization
    - Minimize interprocessor communication

Mapping Example



While this may reduce communication, load balance may be an issue

Optimal Mapping

Optimal Mapping

- Finding optimal mapping is NP-hard

Optimal Mapping

- Finding optimal mapping is NP-hard
- Must rely on heuristics

Optimal Mapping

- Finding optimal mapping is NP-hard
- Must rely on heuristics
- Metis is a popular package for partitioning graphs

Optimal Mapping

- Finding optimal mapping is NP-hard
- Must rely on heuristics
- Metis is a popular package for partitioning graphs
    - Minimizes the number of edges between nodes in a graph

Optimal Mapping

- Finding optimal mapping is NP-hard
- Must rely on heuristics
- Metis is a popular package for partitioning graphs
  - Minimizes the number of edges between nodes in a graph
  - Edges, for our purposes, can be thought of as communication

Mapping Decision Tree

Mapping Decision Tree

- Static number of tasks

Mapping Decision Tree

- Static number of tasks
  - Structured communication

Mapping Decision Tree

- Static number of tasks
  - Structured communication
    - Constant computation time per task

Mapping Decision Tree

- Static number of tasks
  - Structured communication
    - Constant computation time per task
      - Agglomerate tasks to minimize communication

Mapping Decision Tree

- Static number of tasks
    - Structured communication
        - Constant computation time per task
            - Agglomerate tasks to minimize communication
            - Create one task per processor

Mapping Decision Tree

- Static number of tasks
    - Structured communication
        - Constant computation time per task
            - Agglomerate tasks to minimize communication
            - Create one task per processor
        - Variable computation time per task

Mapping Decision Tree

- Static number of tasks
    - Structured communication
        - Constant computation time per task
            - Agglomerate tasks to minimize communication
            - Create one task per processor
        - Variable computation time per task
            - Cyclically map tasks to processors

Mapping Decision Tree

- Static number of tasks
  - Structured communication
    - Constant computation time per task
      - Agglomerate tasks to minimize communication
      - Create one task per processor
    - Variable computation time per task
      - Cyclically map tasks to processors
      - GSS (guided self scheduling)

Mapping Decision Tree

- Static number of tasks
  - Structured communication
    - Constant computation time per task
      - Agglomerate tasks to minimize communication
      - Create one task per processor
    - Variable computation time per task
      - Cyclically map tasks to processors
      - GSS (guided self scheduling)
  - Unstructured communication

Mapping Decision Tree

- Static number of tasks
  - Structured communication
    - Constant computation time per task
      - Agglomerate tasks to minimize communication
      - Create one task per processor
    - Variable computation time per task
      - Cyclically map tasks to processors
      - GSS (guided self scheduling)
  - Unstructured communication
      - Use a static load balancing algorithm

Mapping Decision Tree

- Static number of tasks
  - Structured communication
    - Constant computation time per task
      - Agglomerate tasks to minimize communication
      - Create one task per processor
    - Variable computation time per task
      - Cyclically map tasks to processors
      - GSS (guided self scheduling)
  - Unstructured communication
      - Use a static load balancing algorithm
- Dynamic number of tasks

Mapping Strategy

Mapping Strategy

- Static number of tasks

Mapping Strategy

- Static number of tasks
- Dynamic number of tasks

Mapping Strategy

- Static number of tasks
- Dynamic number of tasks
  - Frequent communications between tasks

Mapping Strategy

- Static number of tasks
- Dynamic number of tasks
  - Frequent communications between tasks
    - Use a dynamic load balancing algorithm

Mapping Strategy

- Static number of tasks
- Dynamic number of tasks
  - Frequent communications between tasks
    - Use a dynamic load balancing algorithm
  - Many short-lived tasks

Mapping Strategy

- Static number of tasks
- Dynamic number of tasks
  - Frequent communications between tasks
    - Use a dynamic load balancing algorithm
  - Many short-lived tasks
    - Use a run-time task-scheduling algorithm

Mapping Strategy

- Static number of tasks
- Dynamic number of tasks
  - Frequent communications between tasks
    - Use a dynamic load balancing algorithm
  - Many short-lived tasks
    - Use a run-time task-scheduling algorithm
    - Cilk and Galois, discussed in the next couple of weeks, do this.

Mapping Checklist

Mapping Checklist

- Considered designs based on one task per processor and multiple tasks per processor

Mapping Checklist

- Considered designs based on one task per processor and multiple tasks per processor
- Evaluated static and dynamic task allocation

Mapping Checklist

- Considered designs based on one task per processor and multiple tasks per processor
- Evaluated static and dynamic task allocation
- If dynamic task allocation chosen, task allocator is not a bottleneck to performance

Mapping Checklist

- Considered designs based on one task per processor and multiple tasks per processor
- Evaluated static and dynamic task allocation
- If dynamic task allocation chosen, task allocator is not a bottleneck to performance
- If static task allocation chosen, ratio of tasks to processors is at least 10:1

Case Studies

Case Studies

- Boundary value problem

Case Studies

- Boundary value problem
- Finding the maximum

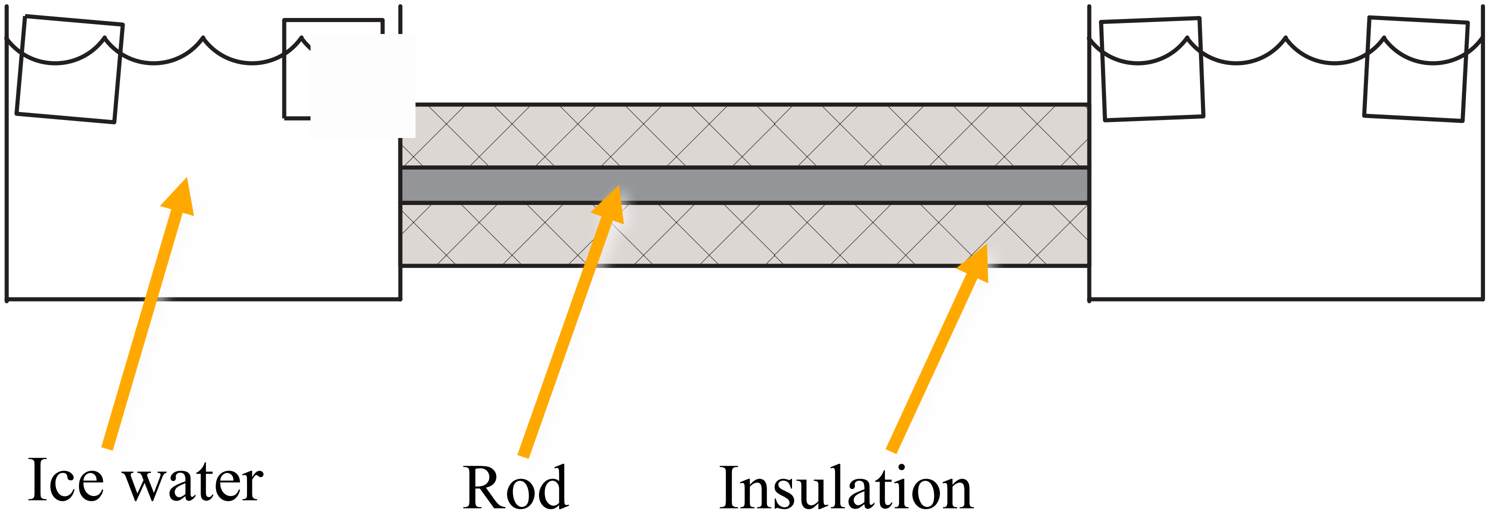
Case Studies

- Boundary value problem
- Finding the maximum
- The n-body problem

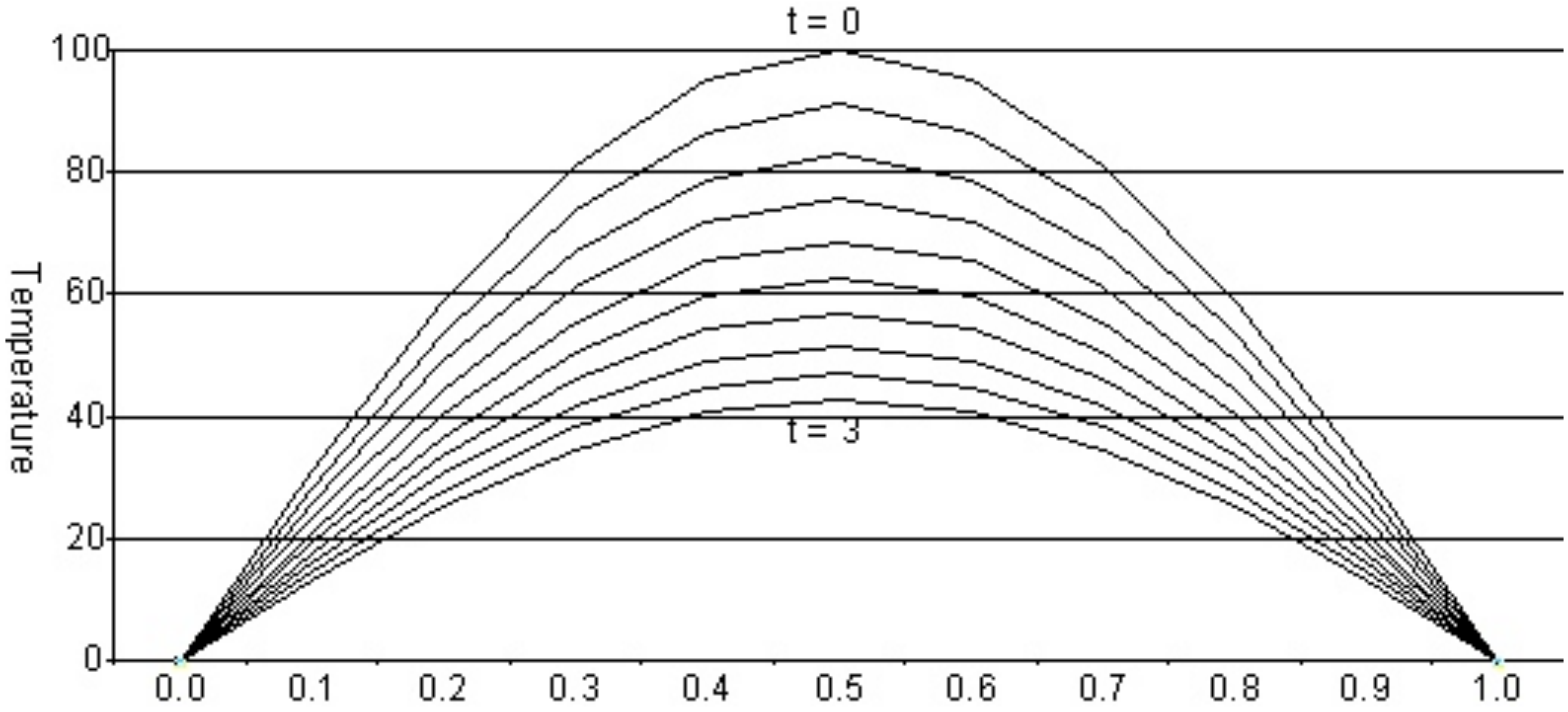
Case Studies

- Boundary value problem
- Finding the maximum
- The n-body problem
- Adding data input

