Scalable Asynchronous Contact Mechanics using Charm++

Xiang Ni*, Laxmikant V. Kale* and Rasmus Tamstorf^

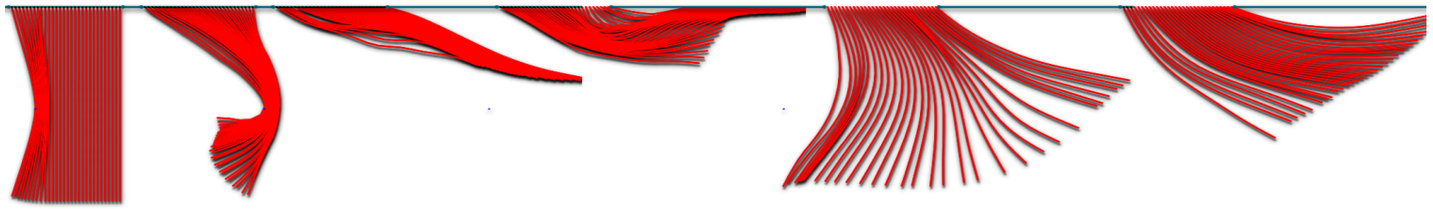
* University of Illinois at Urbana Champaign

^Walt Disney Animation Studios

Asynchronous Contact Mechanics

Asynchronous Contact Mechanics

Asynchronous Contact Mechanics



- Necessary Guarantees

Asynchronous Contact Mechanics

- Necessary Guarantees
  - **Safety:** no missed collisions

Asynchronous Contact Mechanics

- Necessary Guarantees
  - **Safety:** no missed collisions
  - **Correctness:** follow the laws of physics

Asynchronous Contact Mechanics

- Necessary Guarantees
  - **Safety:** no missed collisions
  - **Correctness:** follow the laws of physics
  - **Progress:** finish in a finite amount of time

Asynchronous Contact Mechanics

- Necessary Guarantees
  - **Safety:** no missed collisions
  - **Correctness:** follow the laws of physics
  - **Progress:** finish in a finite amount of time
- Problems with other existing algorithms

Asynchronous Contact Mechanics

- Necessary Guarantees
  - **Safety:** no missed collisions
  - **Correctness:** follow the laws of physics
  - **Progress:** finish in a finite amount of time
- Problems with other existing algorithms
  - An object can end up going through itself or another object

Asynchronous Contact Mechanics

- Necessary Guarantees
  - **Safety:** no missed collisions
  - **Correctness:** follow the laws of physics
  - **Progress:** finish in a finite amount of time
- Problems with other existing algorithms
  - An object can end up going through itself or another object
  - Violate physical properties

What you want

What you get

 pictures from Yi Wang at VT

What you want

What you get

3 pictures from Yi Wang at VT

What you want

What you get

3 pictures from Yi Wang at VT

What you want

What you get

incorrect handling of collisions

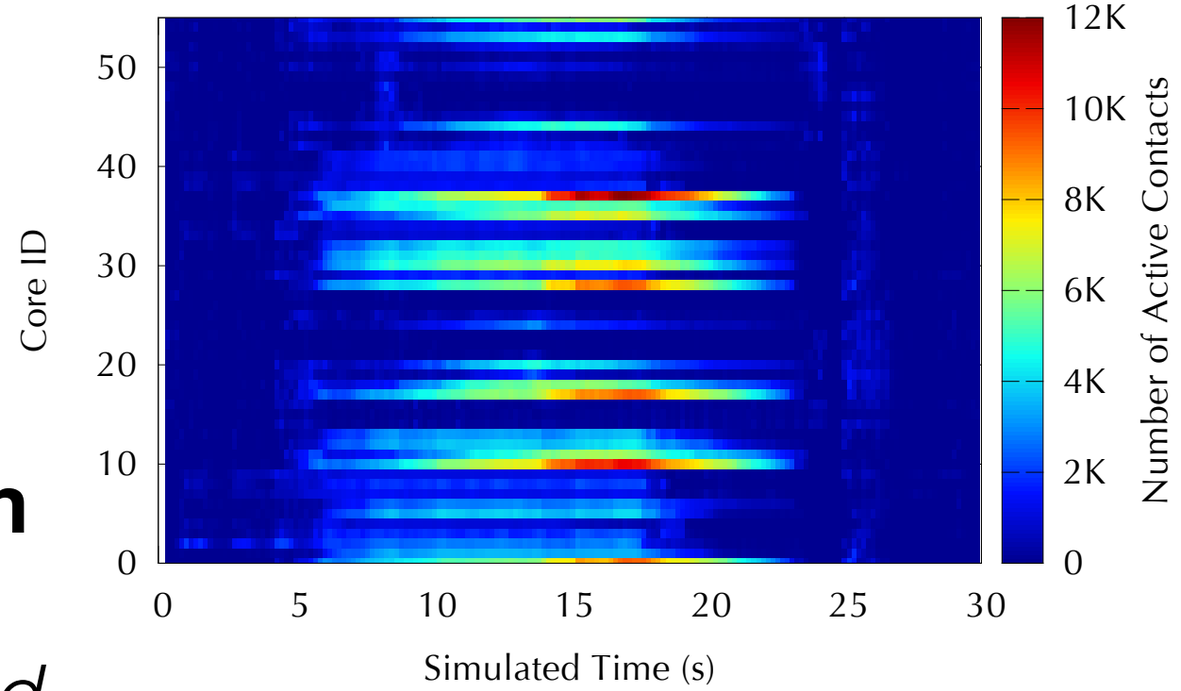
3 pictures from Yi Wang at VT

Parallelization Challenges

- **Highly irregular communication pattern**
  - *Message driven execution in Charm++*
- **Dynamic load imbalancing**
  - *Adaptive runtime system*
- **Very fine grained computation**
  - *Overlapping computation and communication*

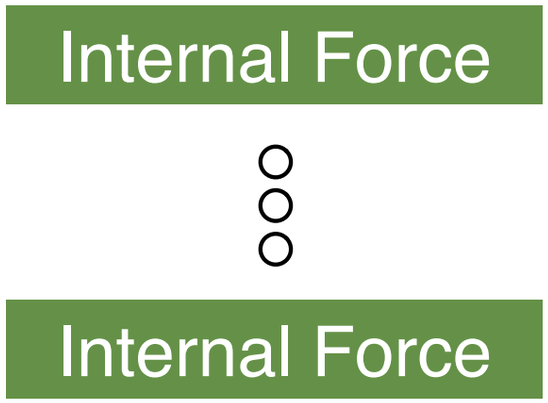
Parallelization Challenges

- **Highly irregular communication pattern**
  - *Message driven execution in Charm++*
- **Dynamic load imbalancing**
  - *Adaptive runtime system*
- **Very fine grained computation**
  - *Overlapping computation and communication*