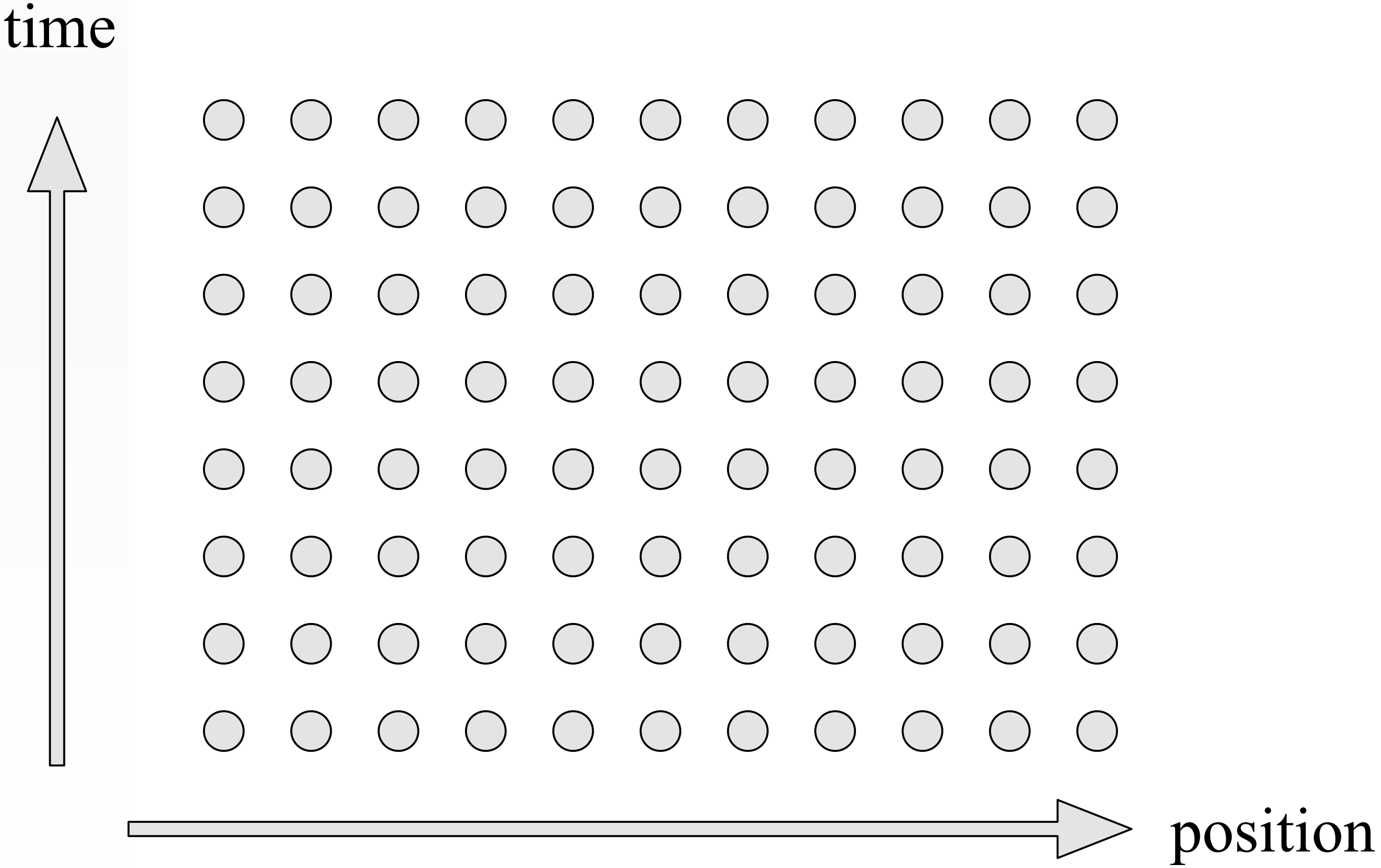
Boundary Value Problem



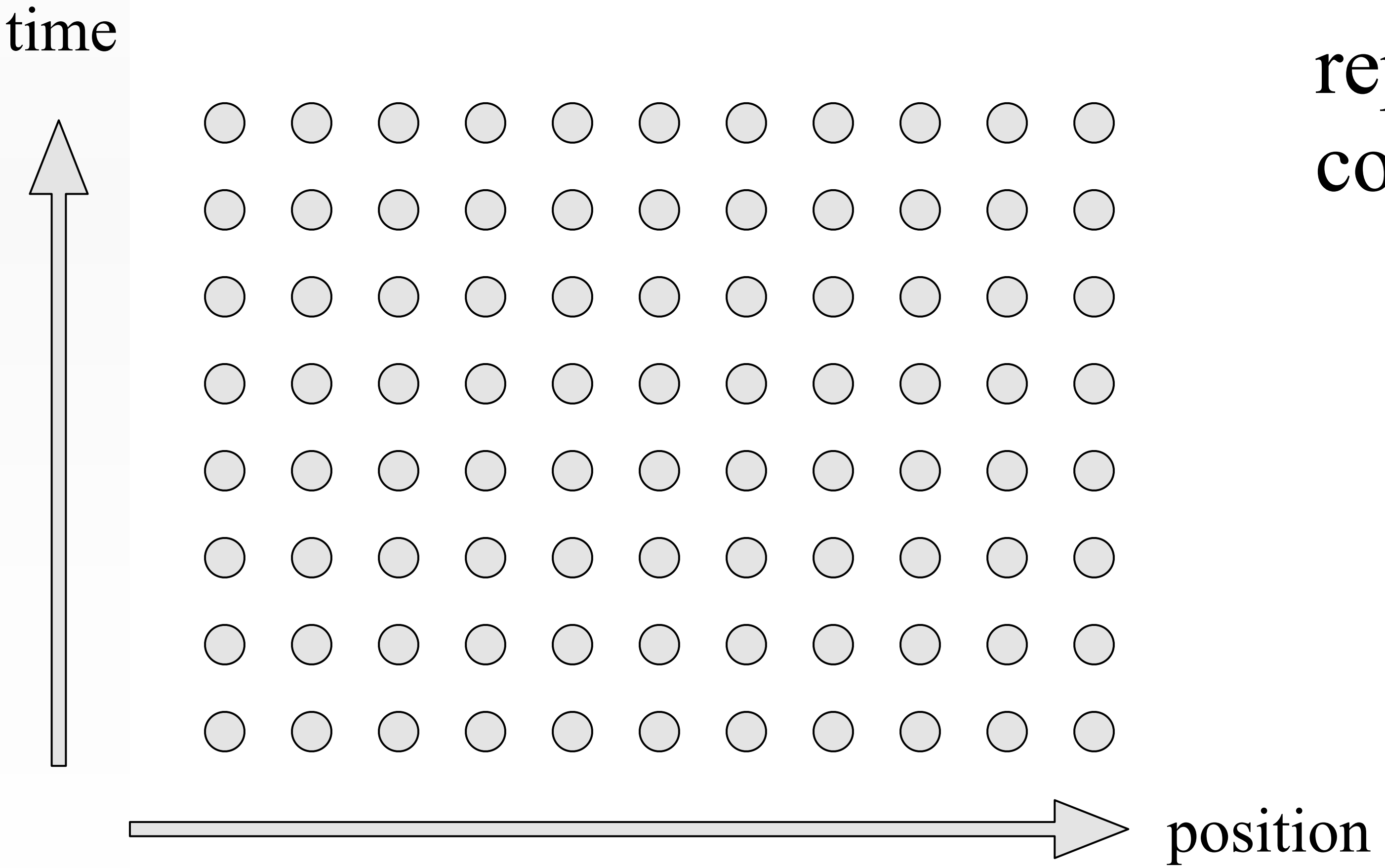
Rod Cools as Time Progresses



Want to use finite-difference method over multiple time steps

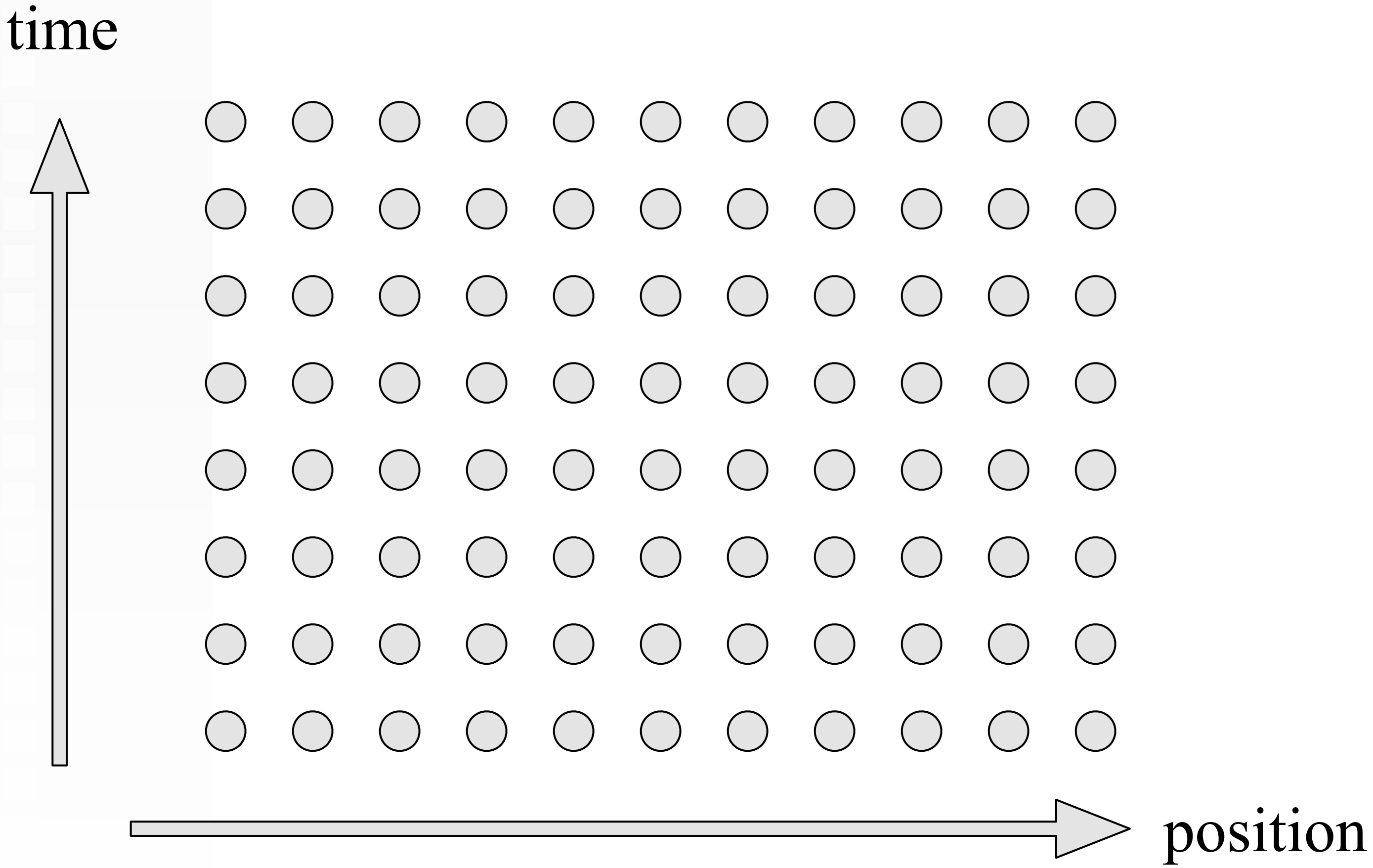


Want to use finite-difference method over multiple time steps

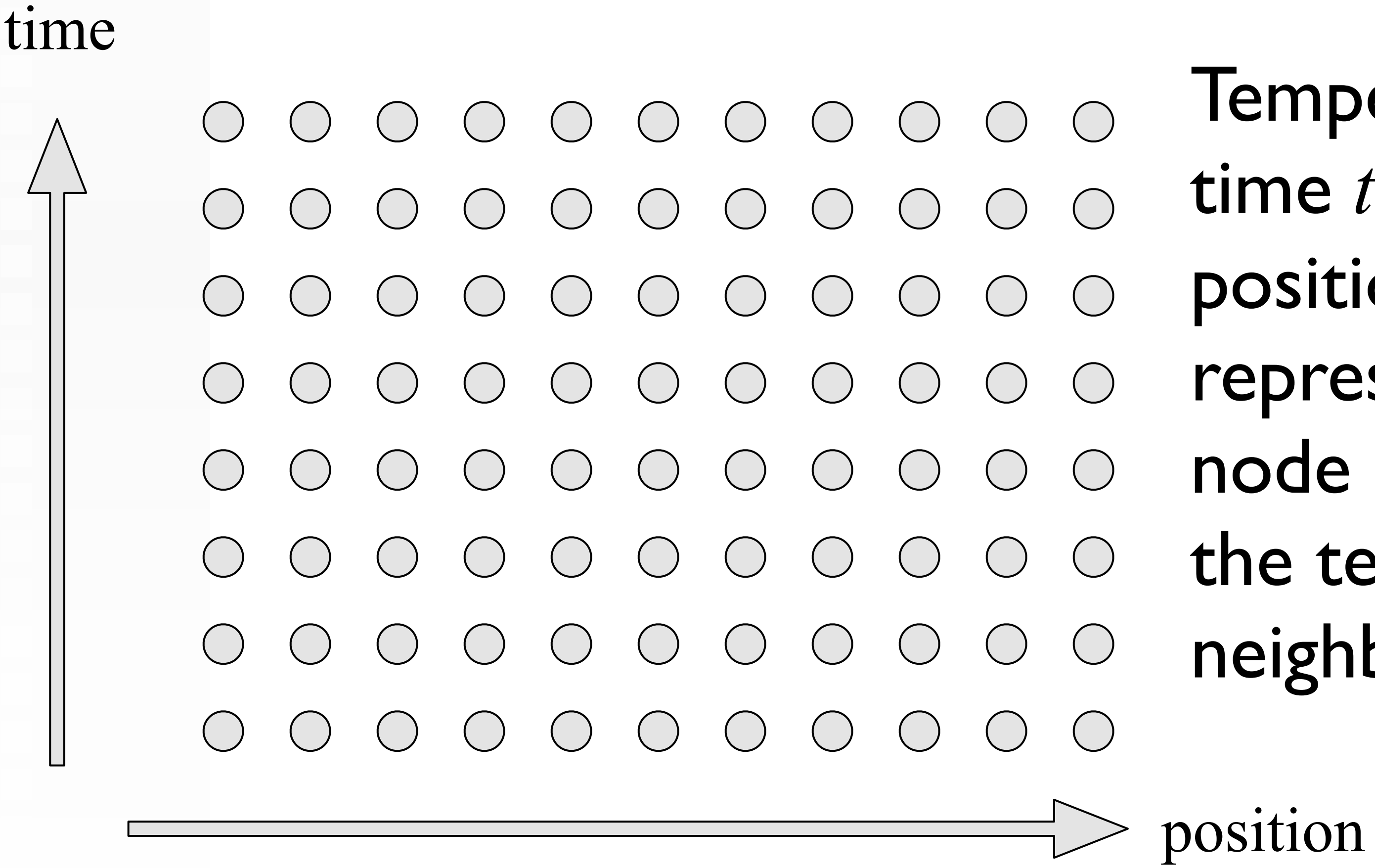


- Each circle represents a computation

Want to use finite-difference method over multiple time steps

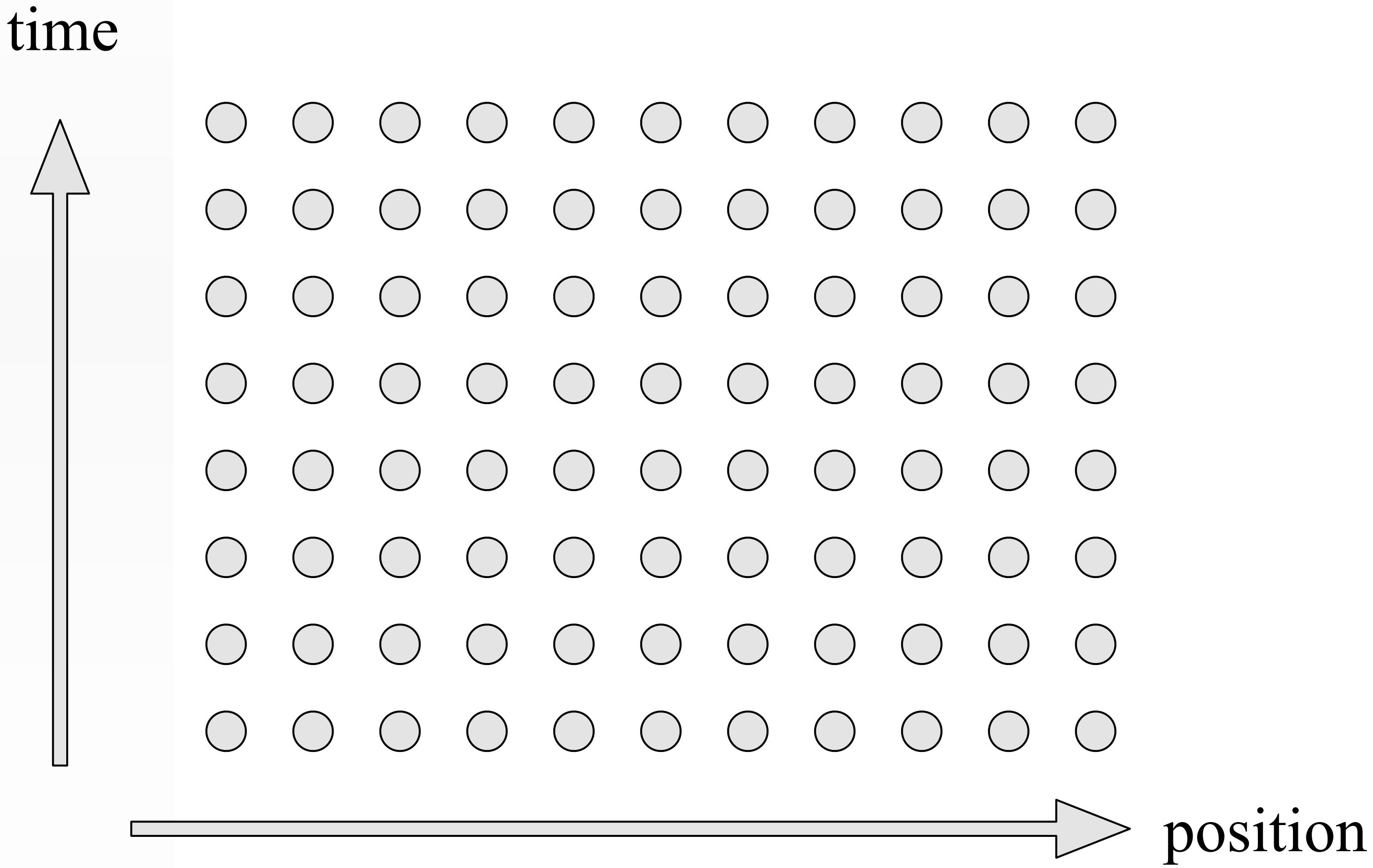


Want to use finite-difference method over multiple time steps



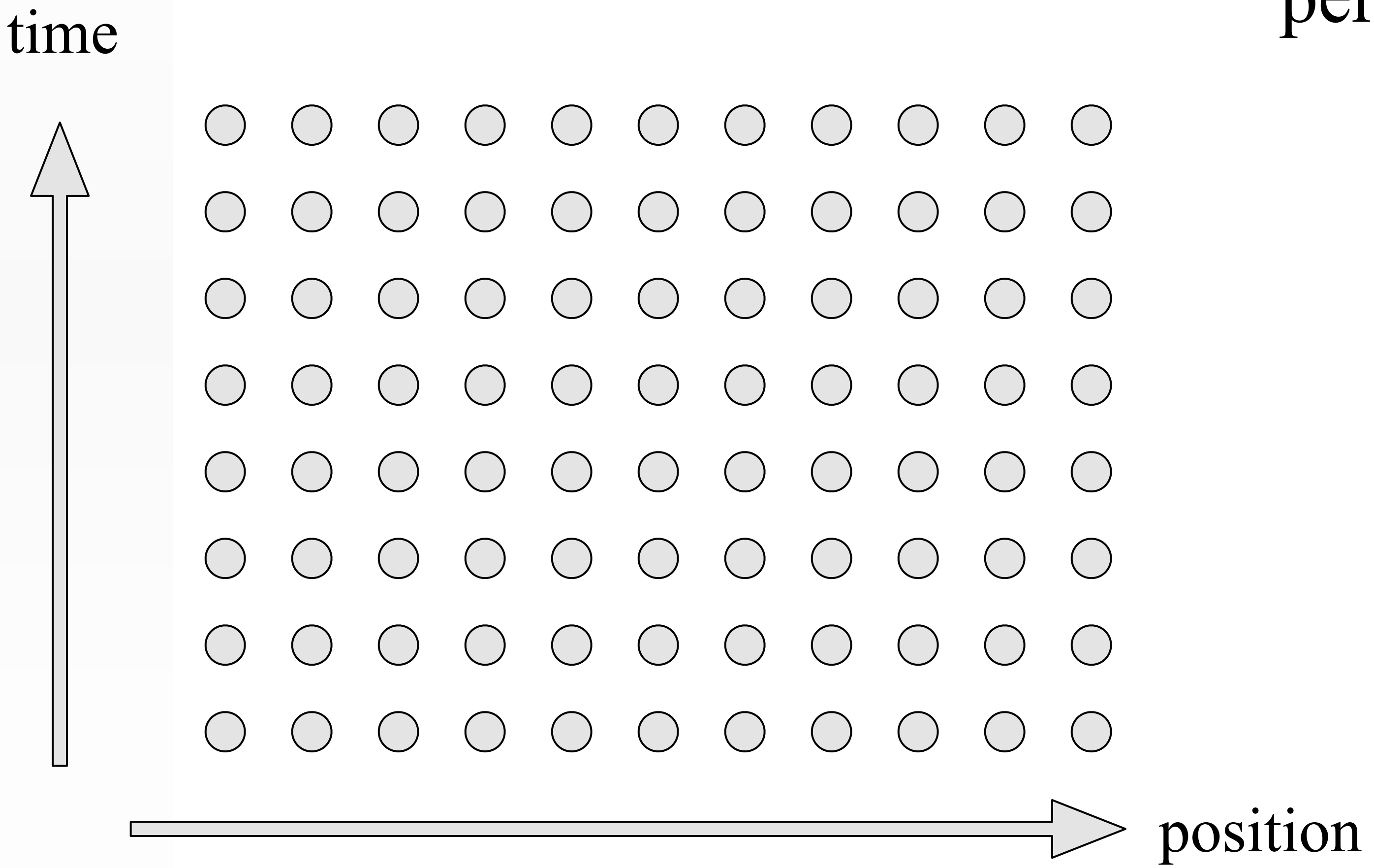
Temperature at time *t+1* for a position on the rod represented by a node depends on the temperature of neighbors at time *t*

Partitioning

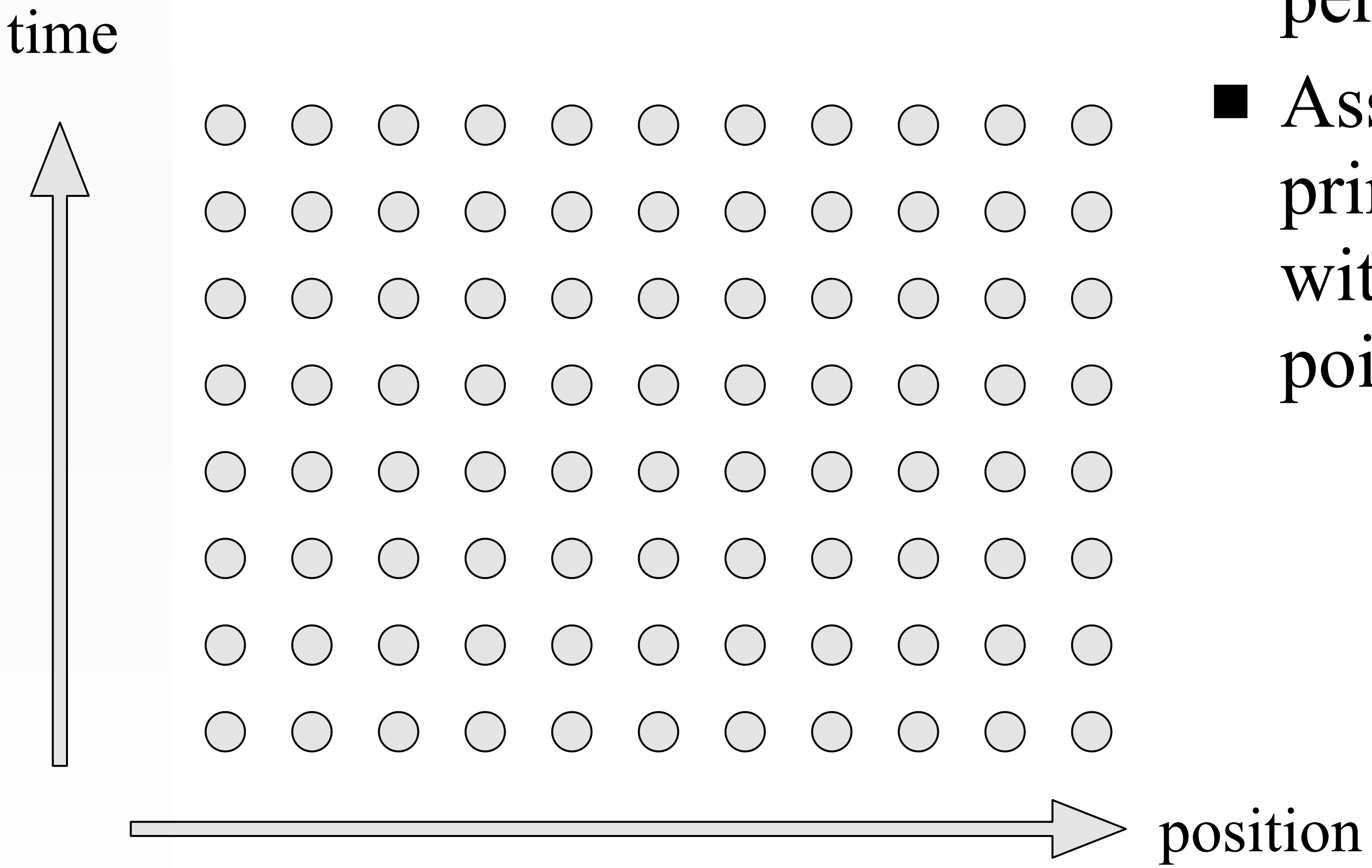


Partitioning

- One data item per grid point

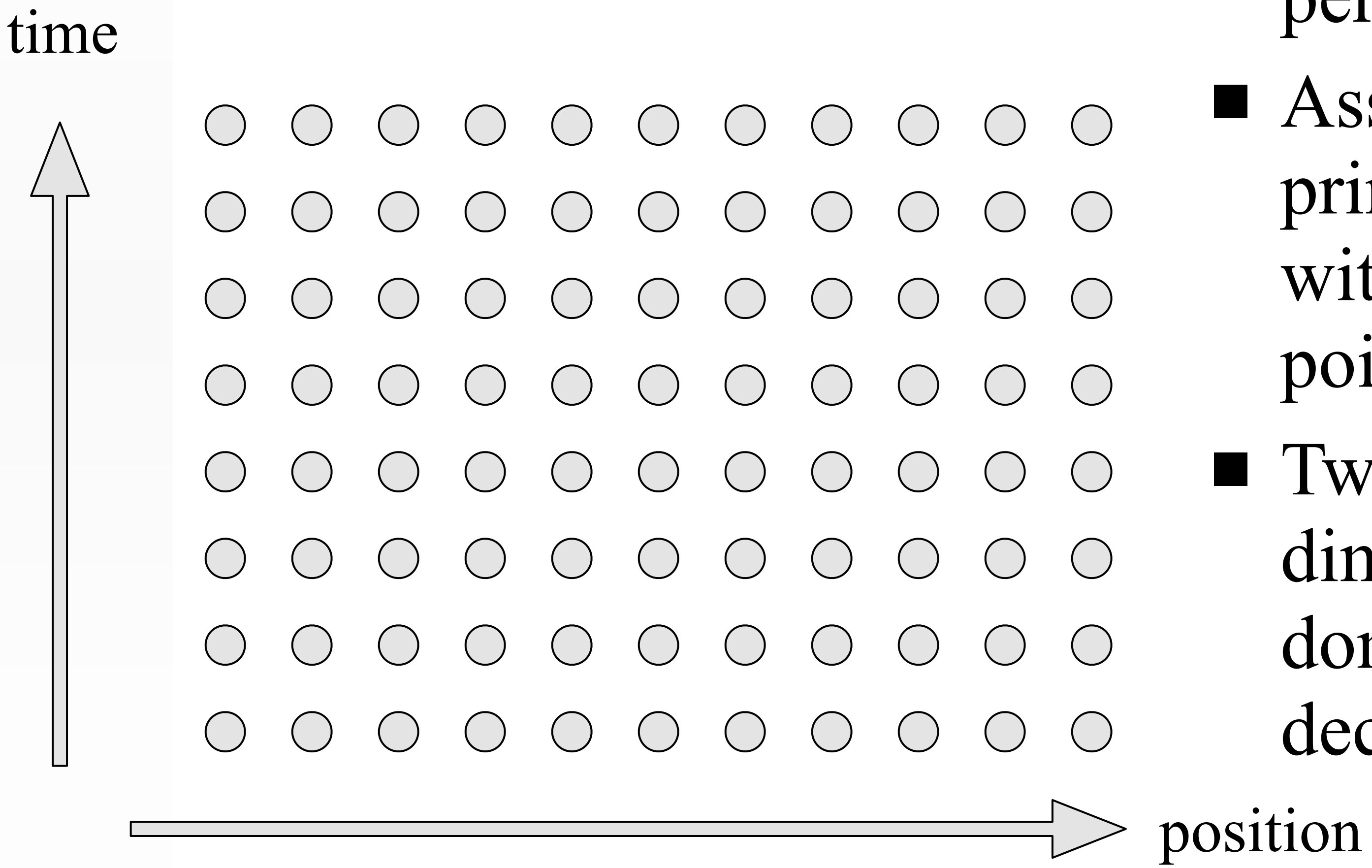


Partitioning



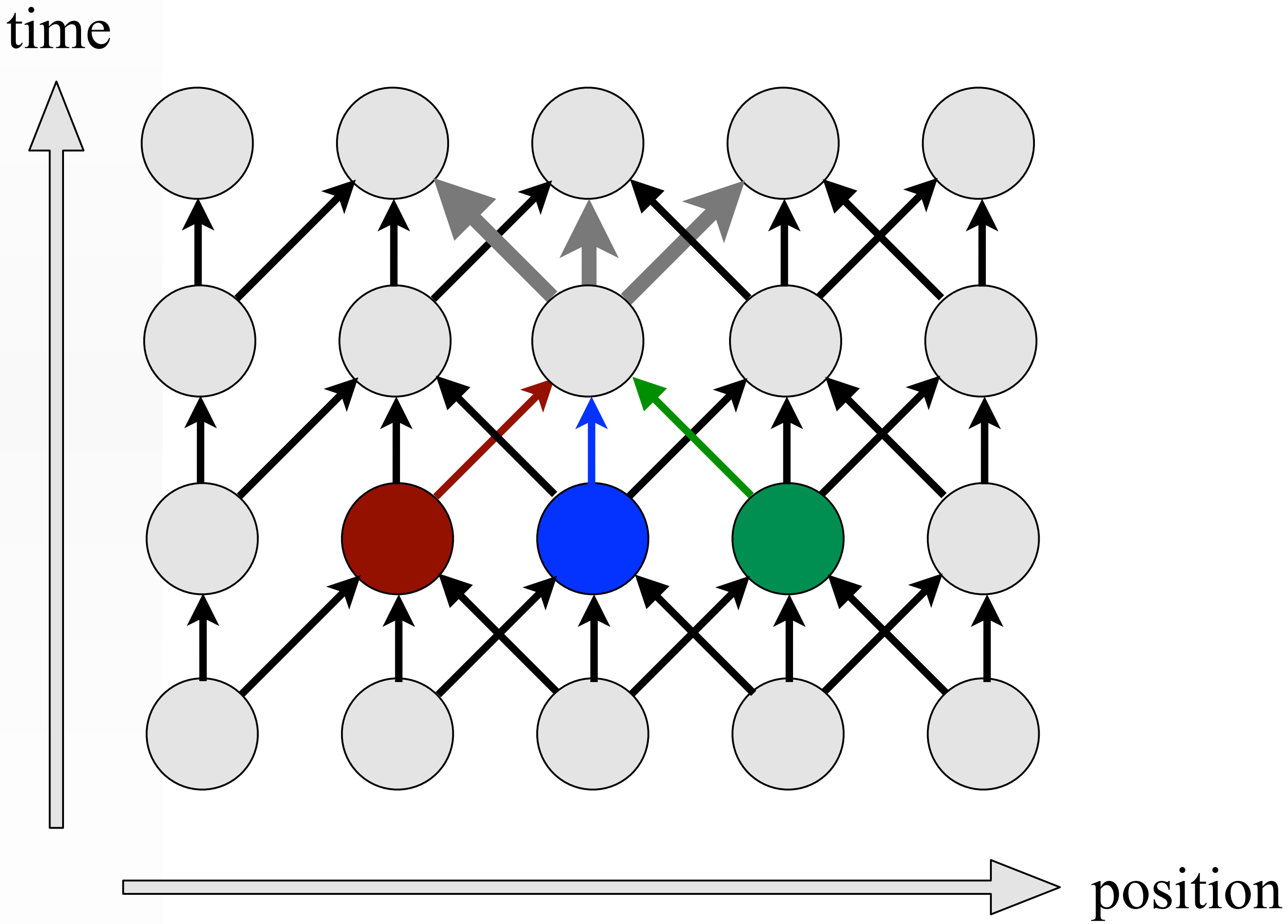
- One data item per grid point
- Associate one primitive task with each grid point

Partitioning

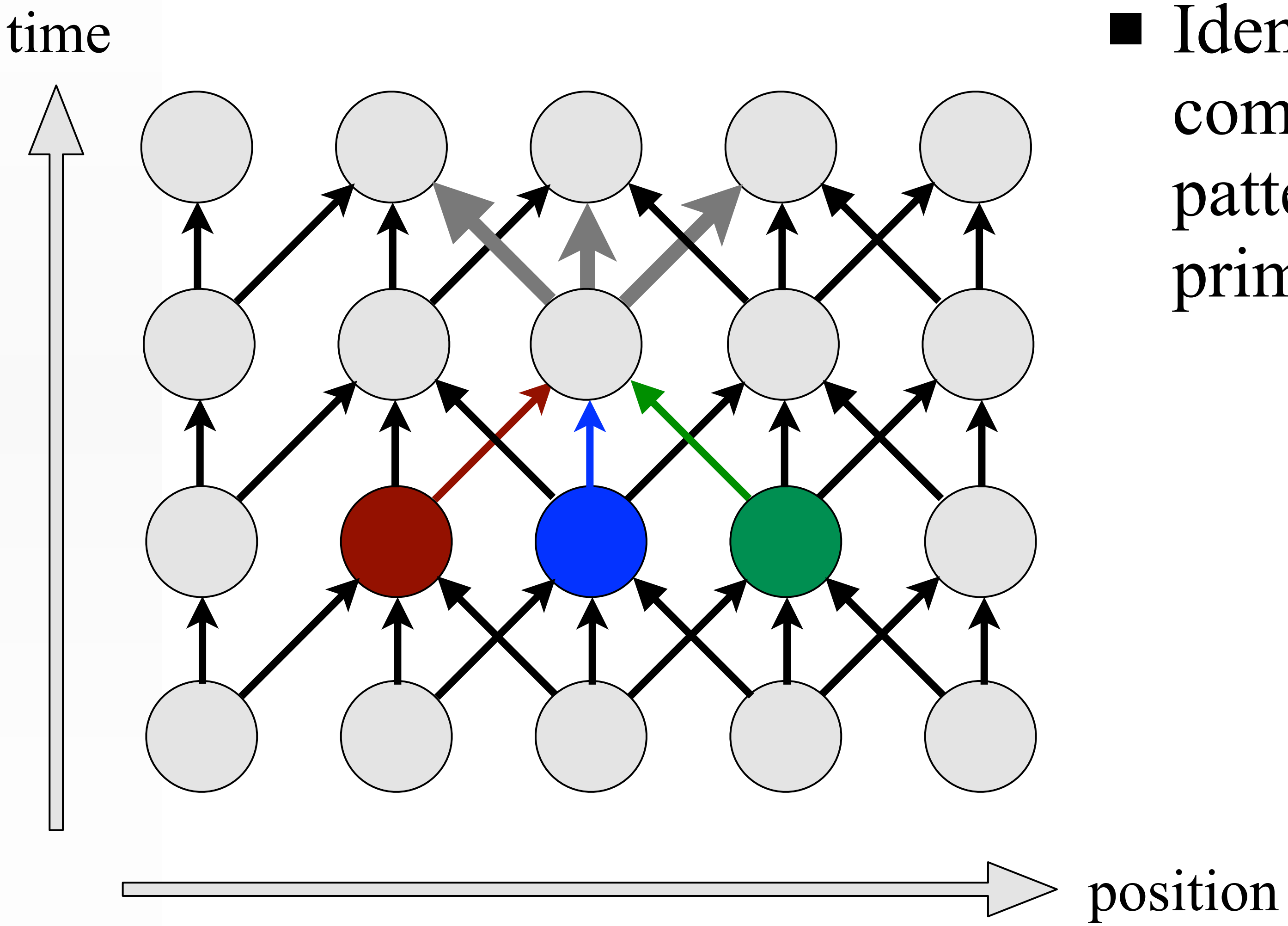


- One data item per grid point
- Associate one primitive task with each grid point
- Two-dimensional domain decomposition

Communication

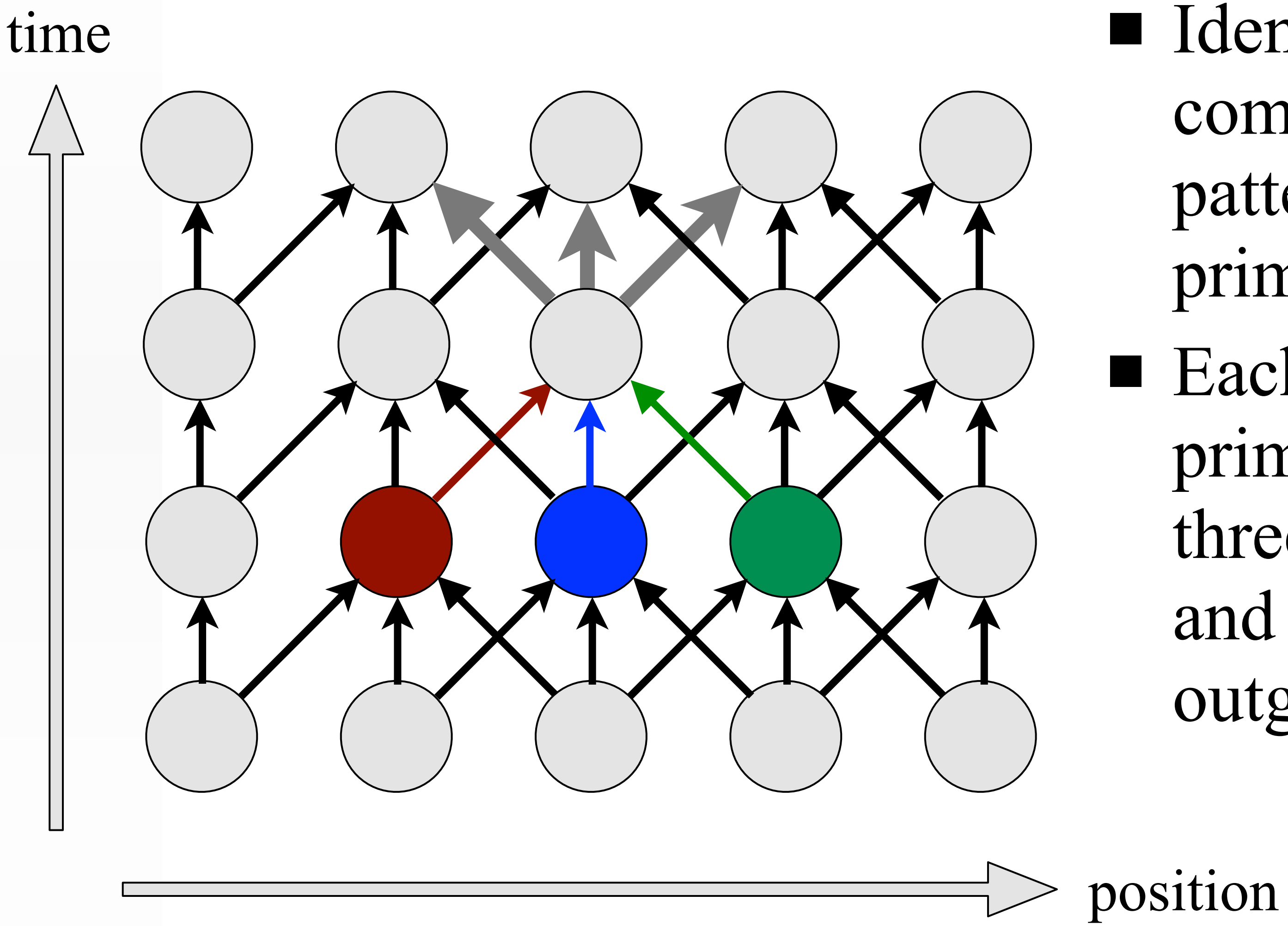


Communication



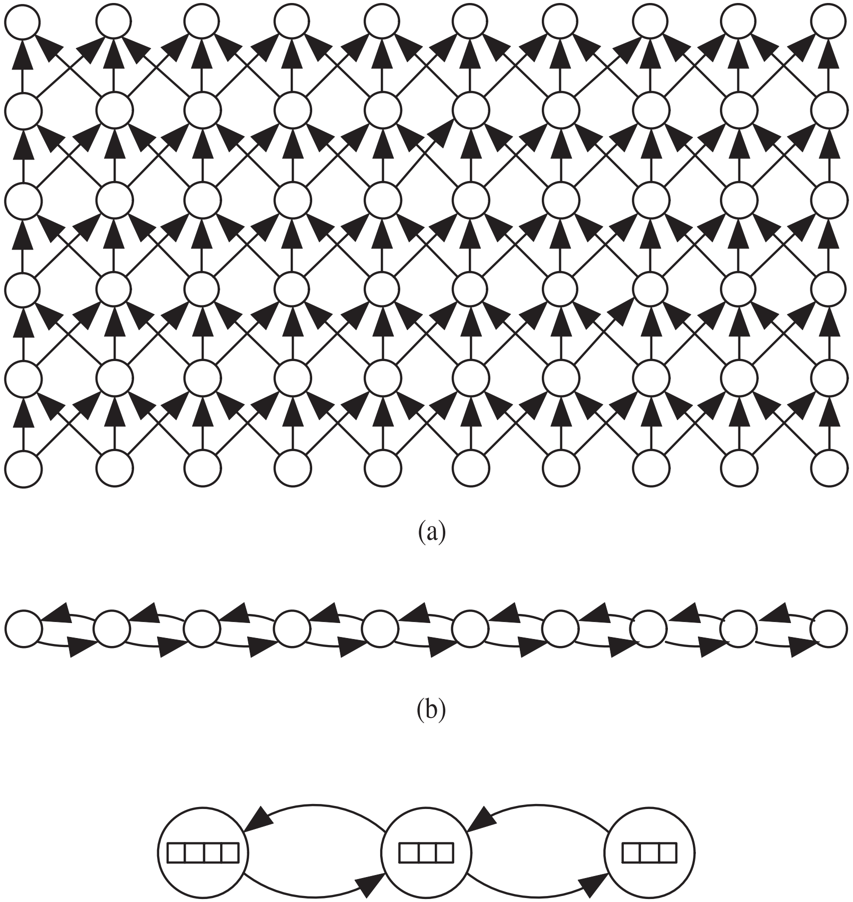
- Identify communication pattern between primitive tasks

Communication



- Identify communication pattern between primitive tasks
- Each interior primitive task has three incoming and three outgoing channels

Agglomeration and Mapping

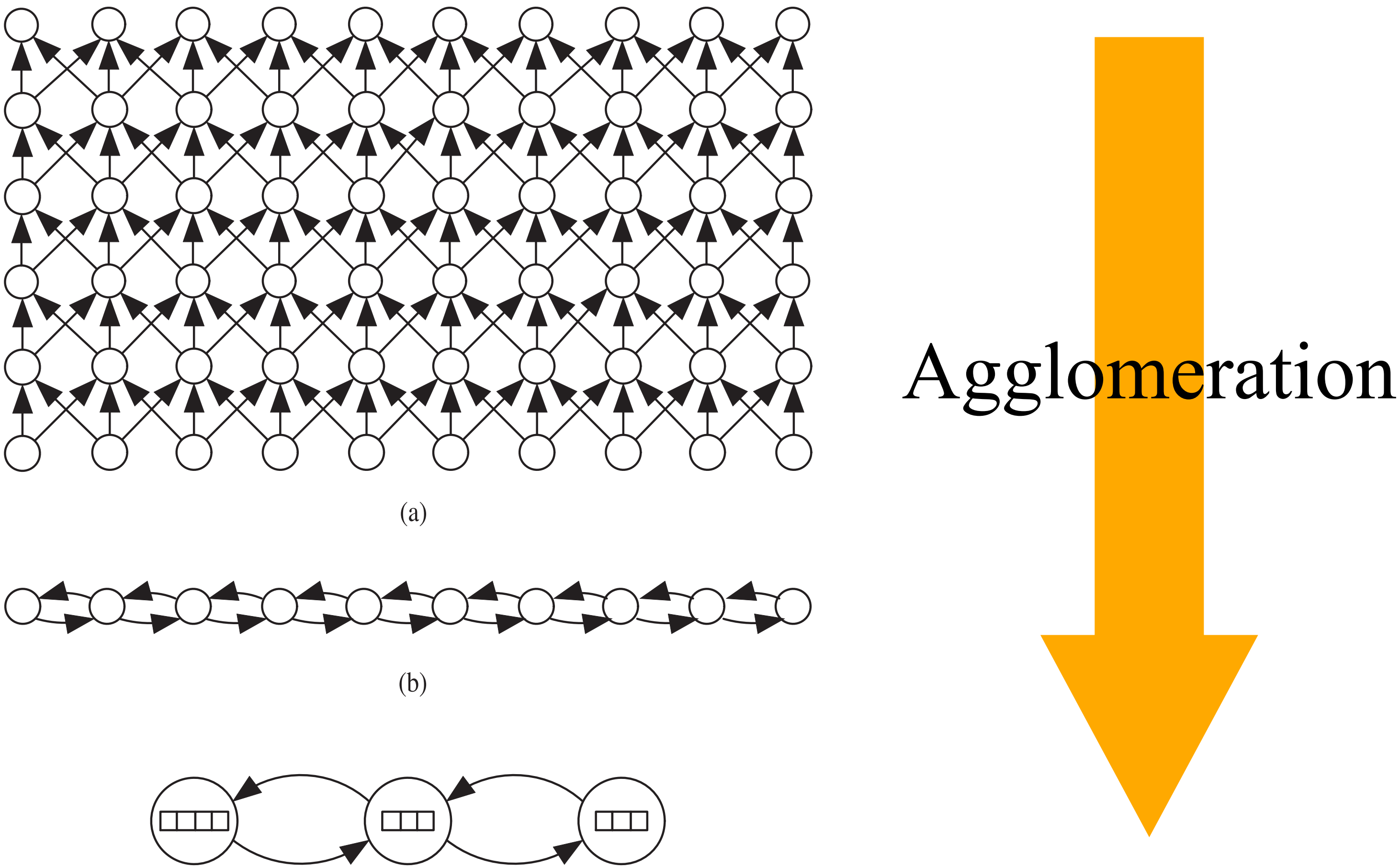


(a)

(b)

(c)

Agglomeration and Mapping



(a)

(b)

(c)

Agglomeration

Sequential execution time

Sequential execution time

- χ – time to update element

Sequential execution time

- χ – time to update element
- n – number of elements

Sequential execution time

- χ – time to update element
- n – number of elements
- m – number of iterations

Sequential execution time

- χ – time to update element
- n – number of elements
- m – number of iterations
- Sequential execution time*:* $m\ (n\text{-}1)\ \chi$

Parallel Execution Time

Parallel Execution Time

- χ – time to update element

Parallel Execution Time

- χ – time to update element
- n – number of elements

Parallel Execution Time

- χ – time to update element
- n – number of elements
- m – number of iterations

Parallel Execution Time

- χ – time to update element
- n – number of elements
- m – number of iterations
- *p* – number of processors

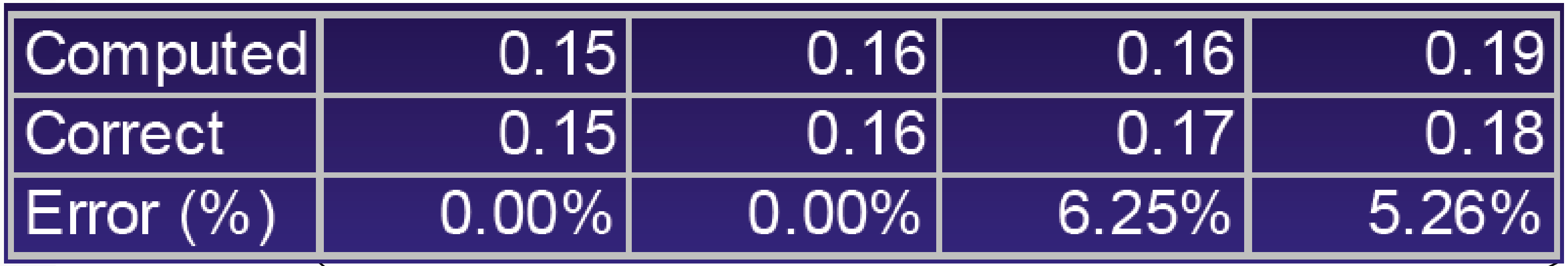
Parallel Execution Time

- χ – time to update element
- n – number of elements
- m – number of iterations
- *p* – number of processors
- λ – message latency

Parallel Execution Time

- χ – time to update element
- n – number of elements
- m – number of iterations
- p – number of processors
- λ – message latency
- Parallel execution time $m(\chi\lceil(n\text{-}1)/p\rceil+2\lambda)$