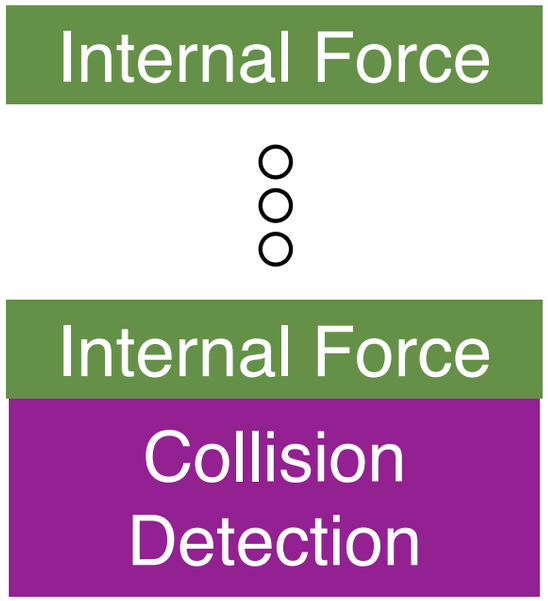


Overall Flow

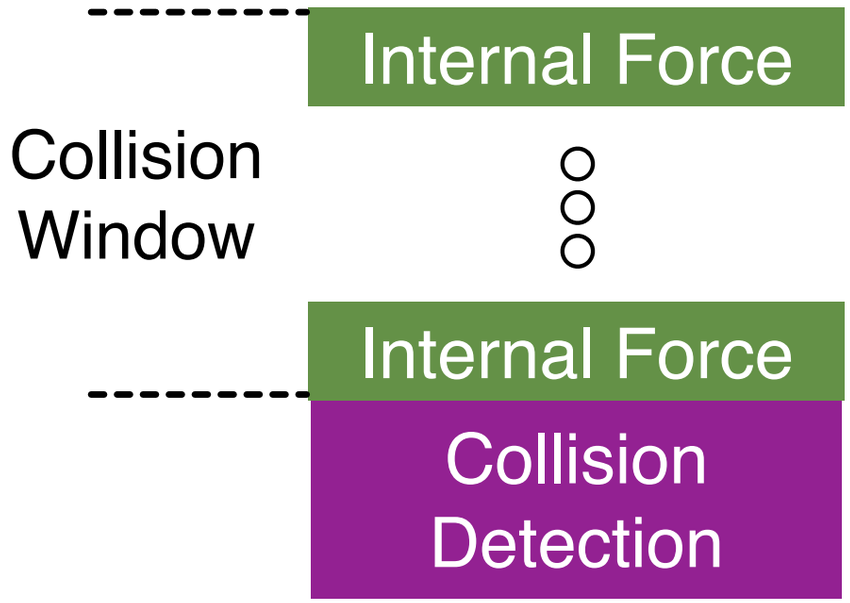
Overall Flow



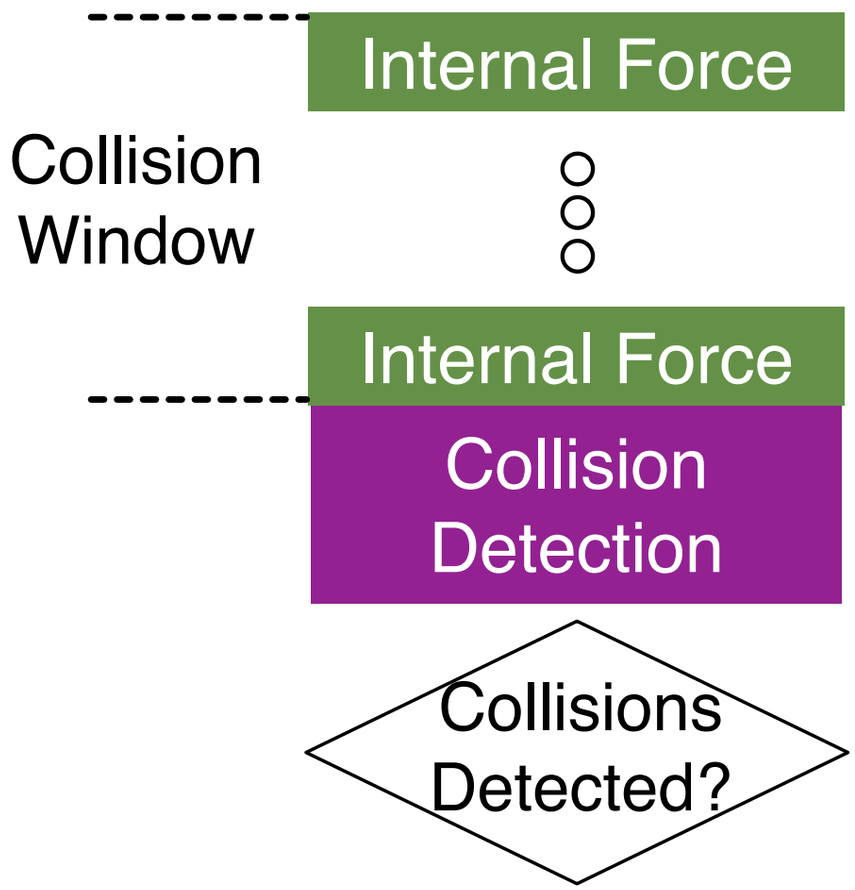
Overall Flow



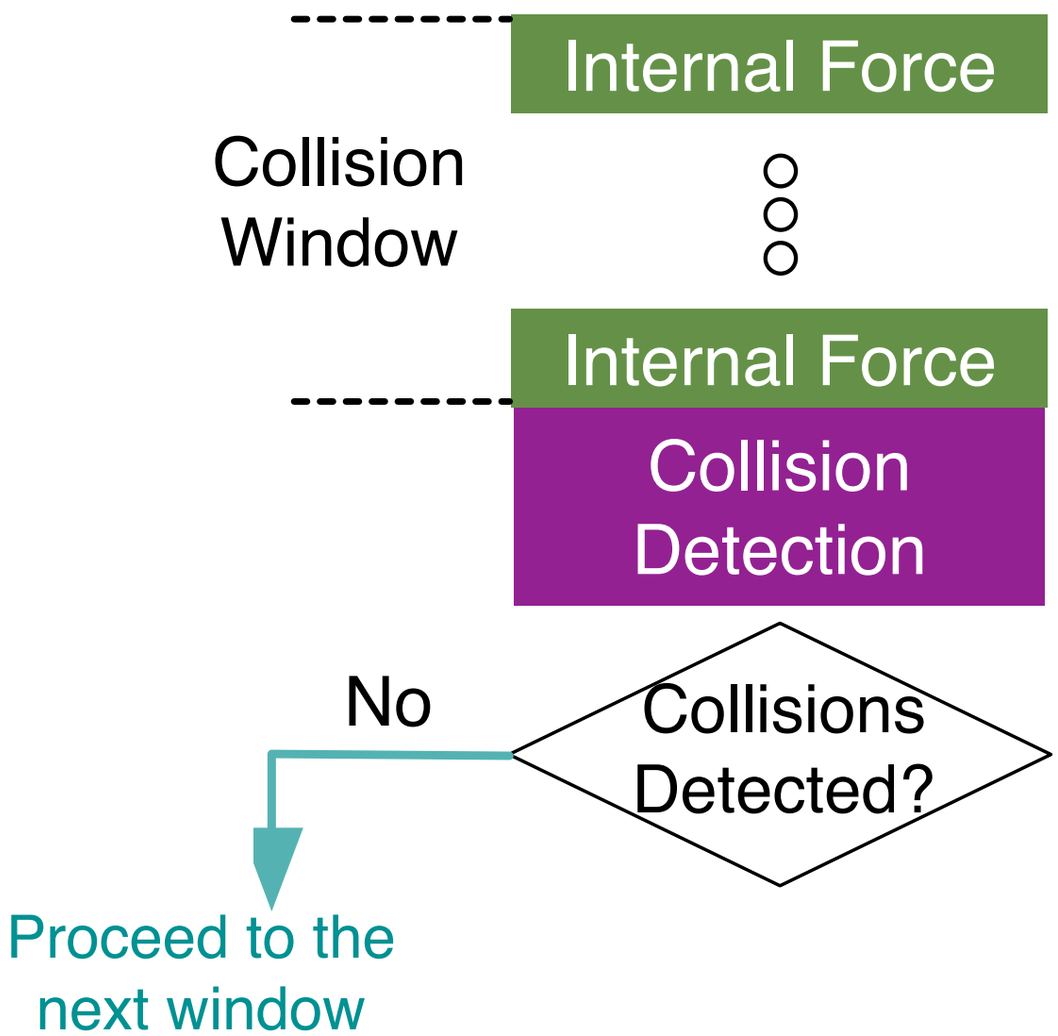
Overall Flow



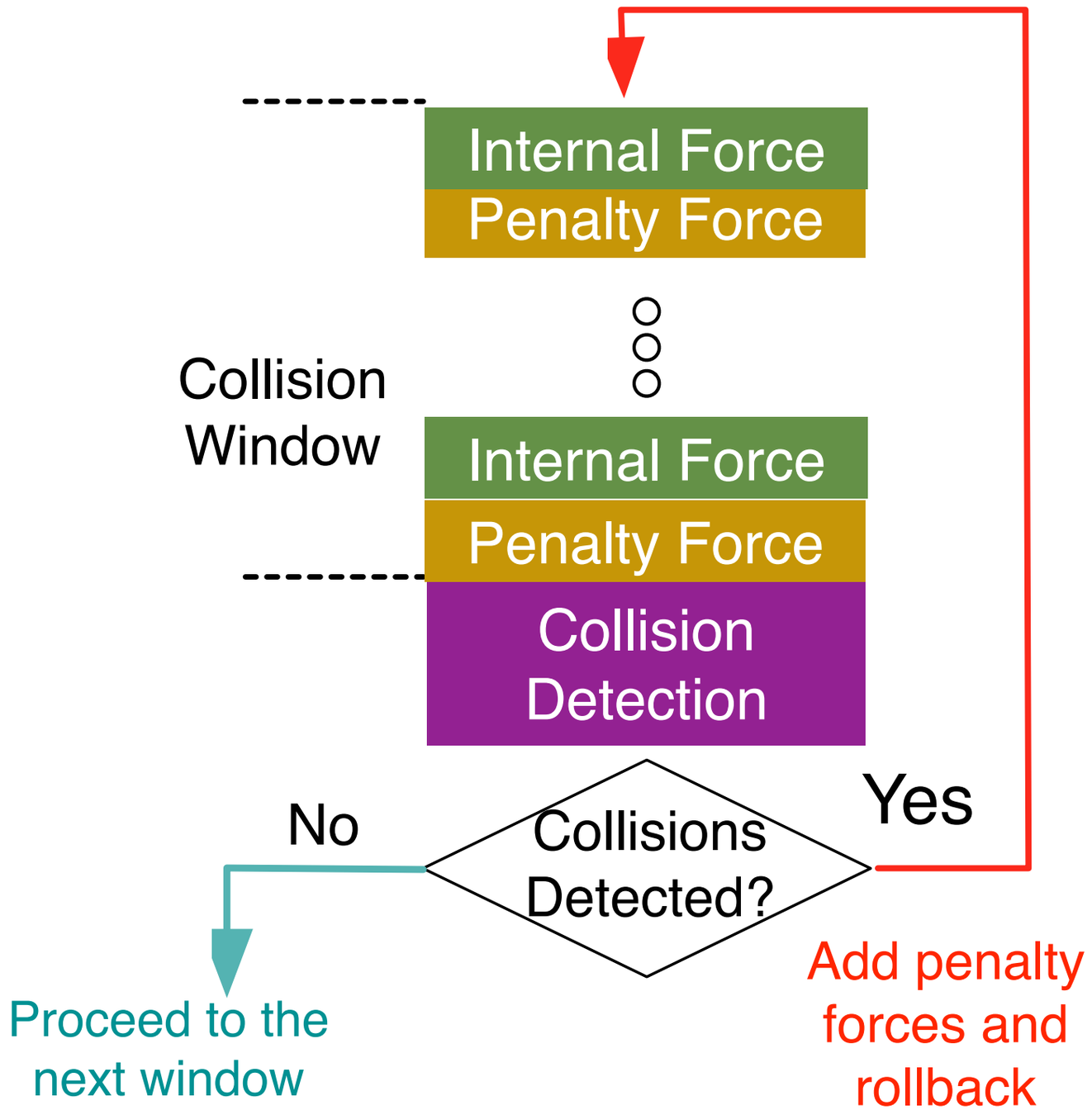
Overall Flow



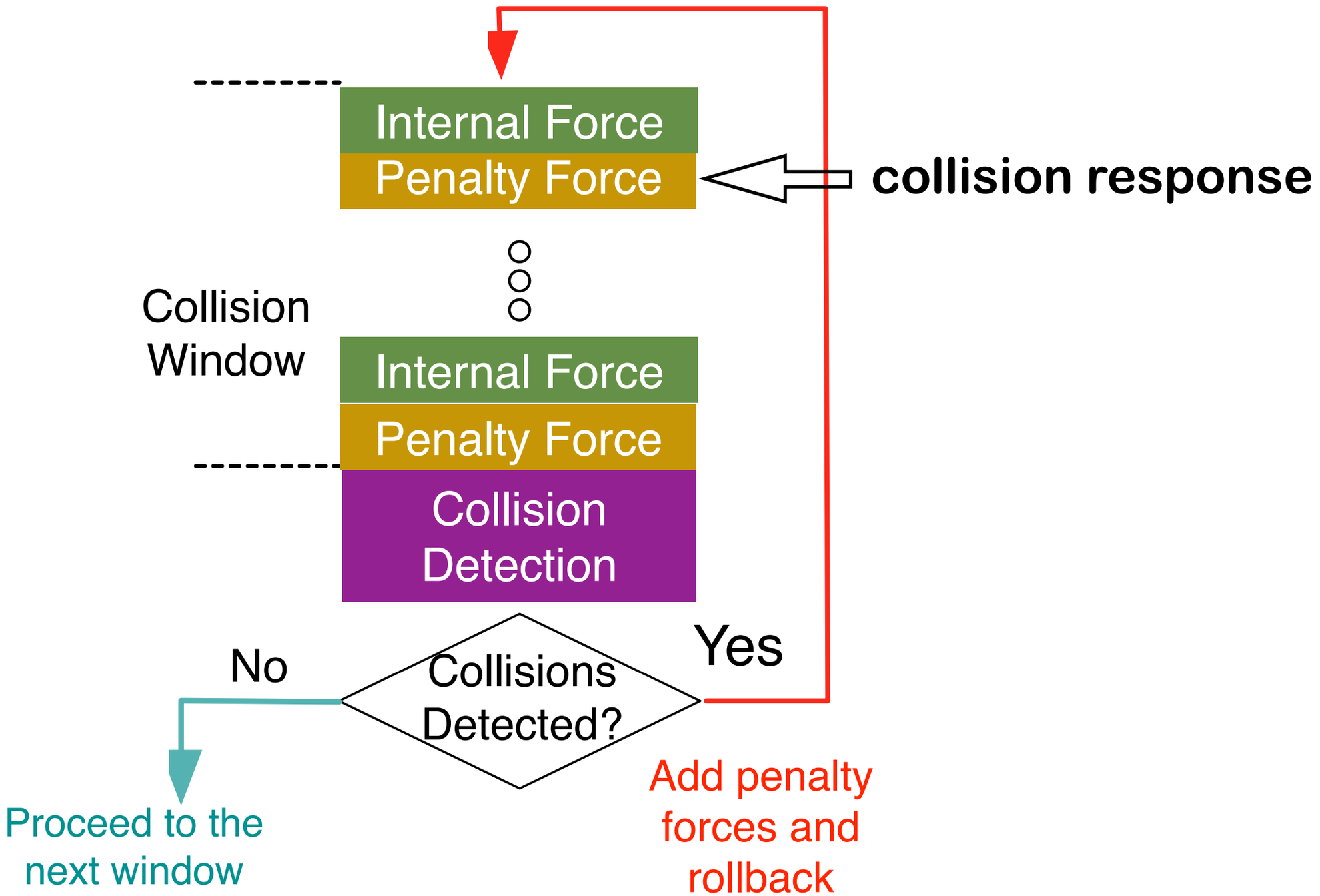
Overall Flow



Overall Flow



Overall Flow



Collision Detection

Collision Detection

Broad Phase

Collision Detection

Broad Phase