Finding the Maximum Error from measured data

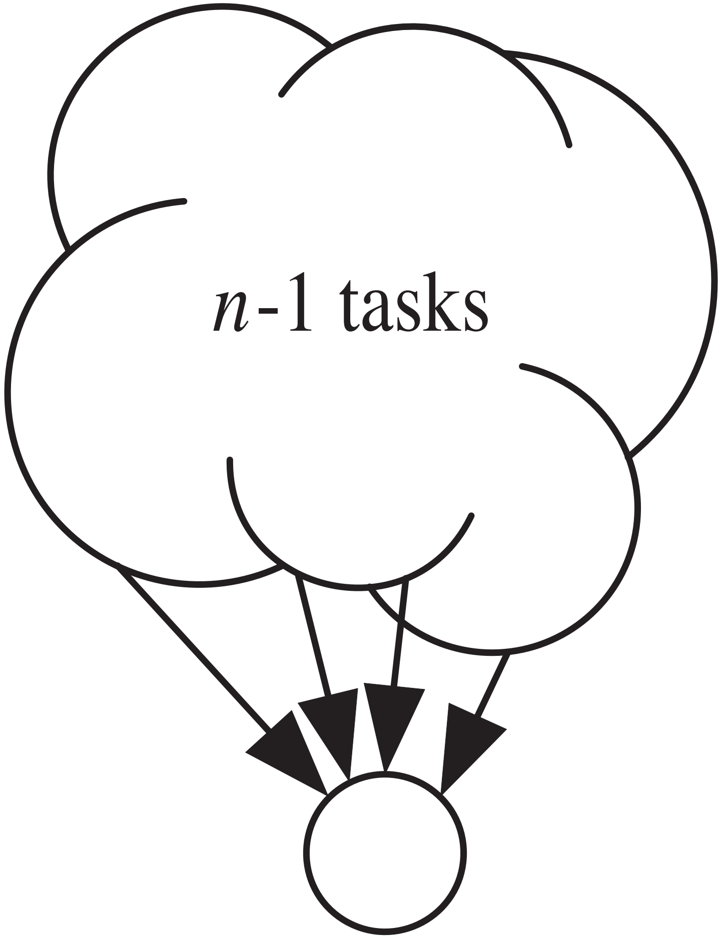


Computed	0.15	0.16	0.16	0.19
Correct	0.15	0.16	0.17	0.18
Error (%)	0.00%	0.00%	6.25%	5.26%

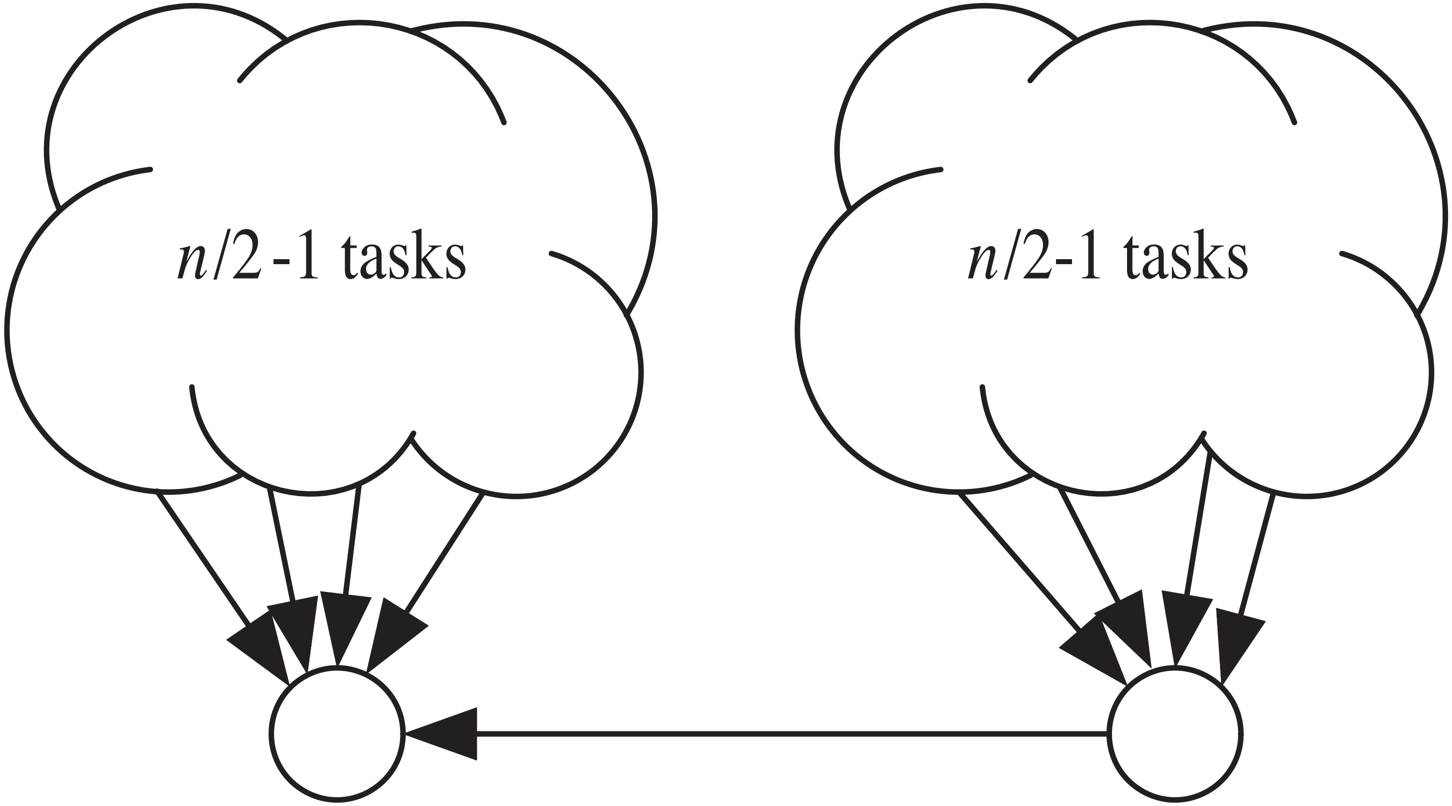
Need to do
a reduction.

6.25%

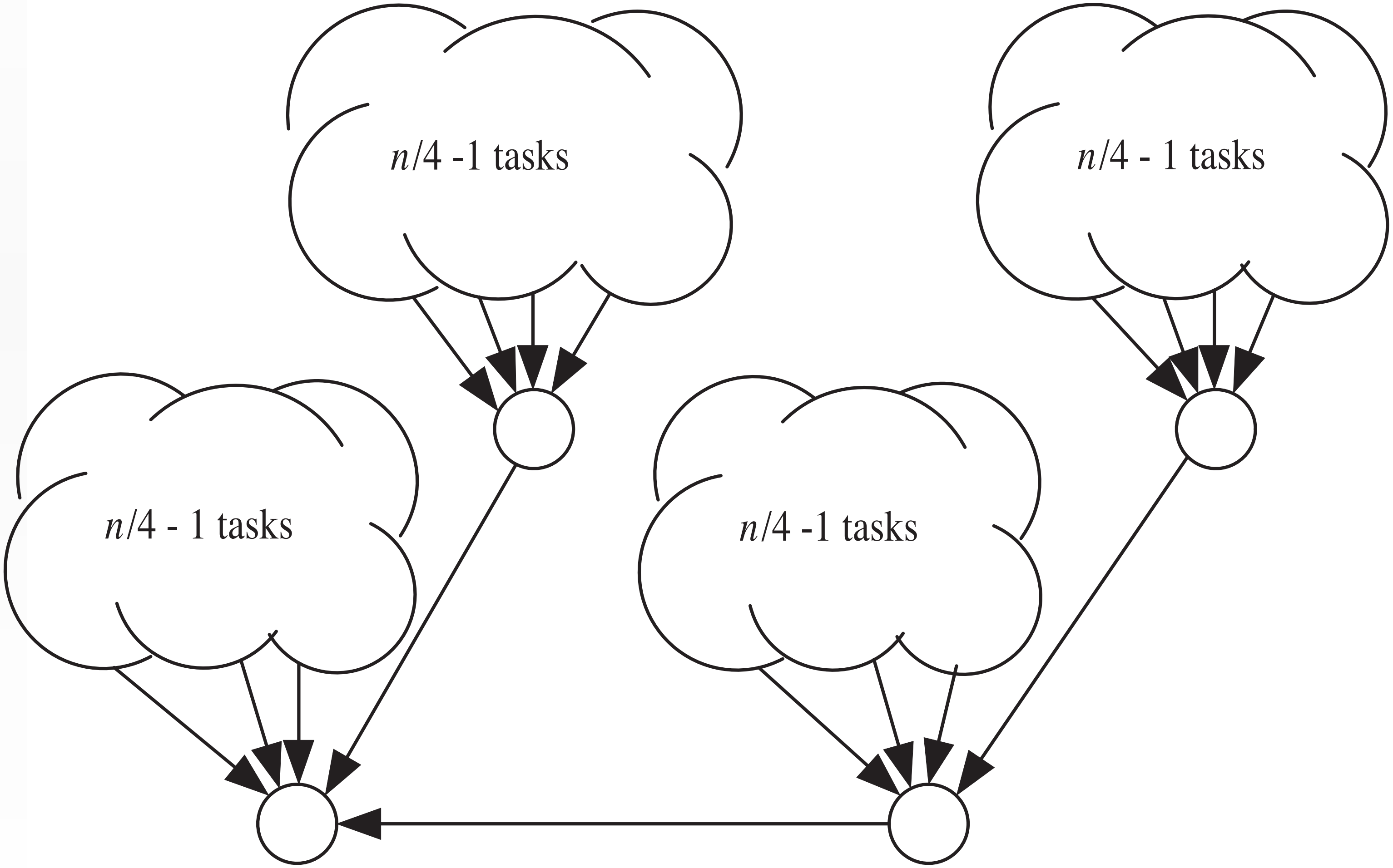
Parallel Reduction Evolution



Parallel Reduction Evolution

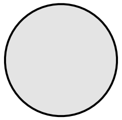


Parallel Reduction Evolution

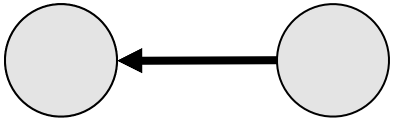


Binomial Trees

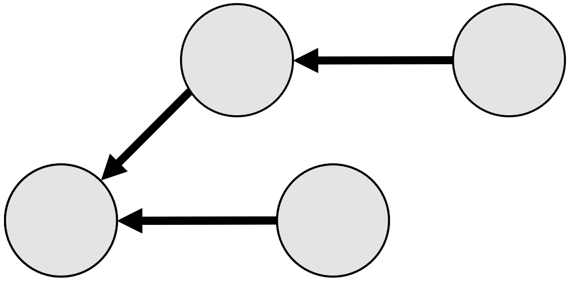
Binomial Trees



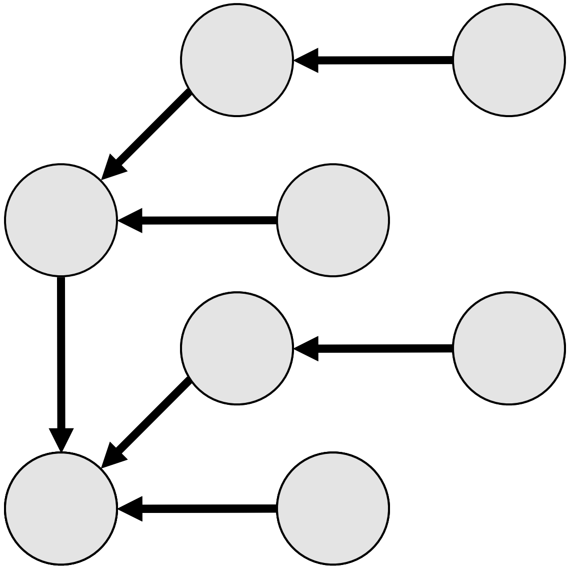
Binomial Trees



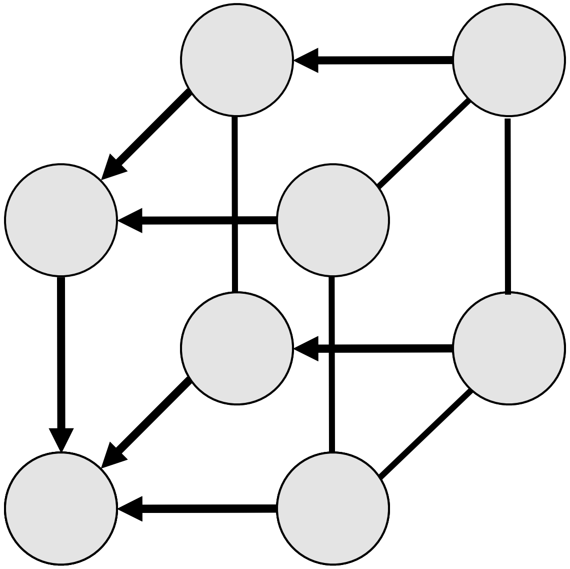
Binomial Trees



Binomial Trees

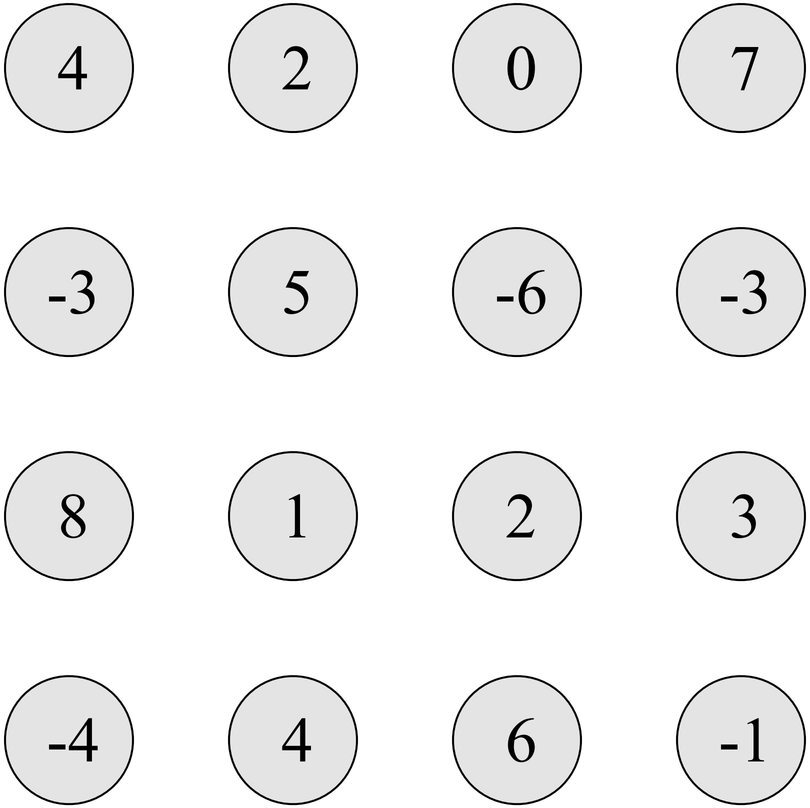


Binomial Trees



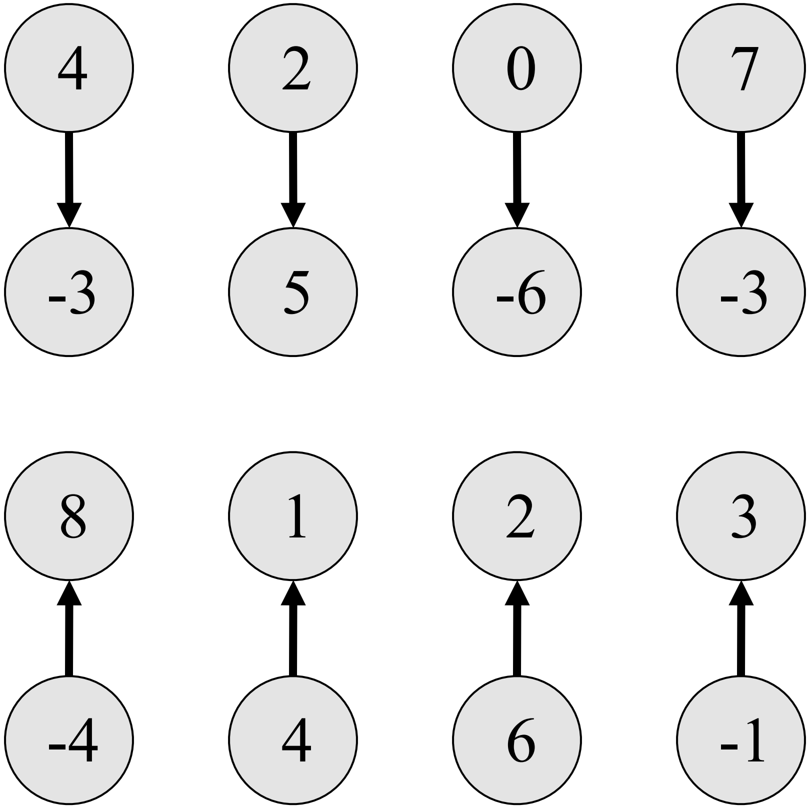
Subgraph of hypercube

Finding Global Sum

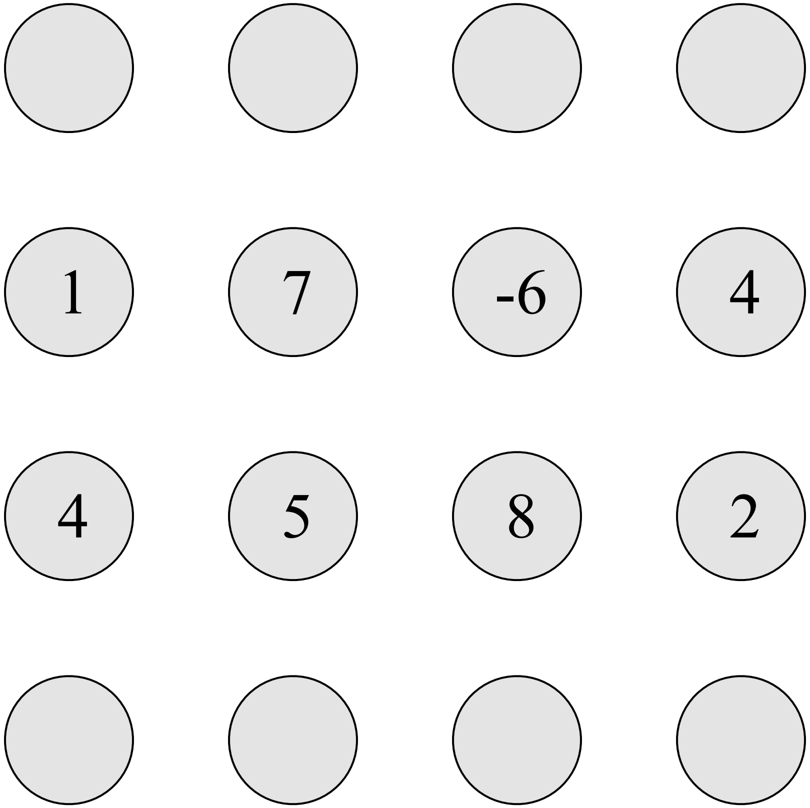


4	2	0	7
-3	5	-6	-3
8	1	2	3
-4	4	6	-1

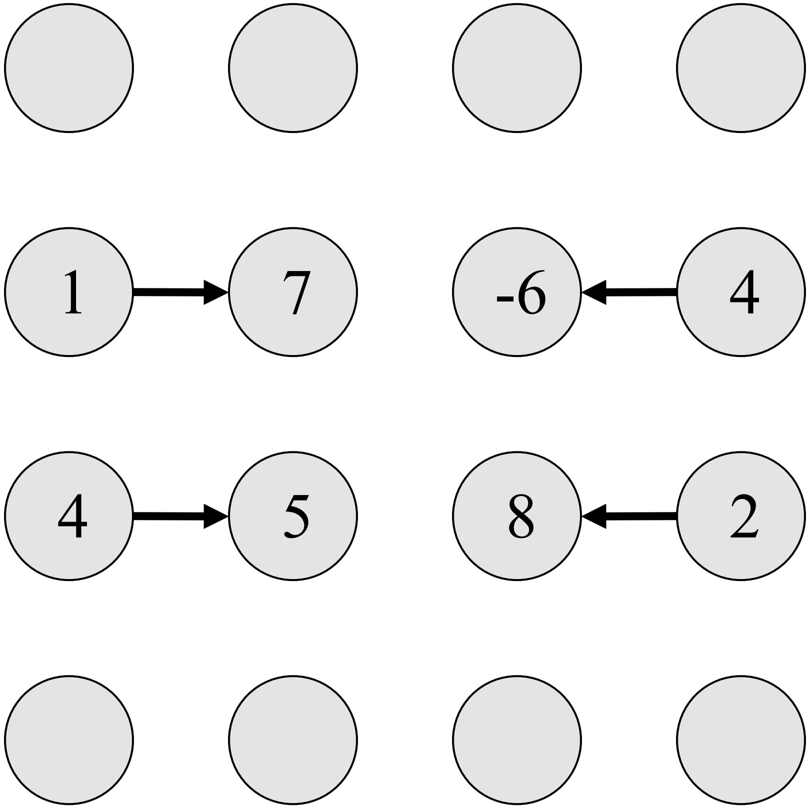
Finding Global Sum



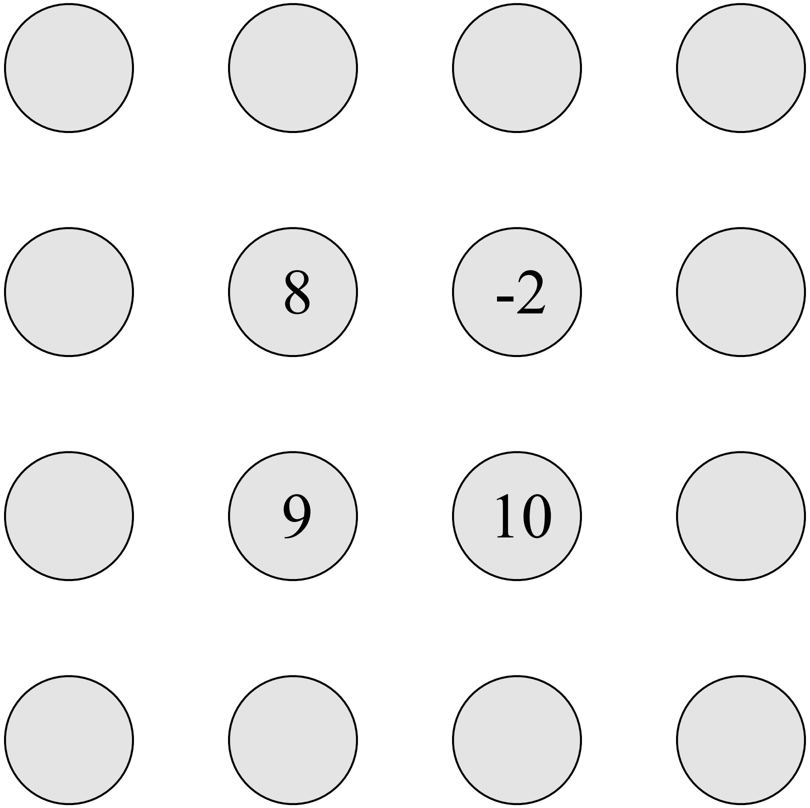
Finding Global Sum



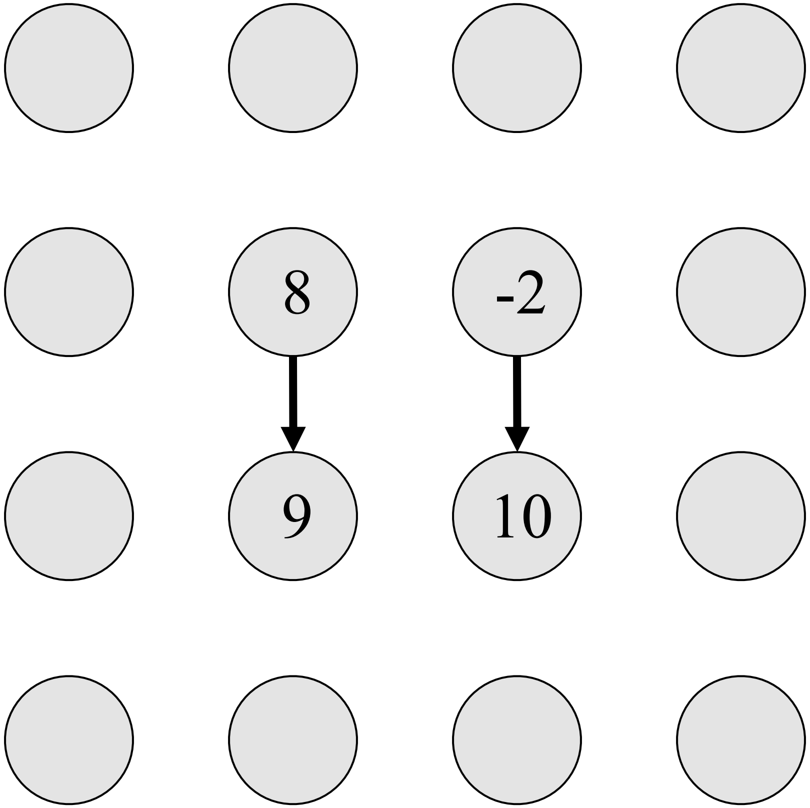
Finding Global Sum



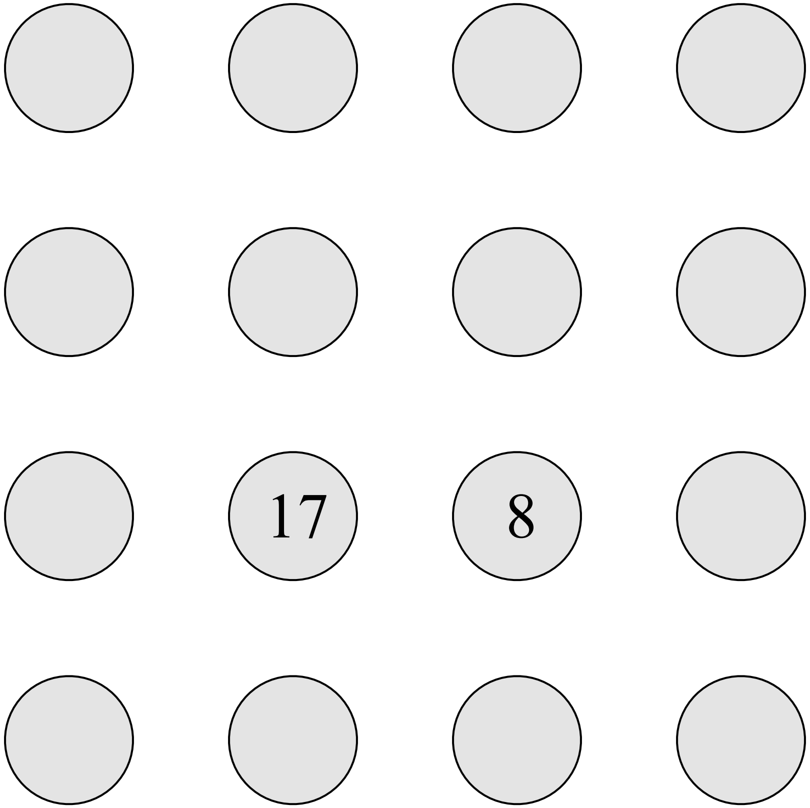
Finding Global Sum



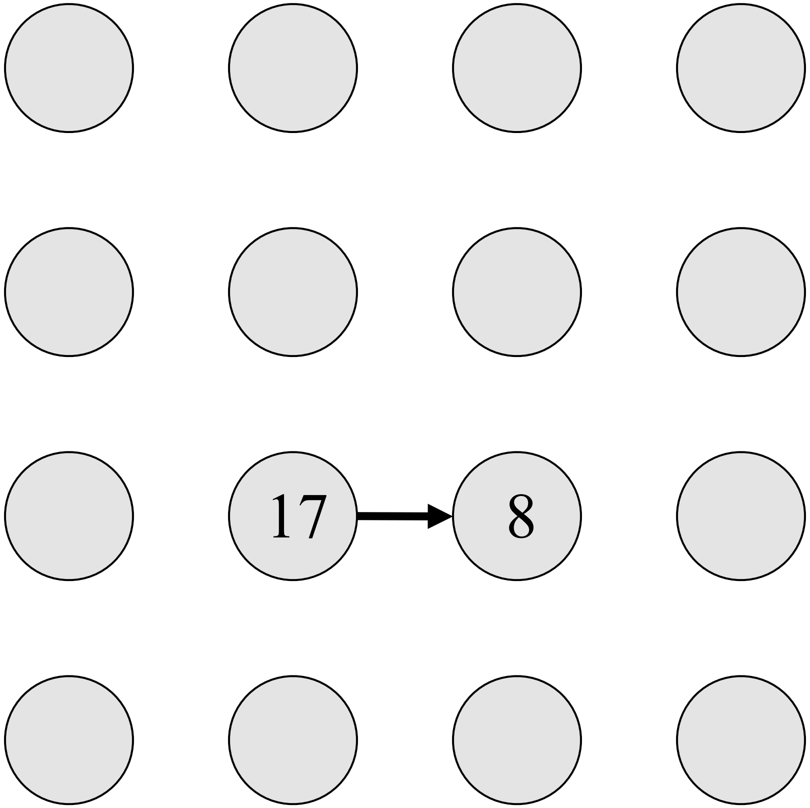
Finding Global Sum



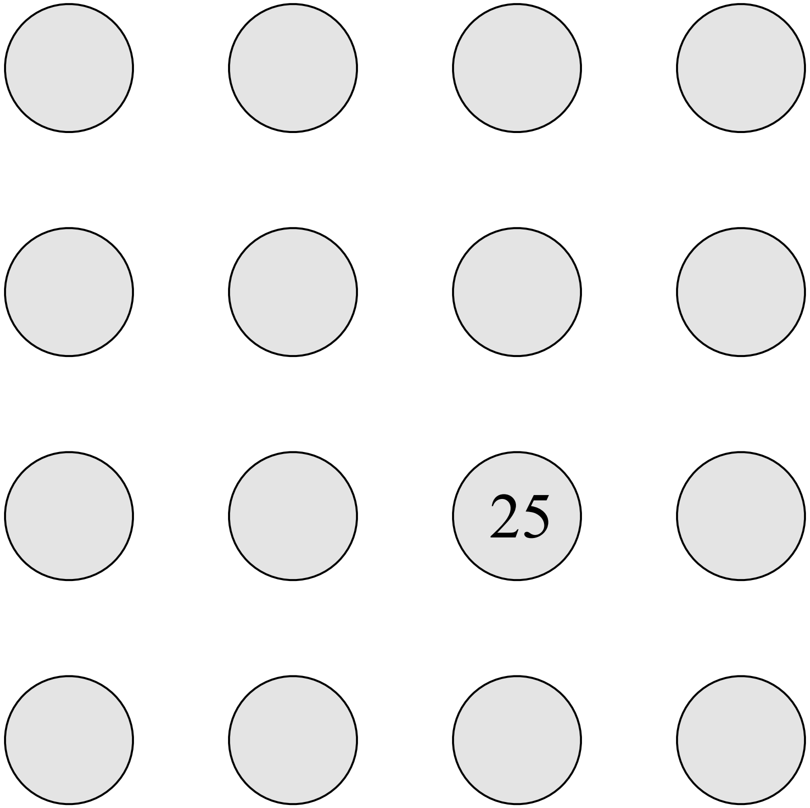
Finding Global Sum



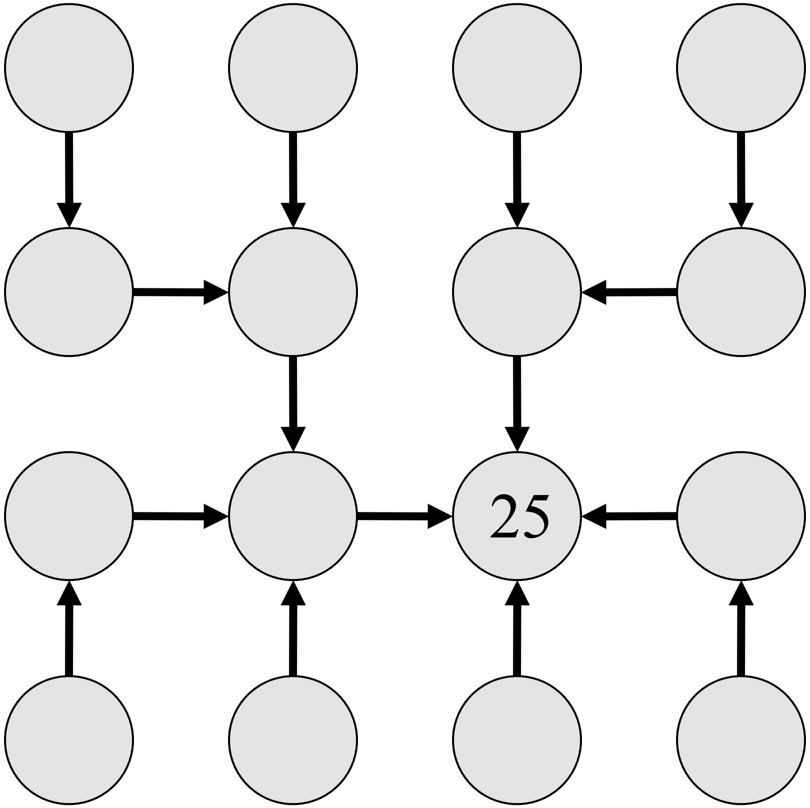
Finding Global Sum



Finding Global Sum

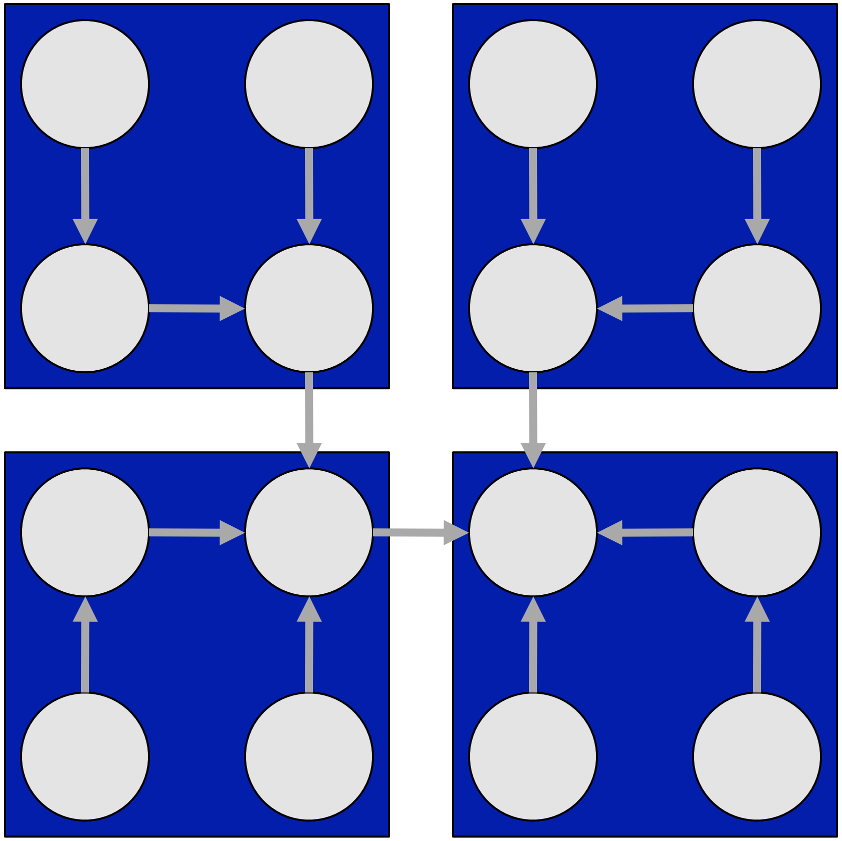


Finding Global Sum

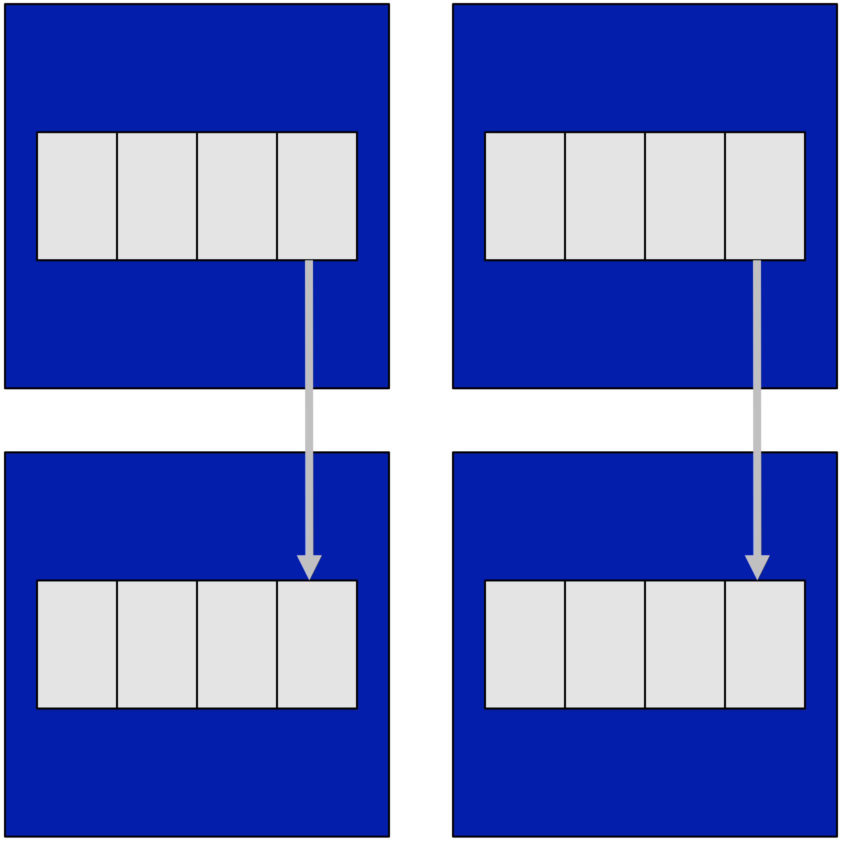


Binomial Tree

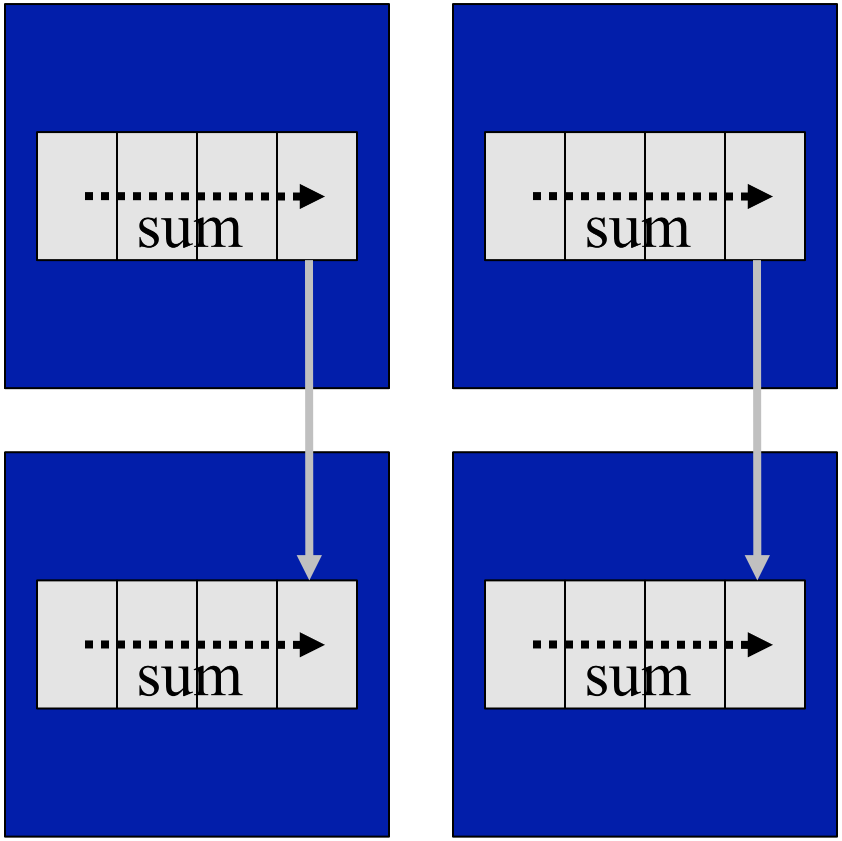
Agglomeration



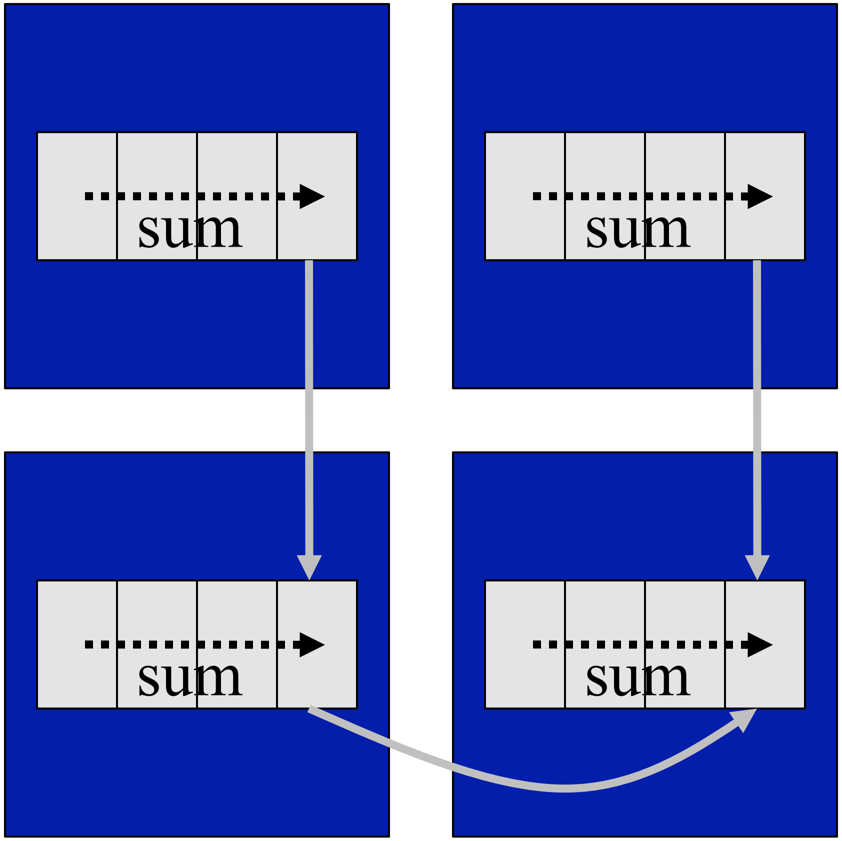
Agglomeration leads to actual communication