Locally inside each partition, we use a **26-DOP hierarchy** to fit the swept volumes of the triangle to detect **potential** collisions.

Collision Detection

Broad Phase

Locally inside each partition, we use a **26-DOP hierarchy** to fit the swept volumes of the triangle to detect **potential** collisions.

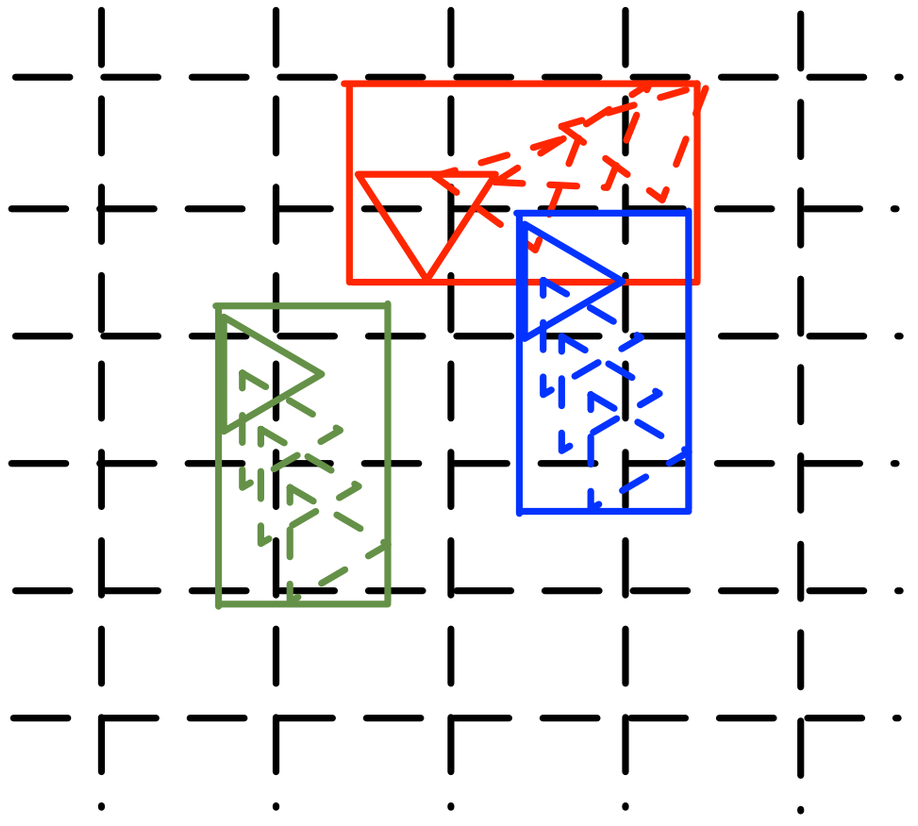
Globally among all the partitions, we fit the trajectory of each triangle to a 3D bounding box and then pass them to the **existing collision detection library** in Charm++.

Collision Detection

Broad Phase

Locally inside each partition, we use a **26-DOP hierarchy** to fit the swept volumes of the triangle to detect **potential** collisions.

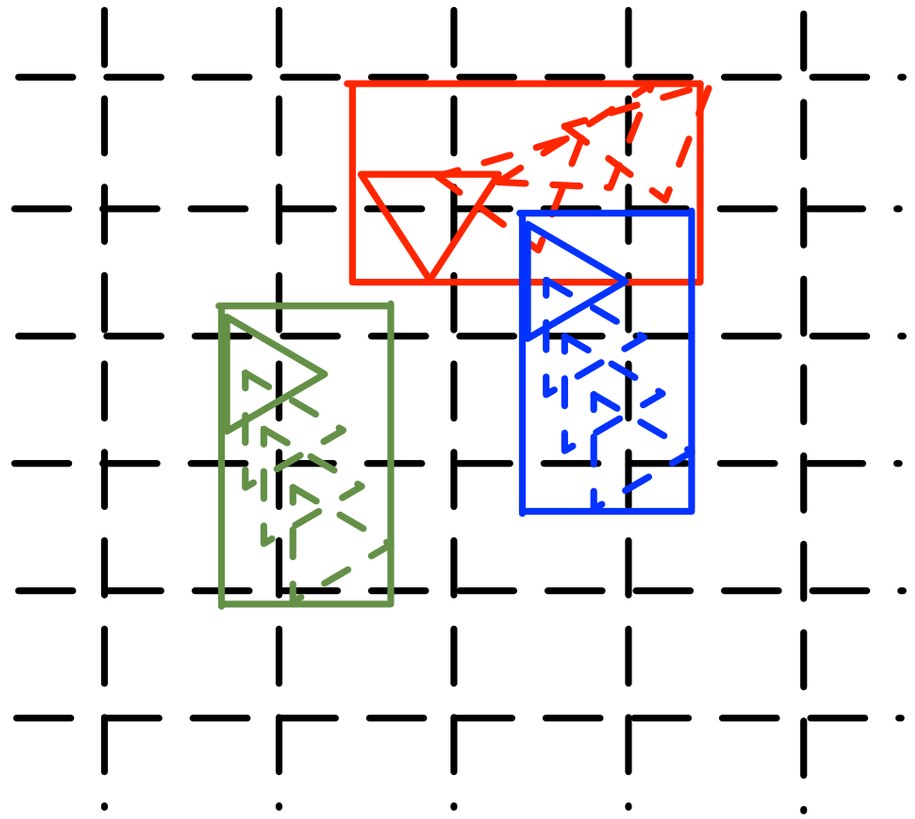
Globally among all the partitions, we fit the trajectory of each triangle to a 3D bounding box and then pass them to the **existing collision detection library** in Charm++.



Collision Detection

Narrow Phase

We apply the space-time separating planes method to filter out potential collisions.