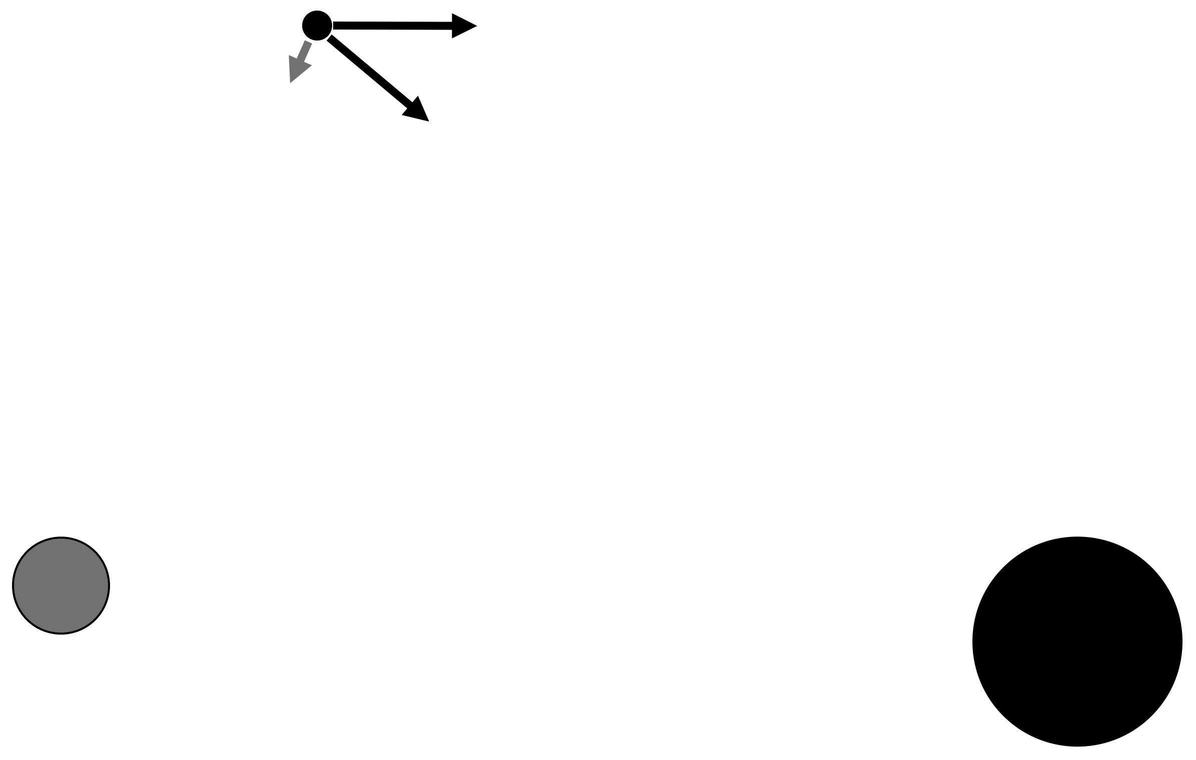
Agglomeration leads to actual communication



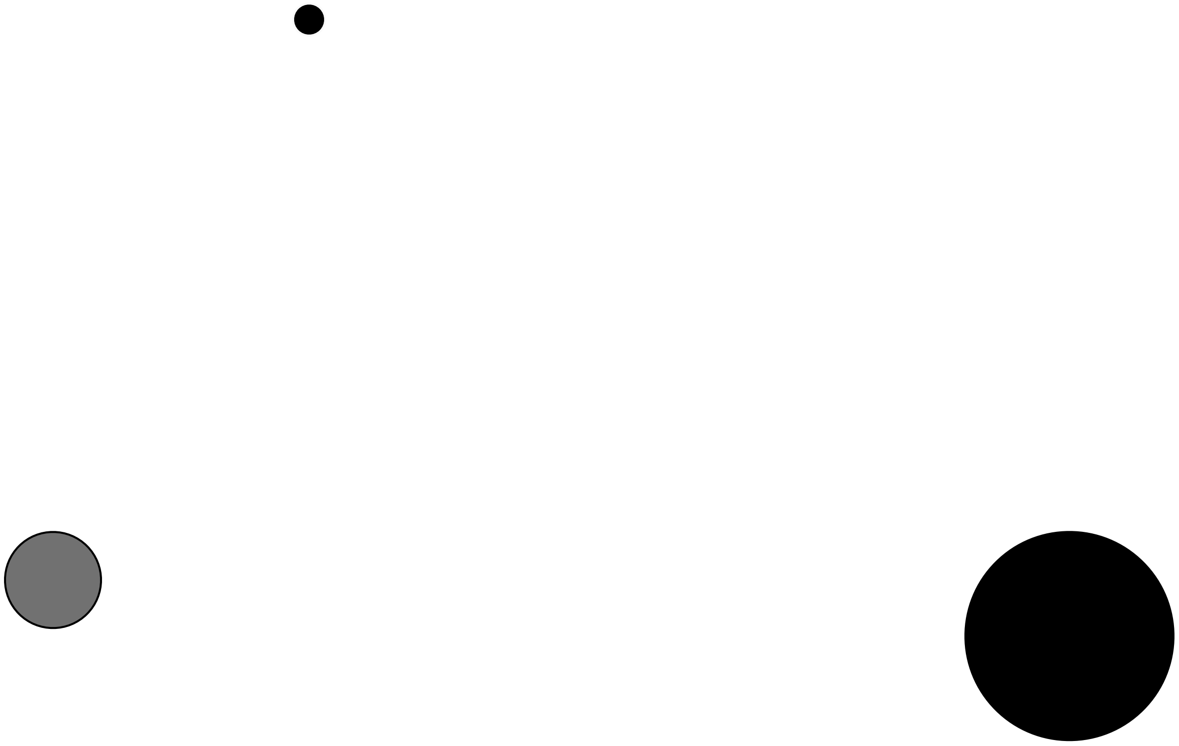
Agglomeration leads to actual communication



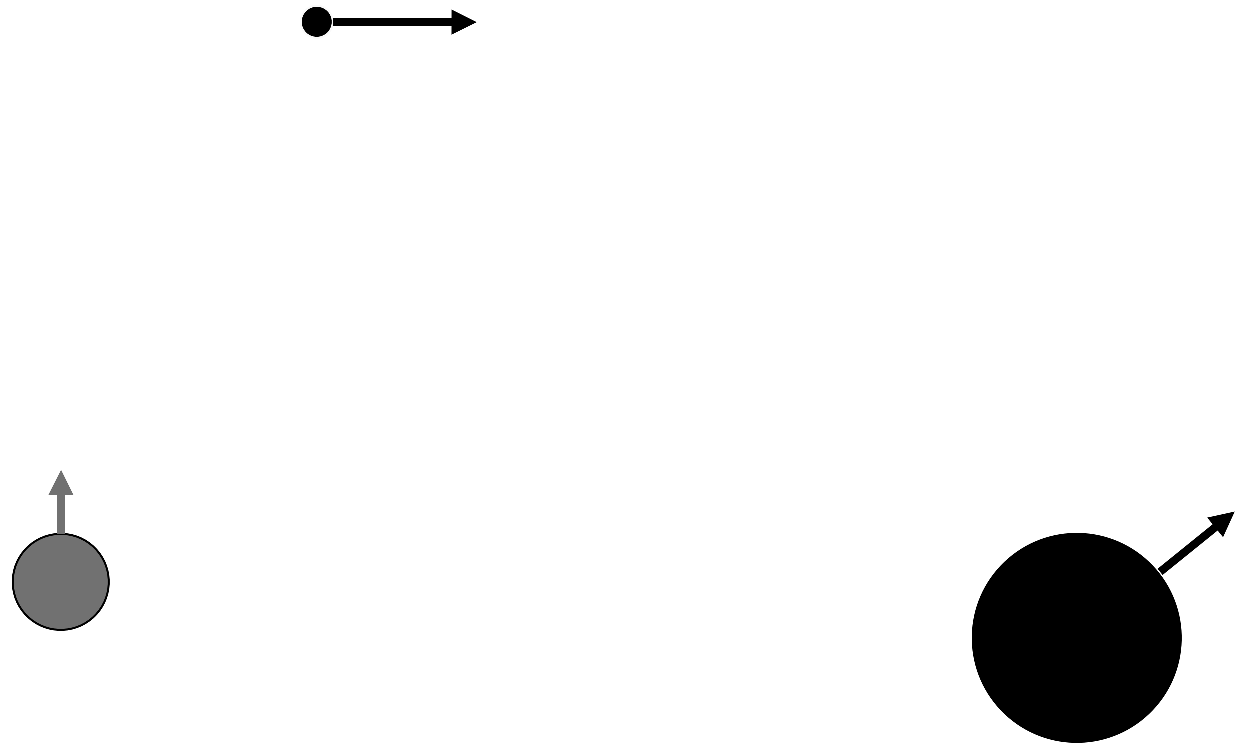
The n-body Problem



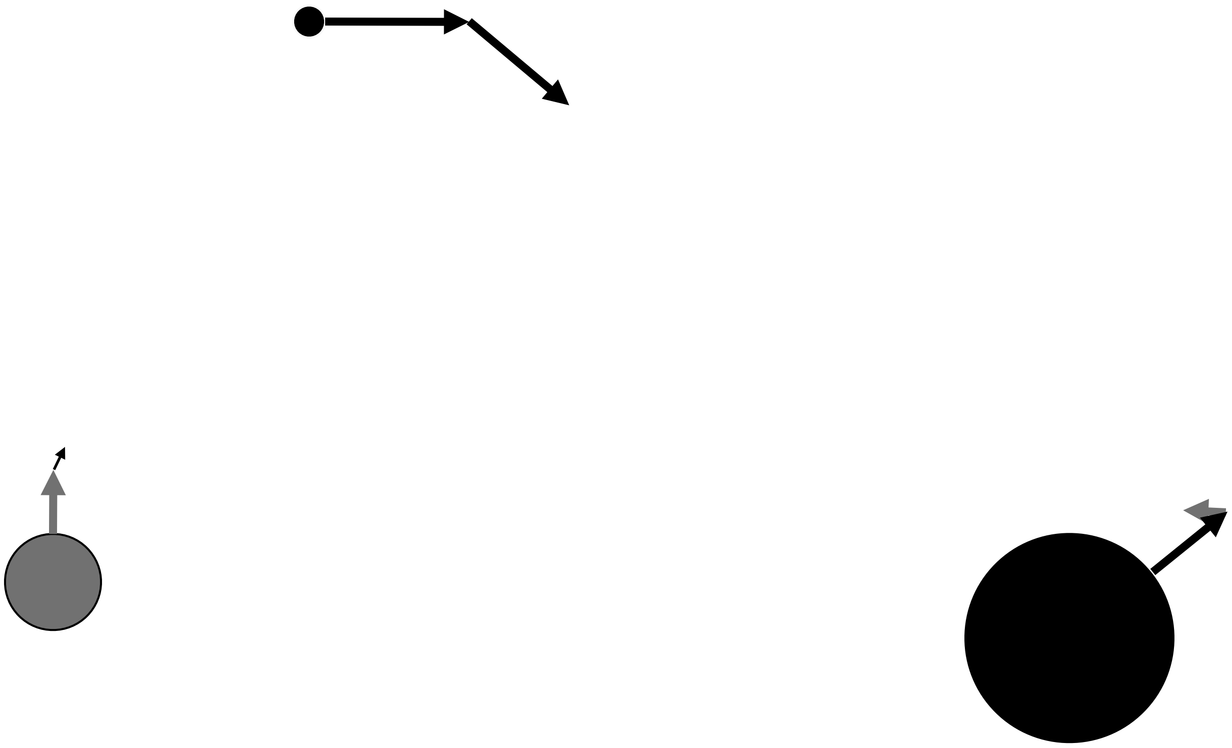
The n-body Problem



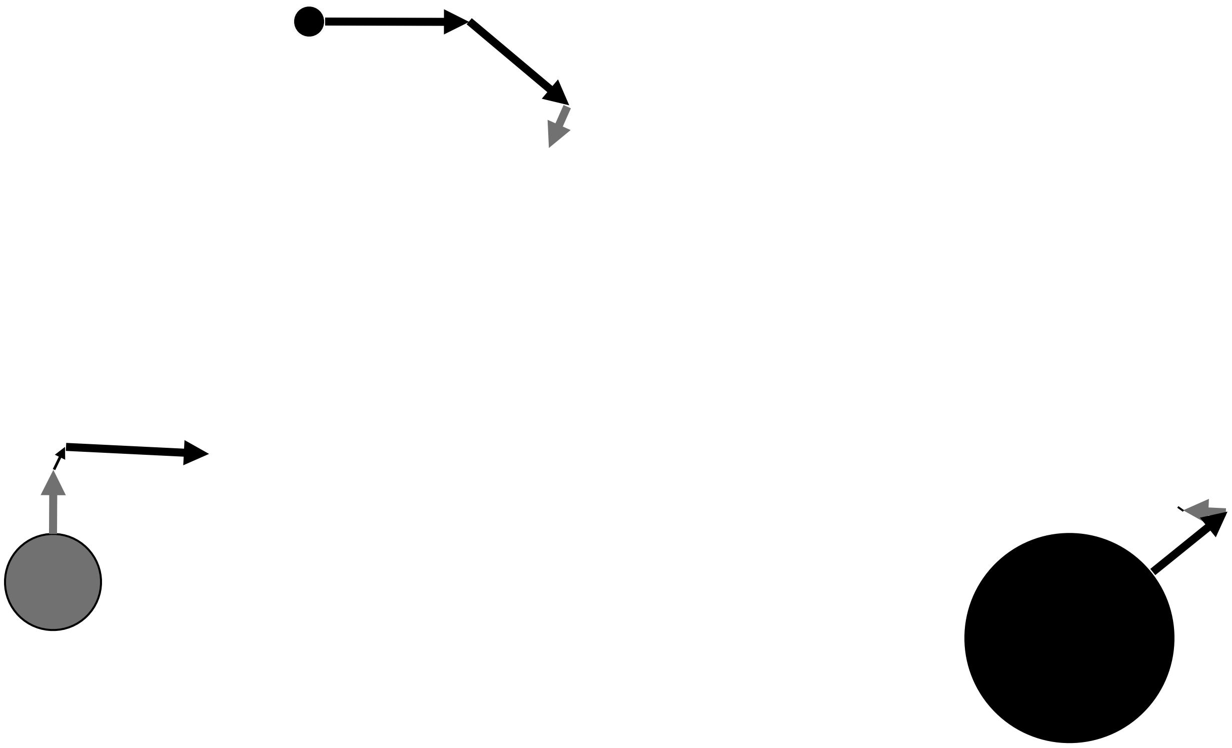
The n-body Problem



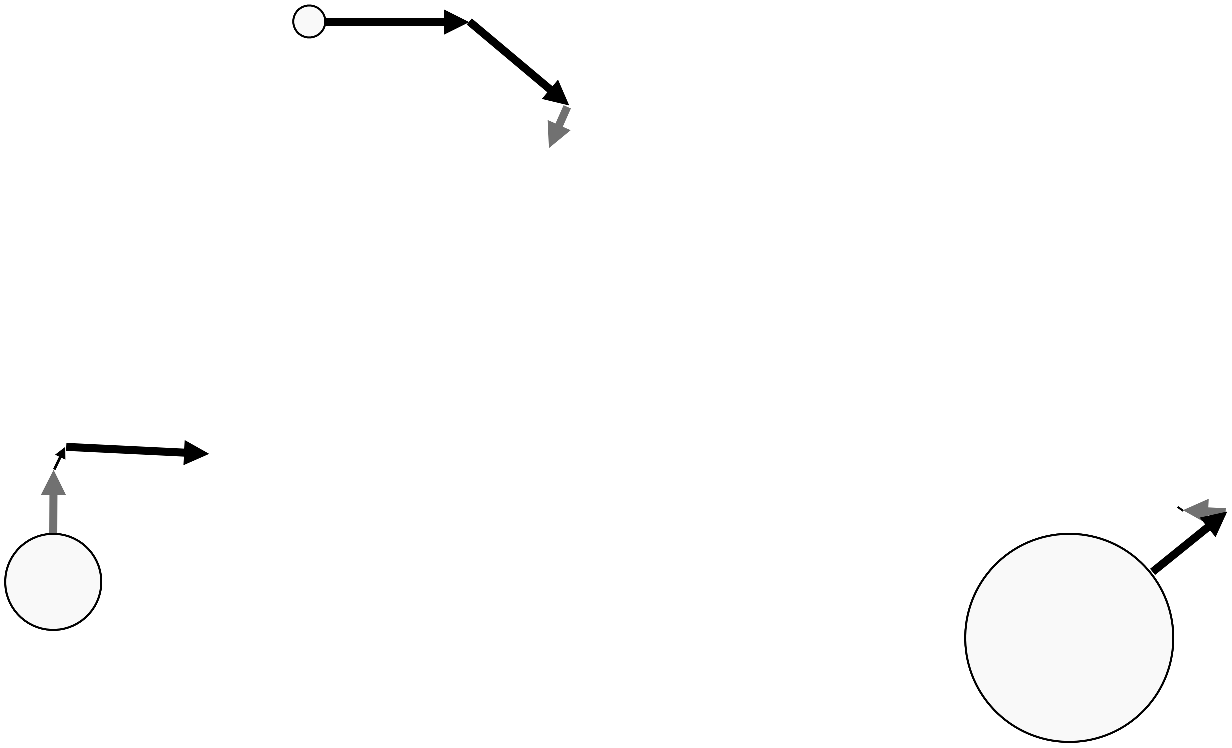
The n-body Problem



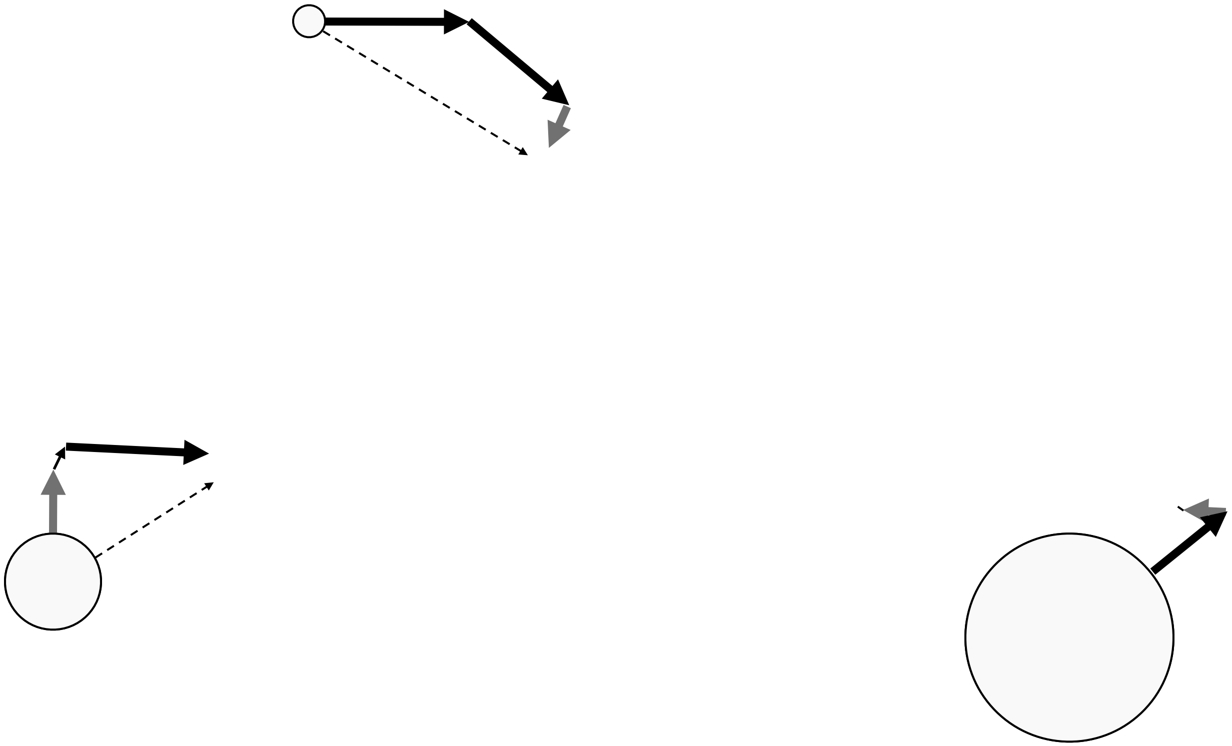
The n-body Problem



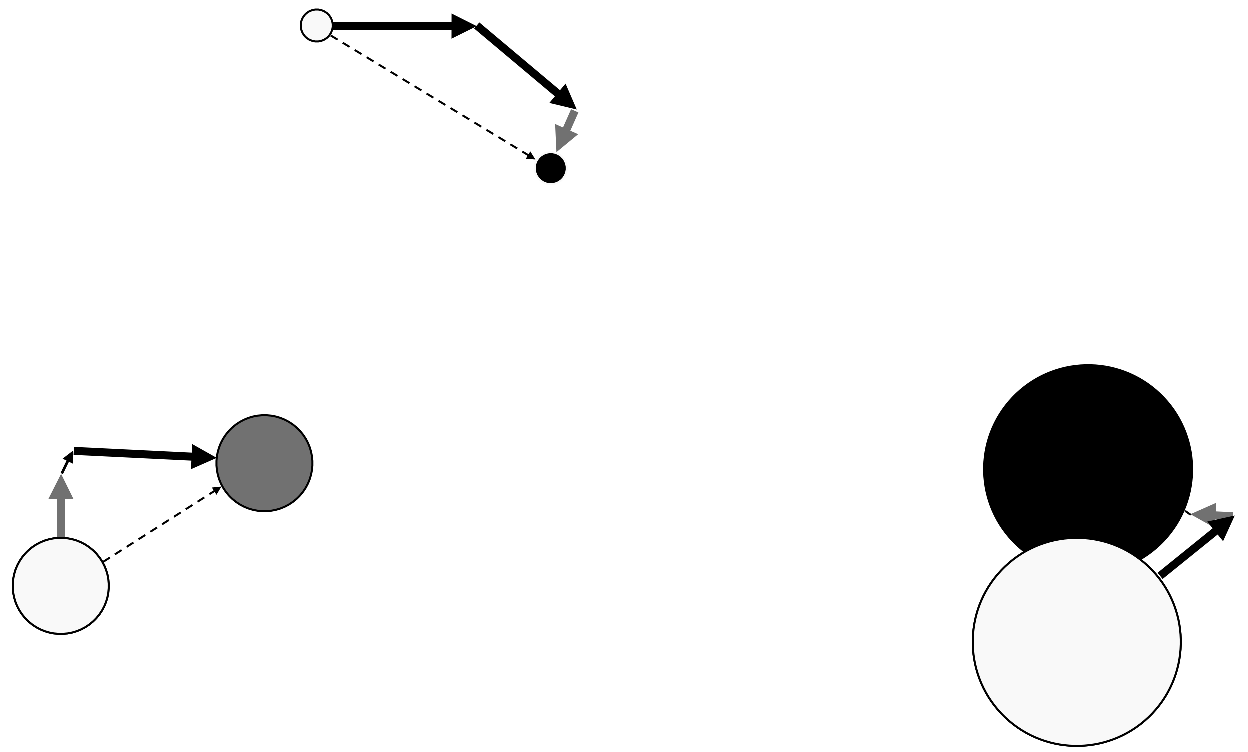
The n-body Problem



The n-body Problem



The n-body Problem



Partitioning

Partitioning

- Domain partitioning

Partitioning

- Domain partitioning
- Assume one task per particle

Partitioning

- Domain partitioning
- Assume one task per particle
- Task has particle's position, velocity vector

Partitioning

- Domain partitioning
- Assume one task per particle
- Task has particle's position, velocity vector
- Iteration

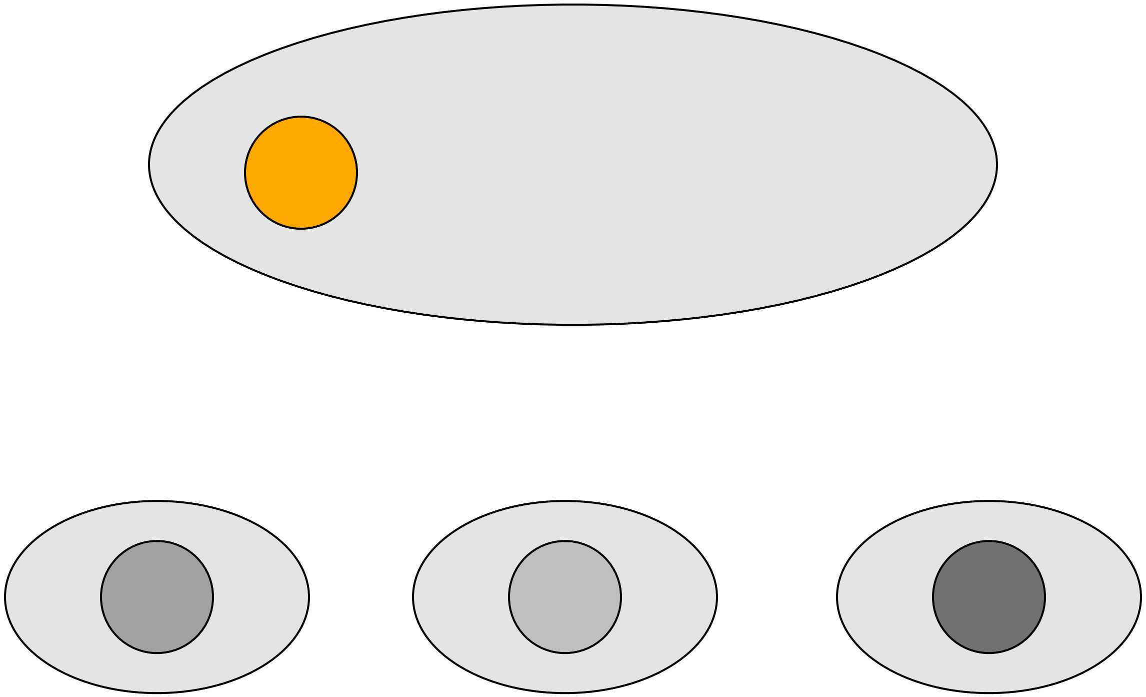
Partitioning

- Domain partitioning
- Assume one task per particle
- Task has particle's position, velocity vector
- Iteration
  - Get positions of all other particles

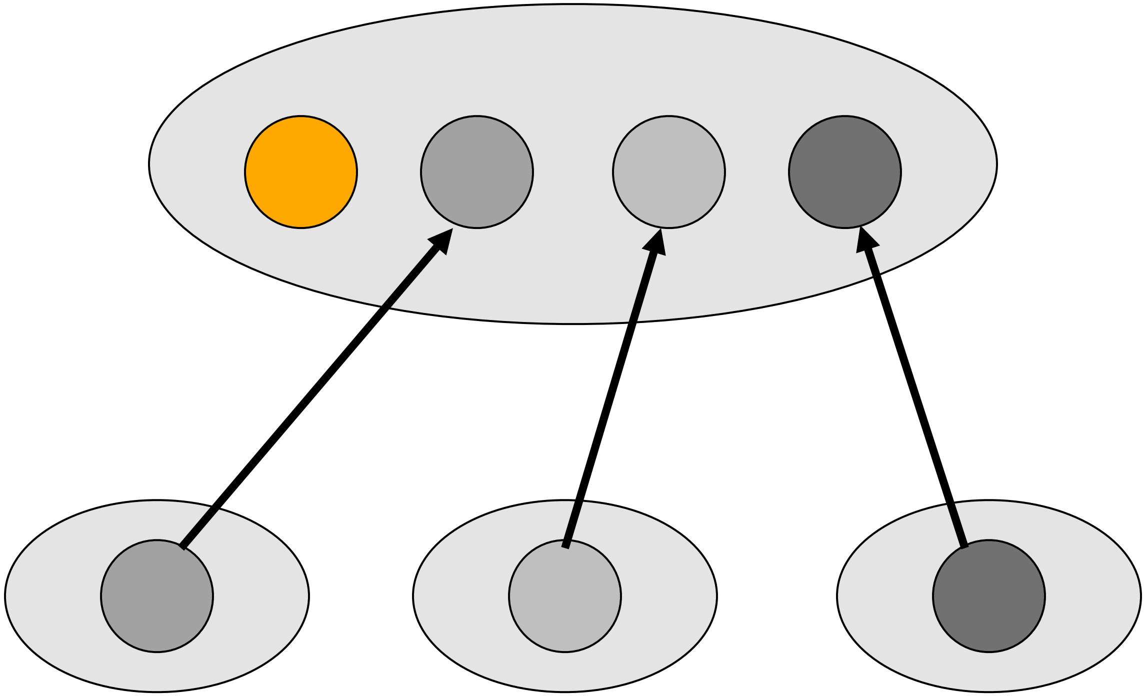
Partitioning

- Domain partitioning
- Assume one task per particle
- Task has particle's position, velocity vector
- Iteration
  - Get positions of all other particles
  - Compute new position, velocity