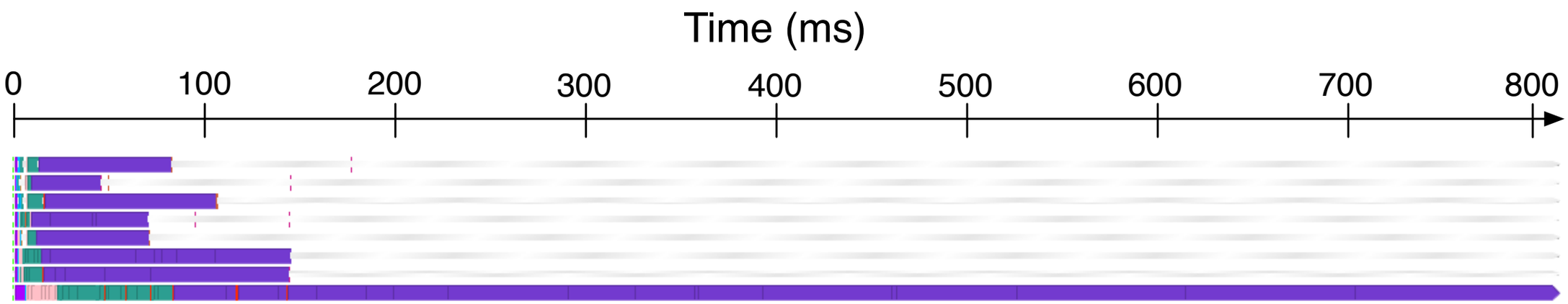


Narrow Phase

First Challenge: **Computation Imbalance**

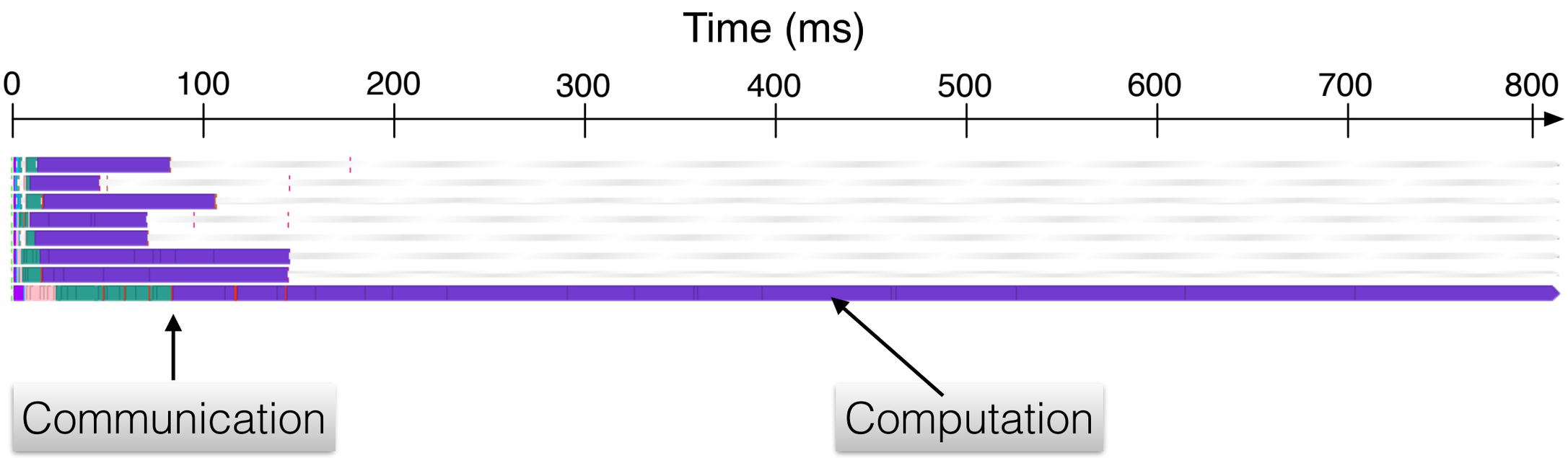
Narrow Phase

First Challenge: **Computation Imbalance**



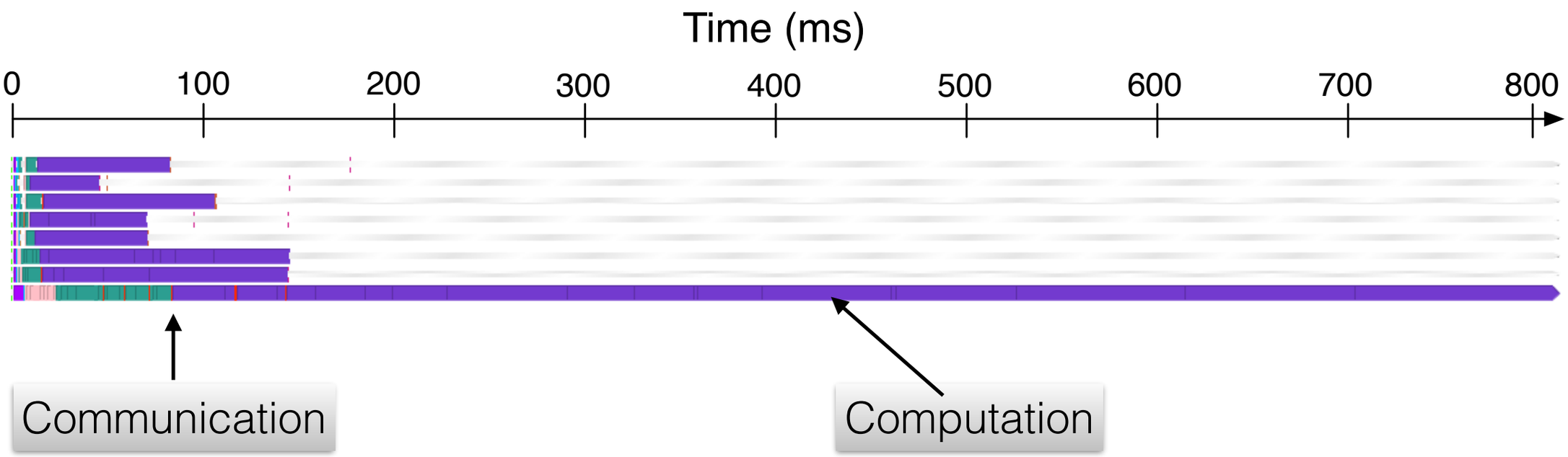
Narrow Phase

First Challenge: **Computation Imbalance**



Narrow Phase

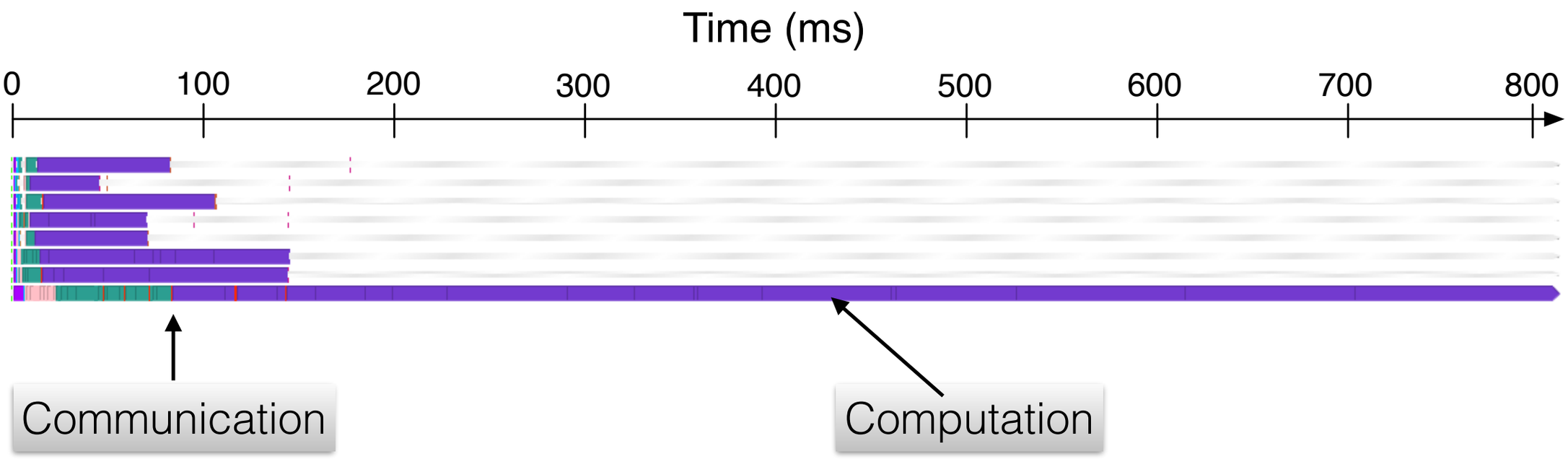
First Challenge: **Computation Imbalance**



Time spent on each potential collision pair is not uniform

Narrow Phase

First Challenge: **Computation Imbalance**

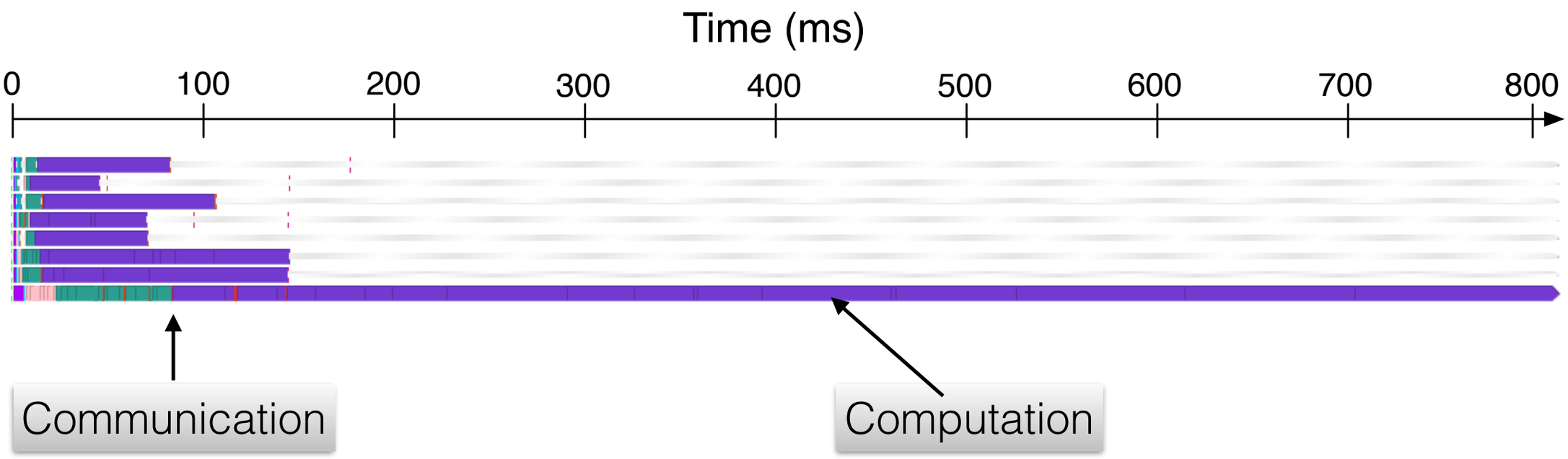


Time spent on each potential collision pair is not uniform

Detection time depends on trajectory length of each vertex in the potential pair

Narrow Phase

First Challenge: **Computation Imbalance**



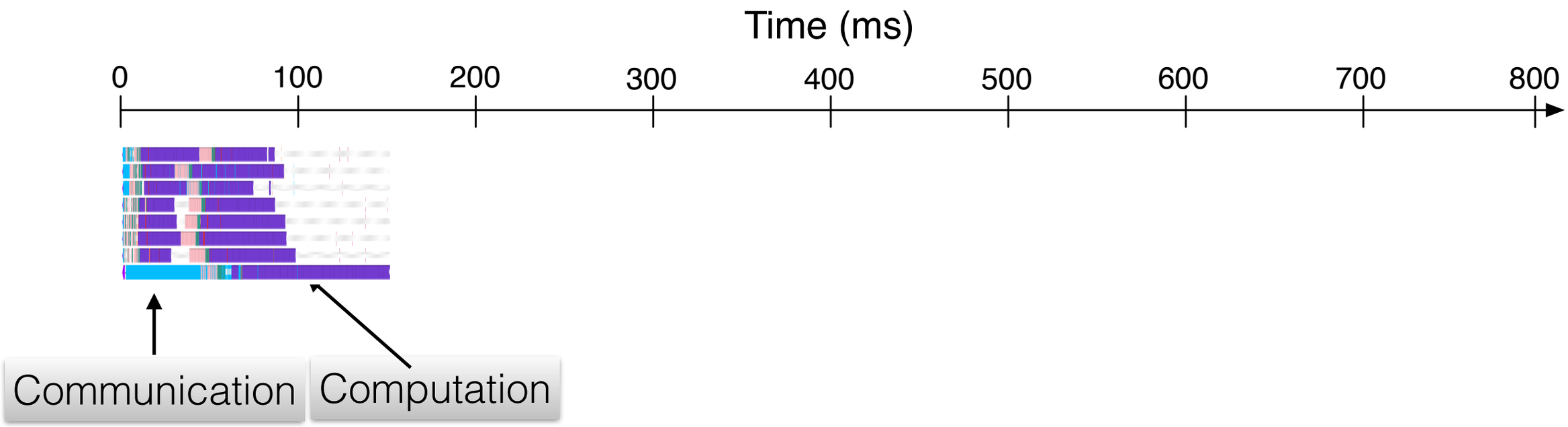
Time spent on each potential collision pair is not uniform

Detection time depends on trajectory length of each vertex in the potential pair

A profiling based load balancer

Narrow Phase

First Challenge: **Computation Imbalance**



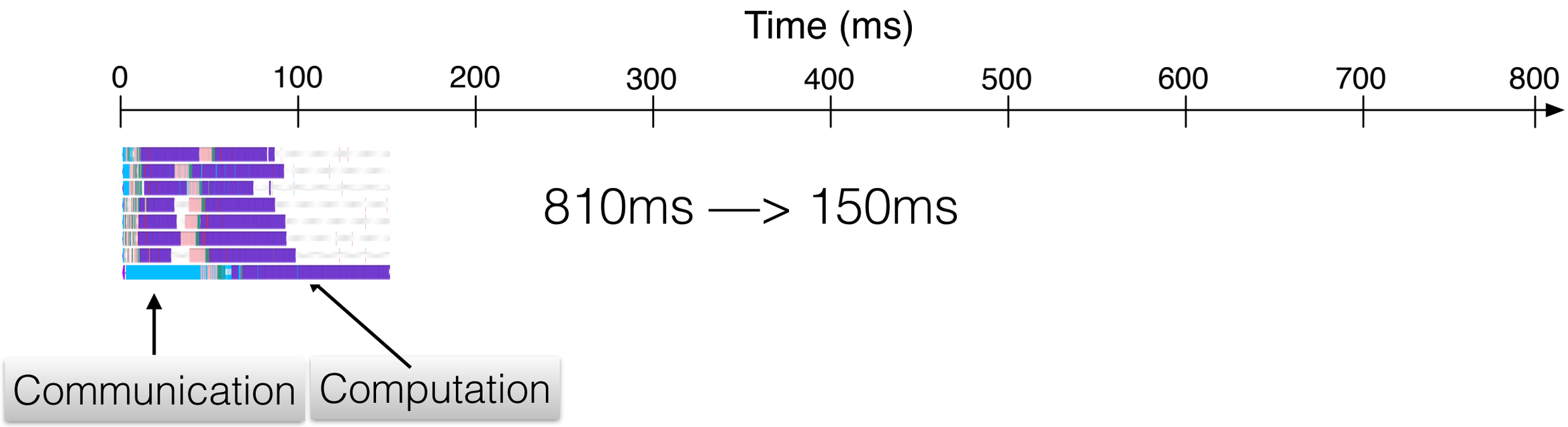
Time spent on each potential collision pair is not uniform

Detection time depends on trajectory length of each vertex in the potential pair

A profiling based load balancer

Narrow Phase

First Challenge: **Computation Imbalance**



810ms —> 150ms

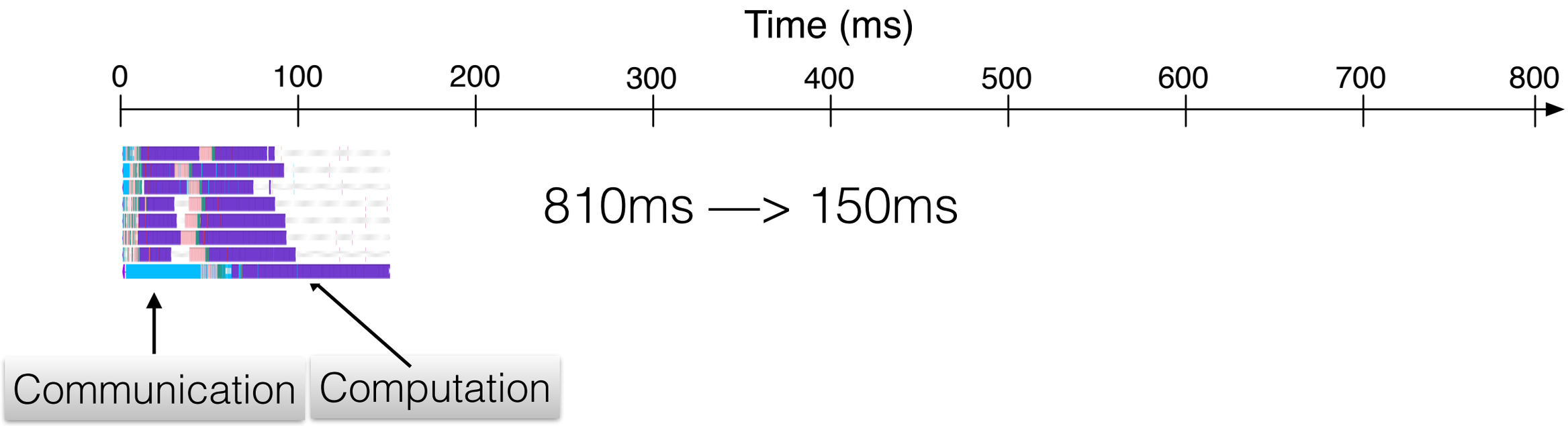
Time spent on each potential collision pair is not uniform