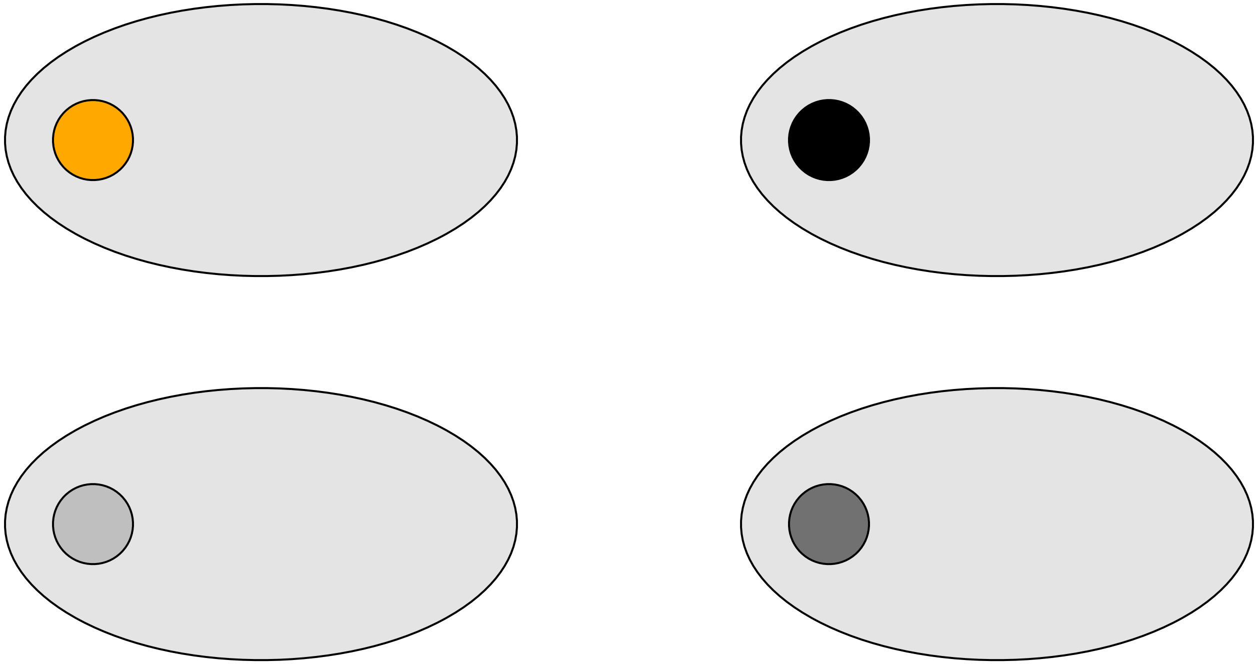
Gather



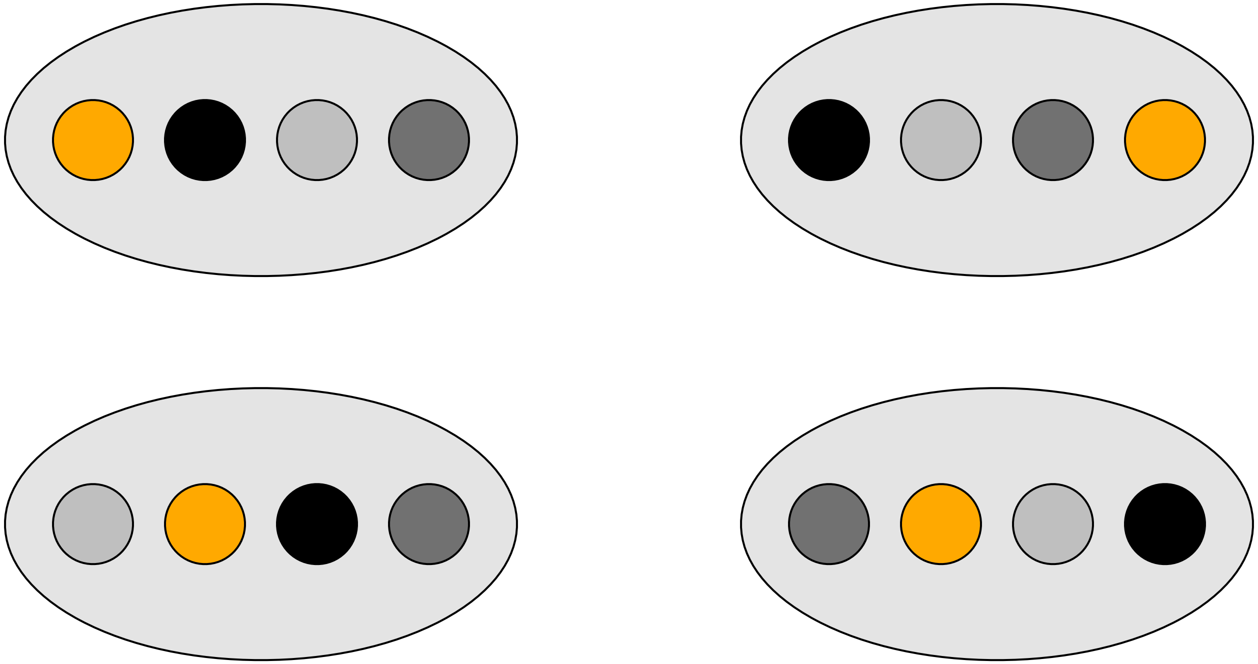
Gather



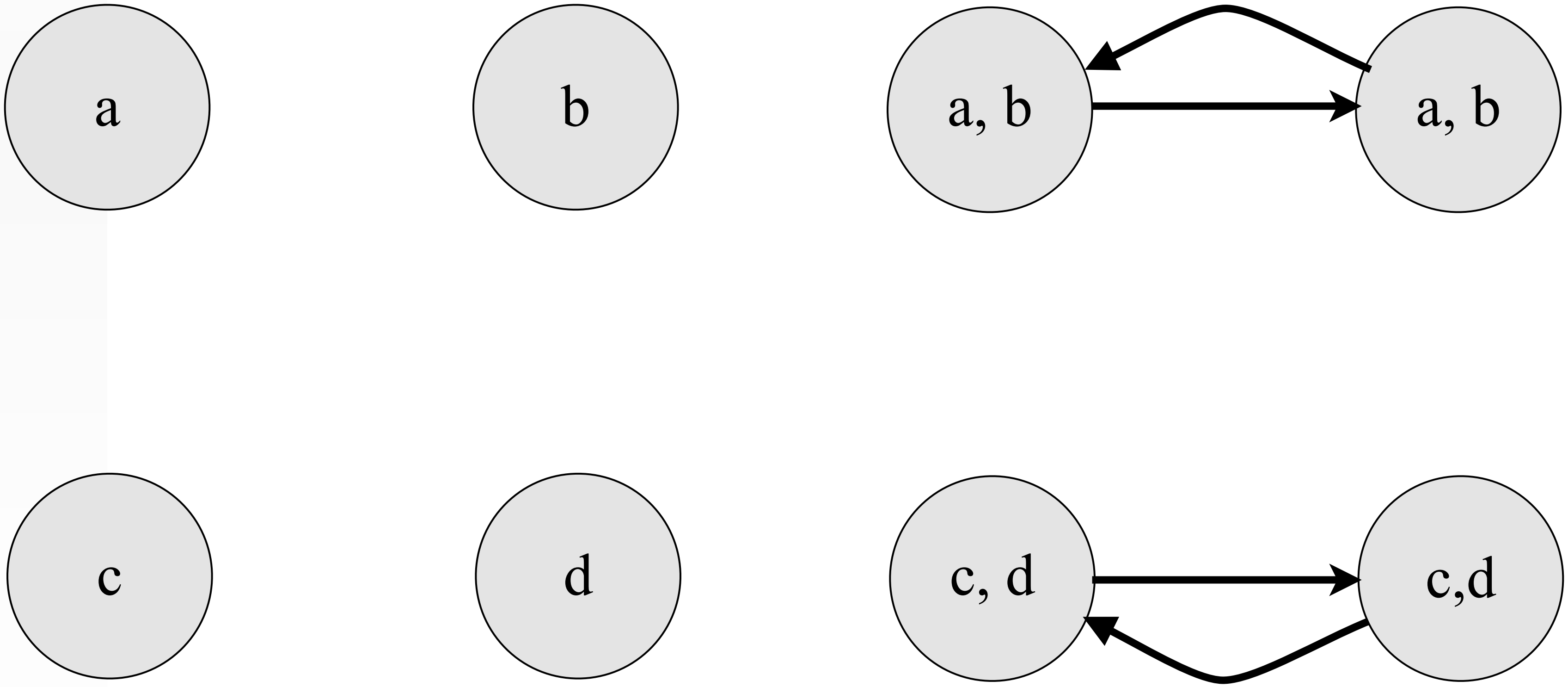
All-gather



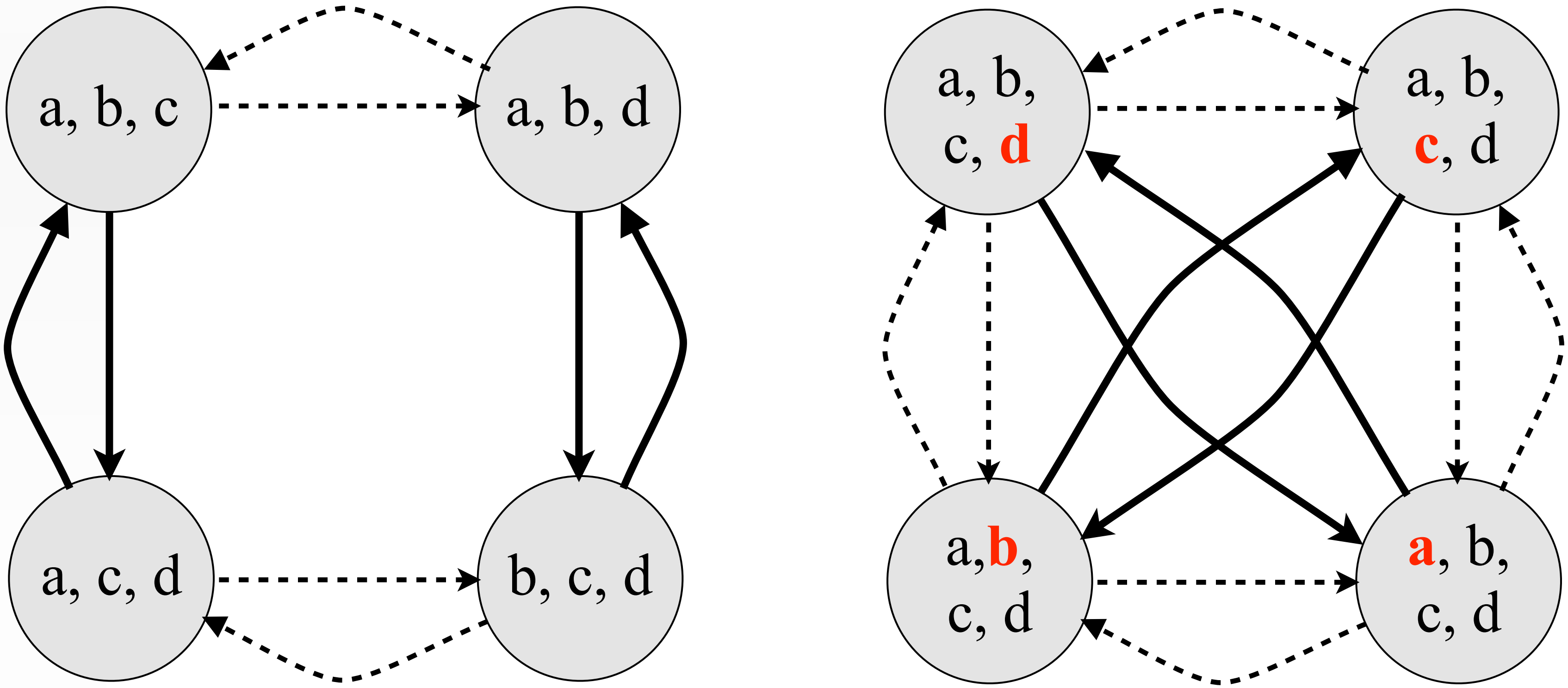
All-gather



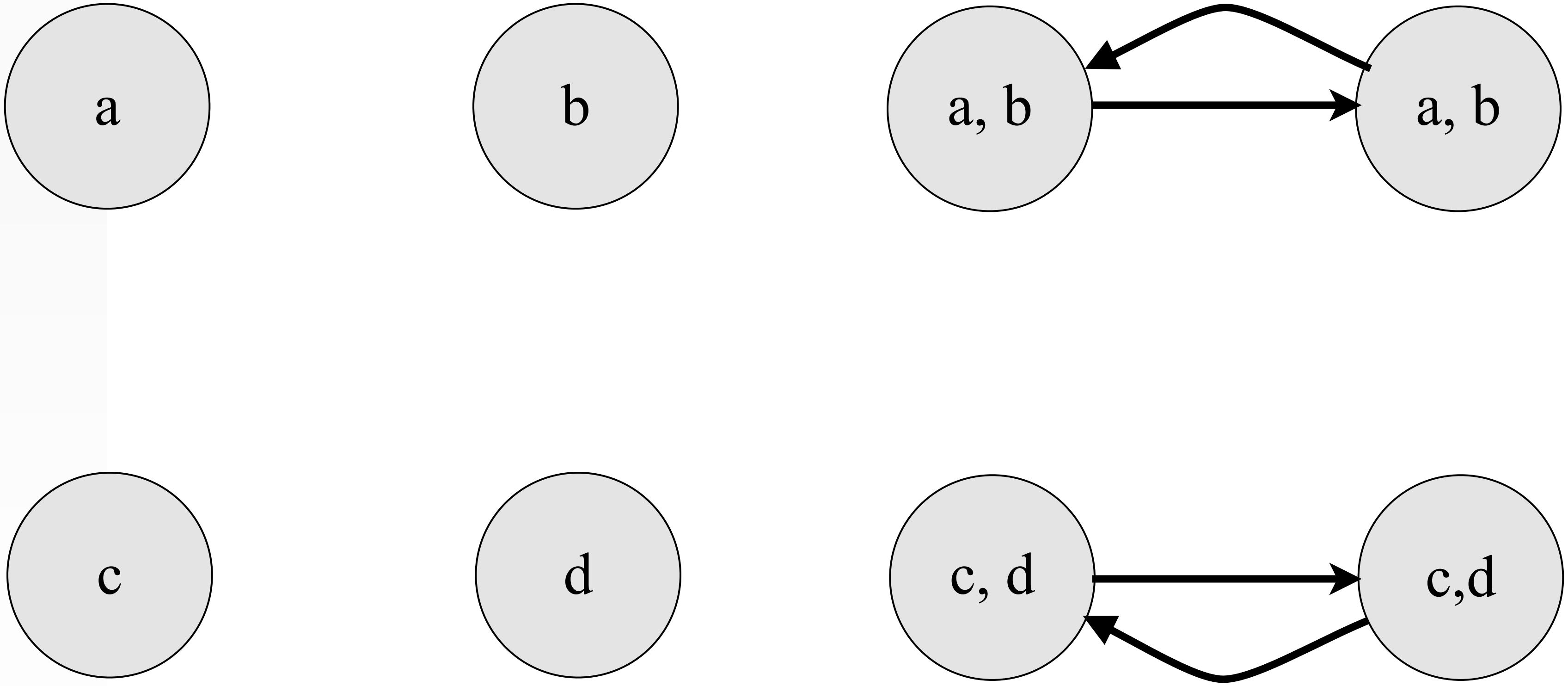
Complete Graph for All-gather -- operations shown, no ordering required



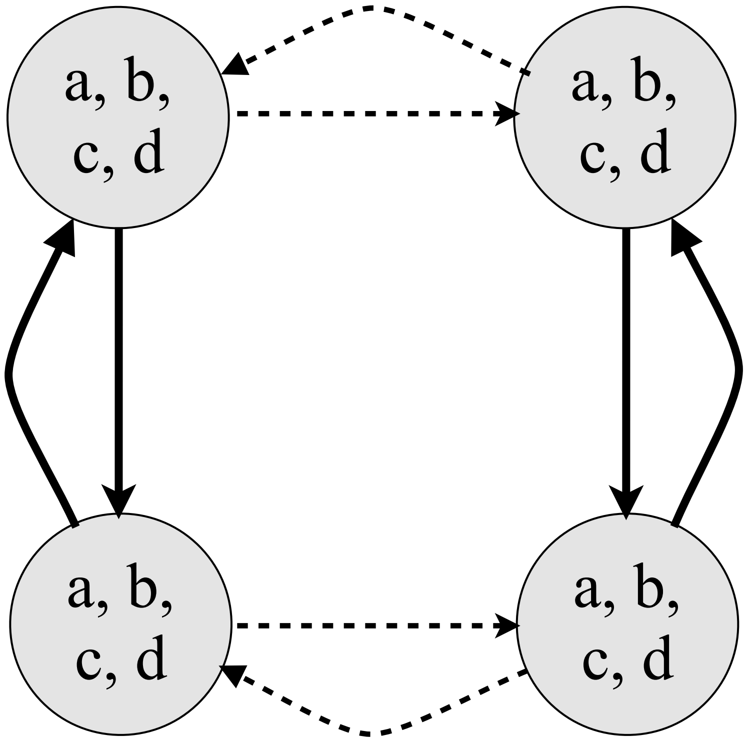
Complete Graph for All-gather -- operations shown, no ordering required



Hypercube-based All-gather -- ordering required



Complete Graph for All-gather



Communication Time

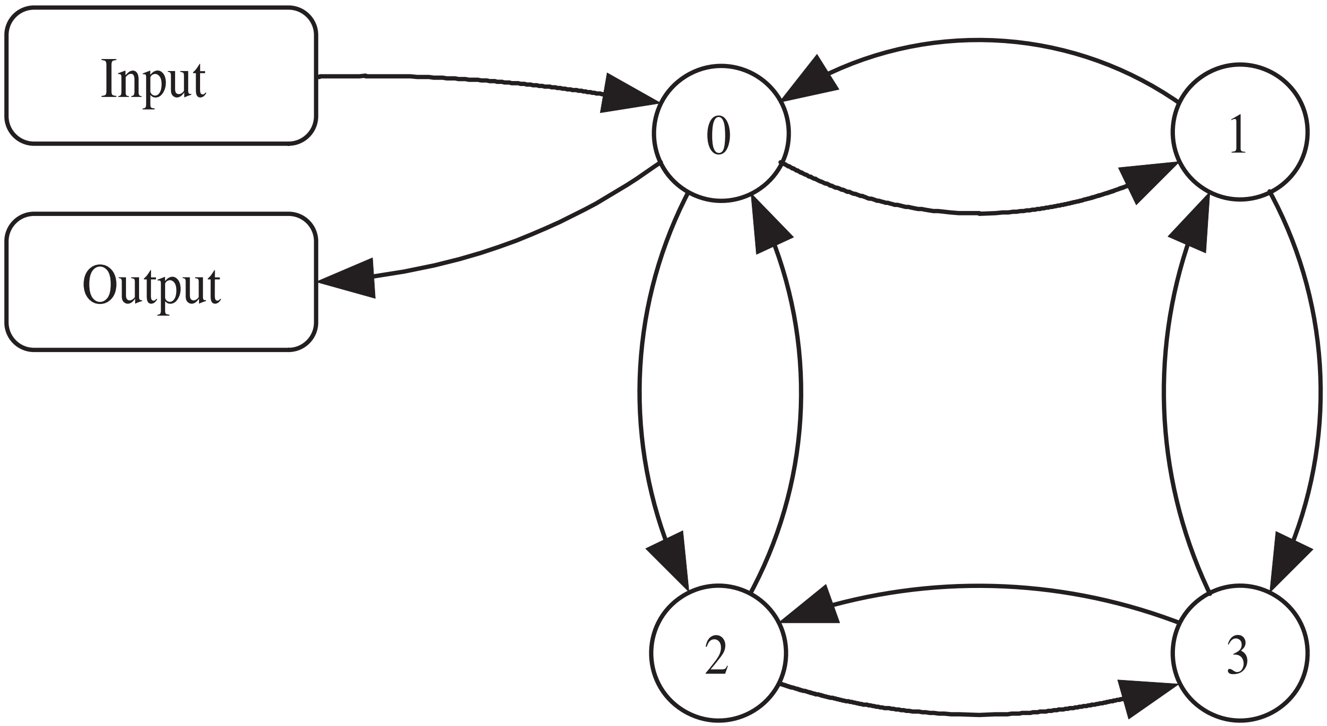
Complete graph

$$(p-1)(\lambda + \frac{n/p}{\beta}) = (p-1)\lambda + \frac{n(p-1)}{\beta p}$$

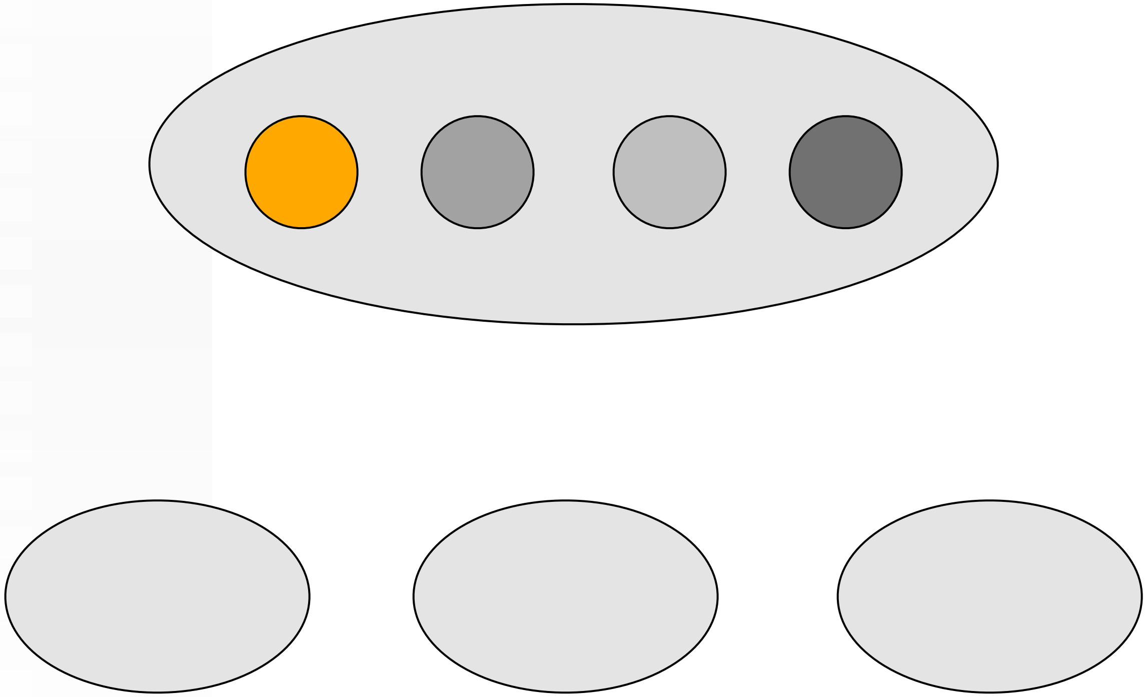
Hypercube

$$\sum_{i=1}^{\log p} \left(\lambda + \frac{2^{\text{i-1}} n}{\beta p} \right) = \lambda \log p + \frac{n(p-1)}{\beta p}$$

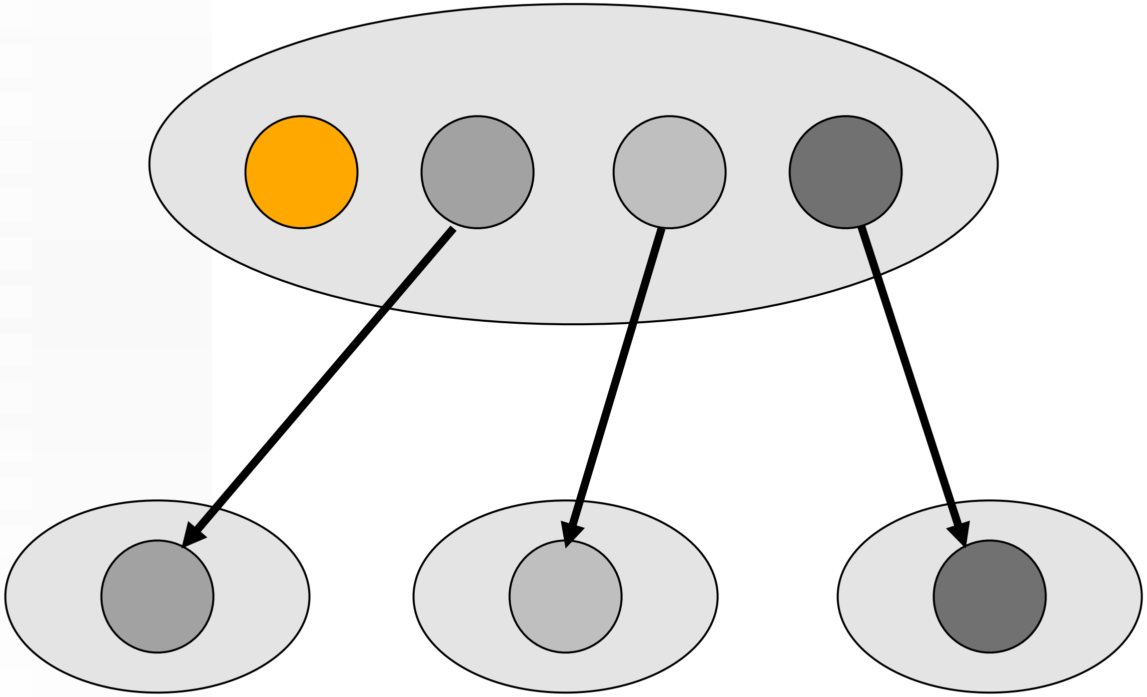
Adding Data Input



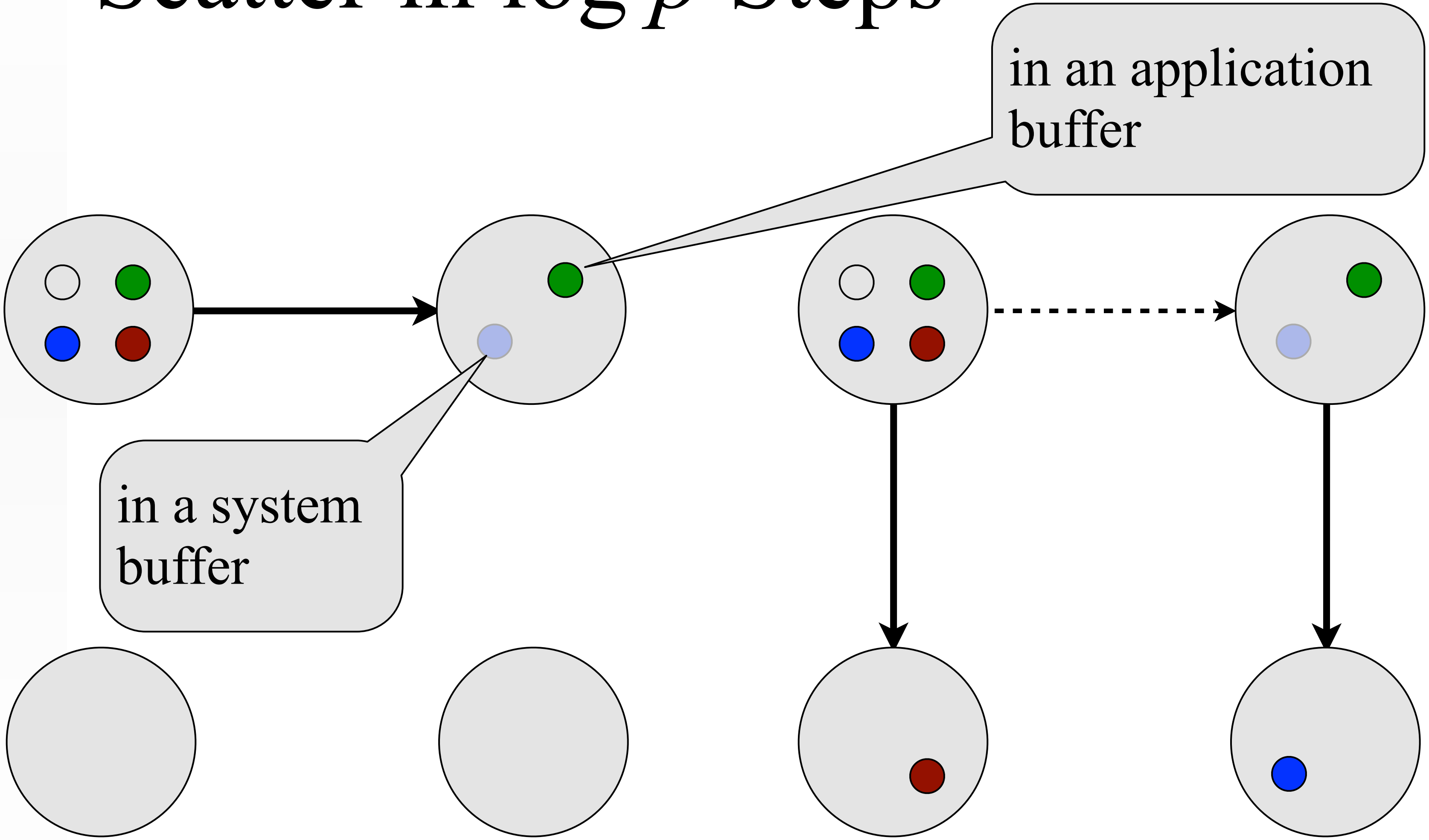
Scatter



Scatter



Scatter in log *p* Steps



in an application buffer

in a system buffer

Summary: Task/channel Model

Summary: Task/channel Model

- Parallel computation

Summary: Task/channel Model

- Parallel computation
  - Set of tasks

Summary: Task/channel Model

- Parallel computation
  - Set of tasks
  - Interactions through channels

Summary: Task/channel Model

- Parallel computation
    - Set of tasks
    - Interactions through channels
- Good designs

Summary: Task/channel Model

- Parallel computation
  - Set of tasks
  - Interactions through channels
- Good designs
  - Maximize local computations

Summary: Task/channel Model

- Parallel computation
  - Set of tasks
  - Interactions through channels
- Good designs
  - Maximize local computations
  - Minimize communications

Summary: Task/channel Model

- Parallel computation
  - Set of tasks
  - Interactions through channels
- Good designs
  - Maximize local computations
  - Minimize communications
  - Scale up

Summary: Design Steps

Summary: Design Steps

- Partition computation

Summary: Design Steps

- Partition computation
- Agglomerate tasks

Summary: Design Steps

- Partition computation
- Agglomerate tasks
- Map tasks to processors

Summary: Design Steps

- Partition computation
- Agglomerate tasks
- Map tasks to processors
- Goals

Summary: Design Steps

- Partition computation
- Agglomerate tasks
- Map tasks to processors
- Goals
  - Maximize processor utilization

Summary: Design Steps

- Partition computation
- Agglomerate tasks
- Map tasks to processors
- Goals
  - Maximize processor utilization
  - Minimize inter-processor communication

Summary: Fundamental Algorithms

Summary: Fundamental Algorithms

- Reduction

Summary: Fundamental Algorithms

- Reduction
- Gather and scatter

Summary: Fundamental Algorithms

- Reduction
- Gather and scatter
- All-gather