Detection time depends on trajectory length of each vertex in the potential pair

A profiling based load balancer

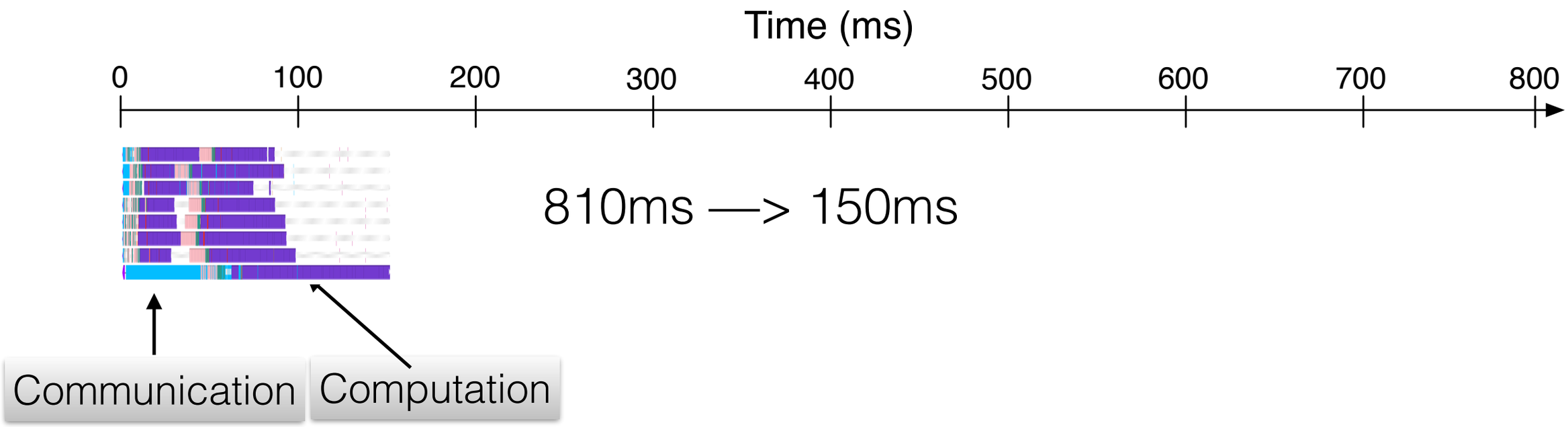
Narrow Phase

Second Challenge: **Communication Imbalance**



Narrow Phase

Second Challenge: **Communication Imbalance**



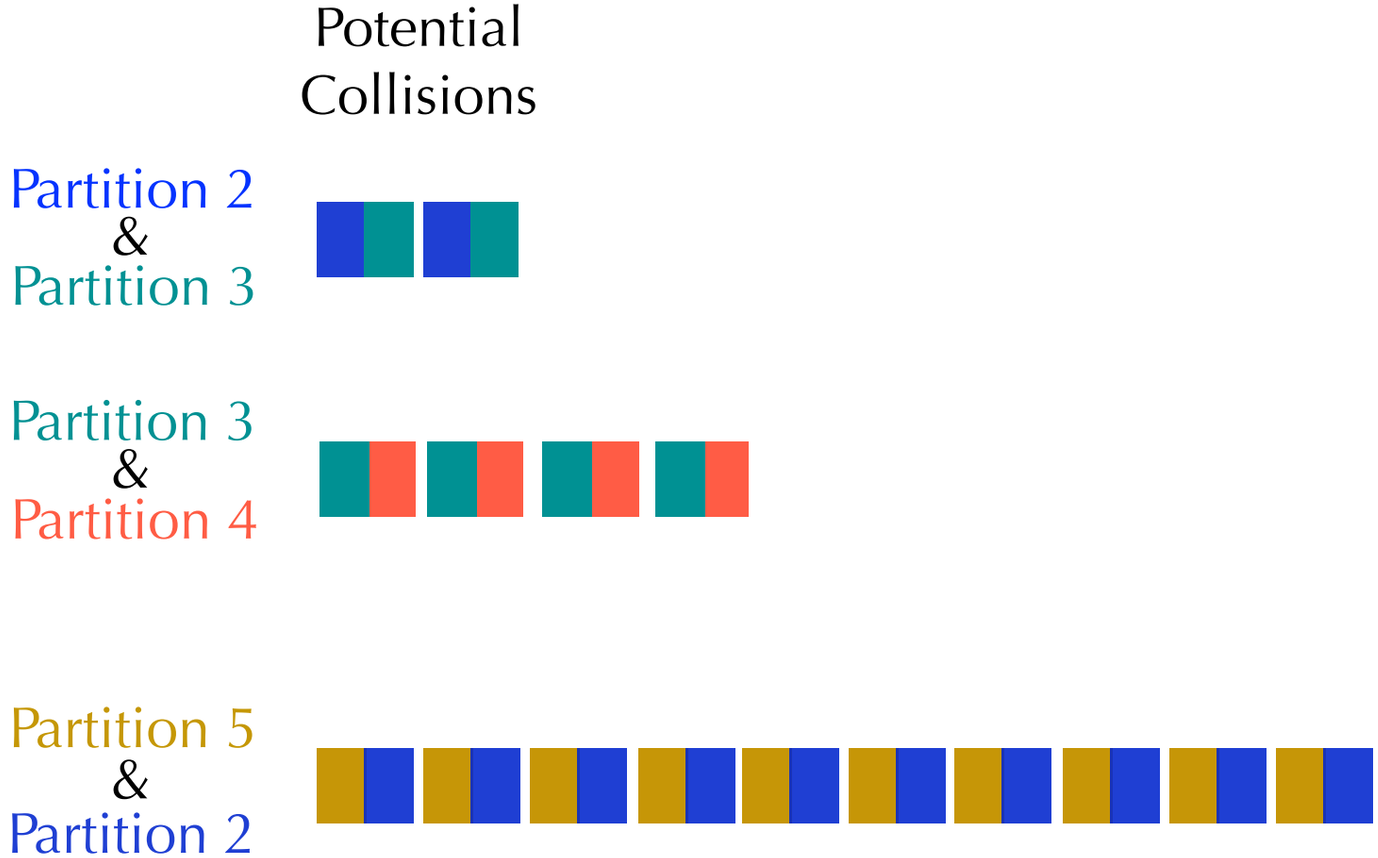
The more potential collision pairs are spread, the more communication requests.

Narrow Phase: Communication Imbalance

Locality Aware Load Balancer

Narrow Phase: Communication Imbalance

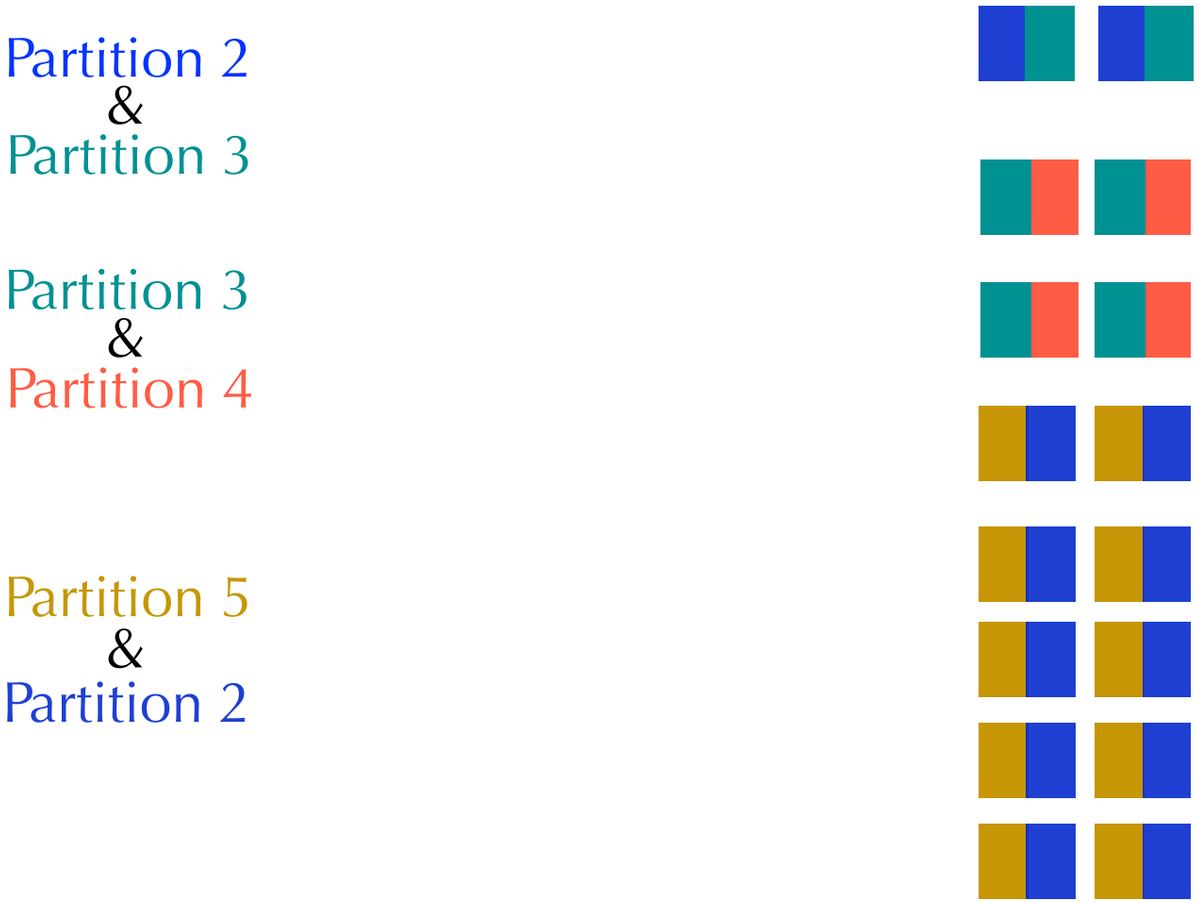
Locality Aware Load Balancer



Narrow Phase: Communication Imbalance

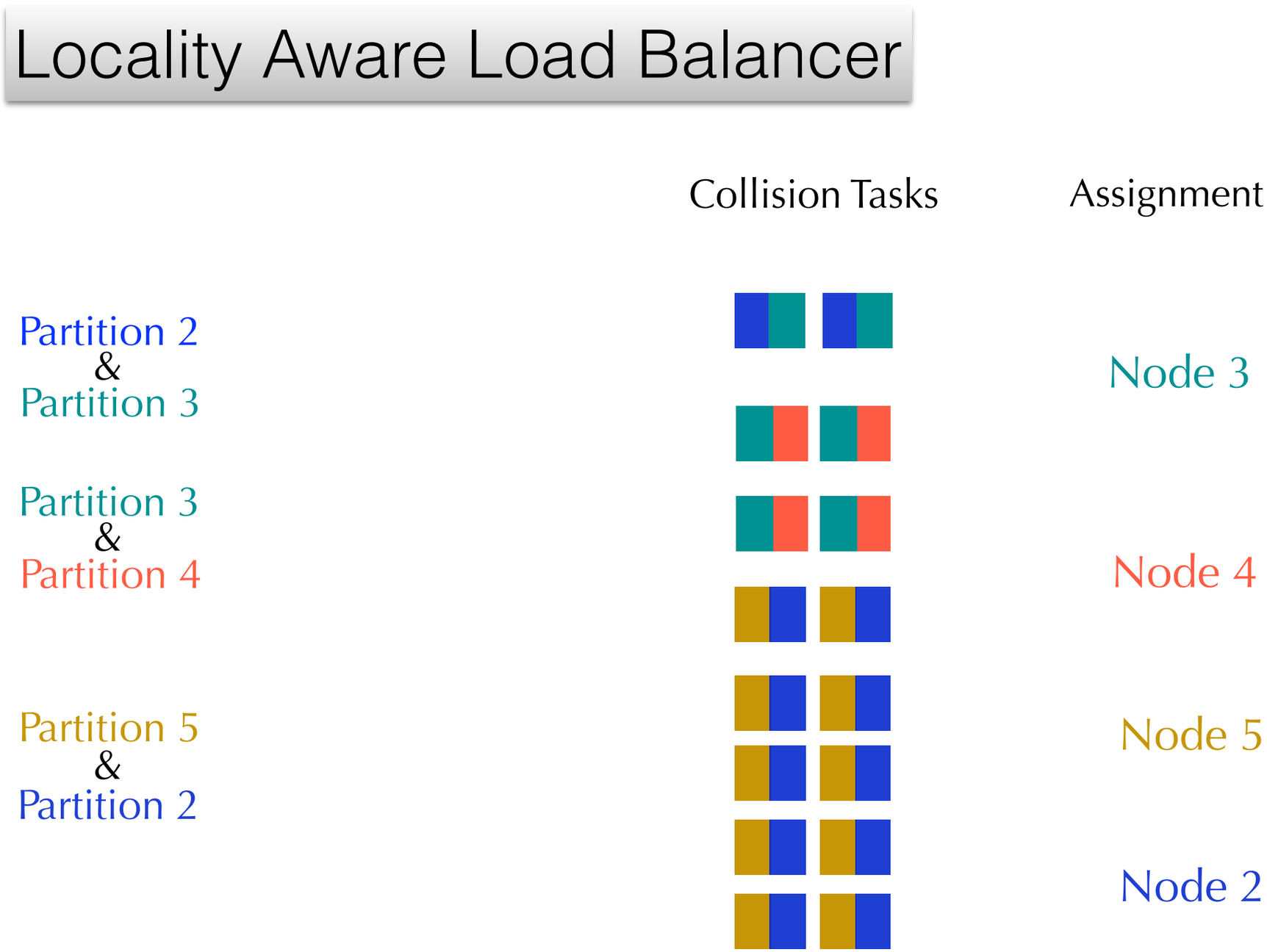
Locality Aware Load Balancer

Collision Tasks



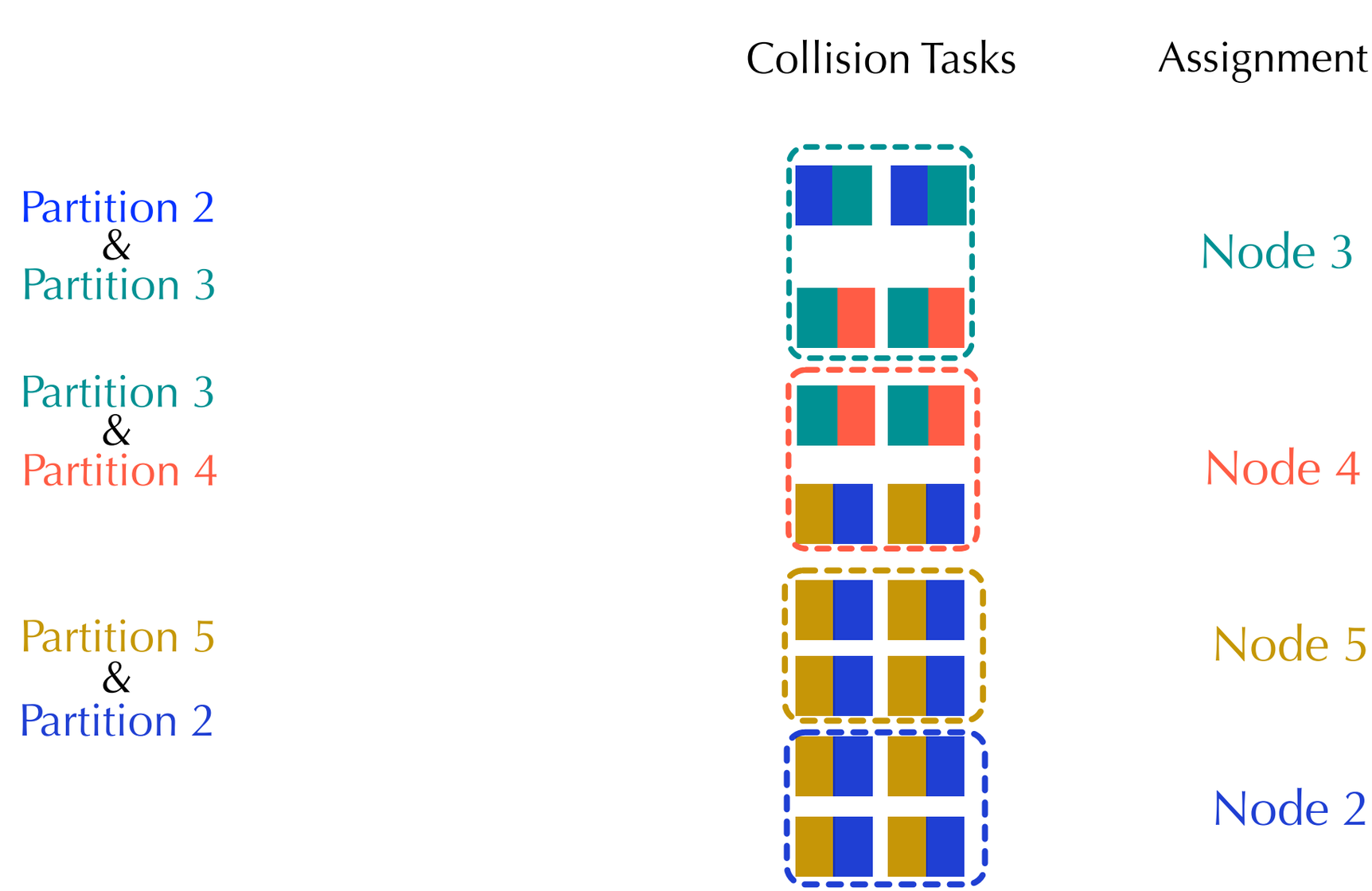
Narrow Phase: Communication Imbalance

Locality Aware Load Balancer



Narrow Phase: Communication Imbalance

Locality Aware Load Balancer



Narrow Phase: Communication Imbalance

Overlapping Computation and Communication

Narrow Phase: Communication Imbalance

Overlapping Computation and Communication

Let's look at the flow on one node

Narrow Phase: Communication Imbalance

Overlapping Computation and Communication

Let's look at the flow on one node

L ← list of potential collision tasks

Narrow Phase: Communication Imbalance

Overlapping Computation and Communication

Let's look at the flow on one node

L ← list of potential collision tasks

1. Send data request for the external vertices in **L**

Narrow Phase: Communication Imbalance

Overlapping Computation and Communication

Let's look at the flow on one node

L ← list of potential collision tasks

1. Send data request for the external vertices in **L**
2. On receiving message **M**

Narrow Phase: Communication Imbalance

Overlapping Computation and Communication

Let's look at the flow on one node

L ← list of potential collision tasks

1. Send data request for the external vertices in **L**
2. On receiving message **M**

M.type()

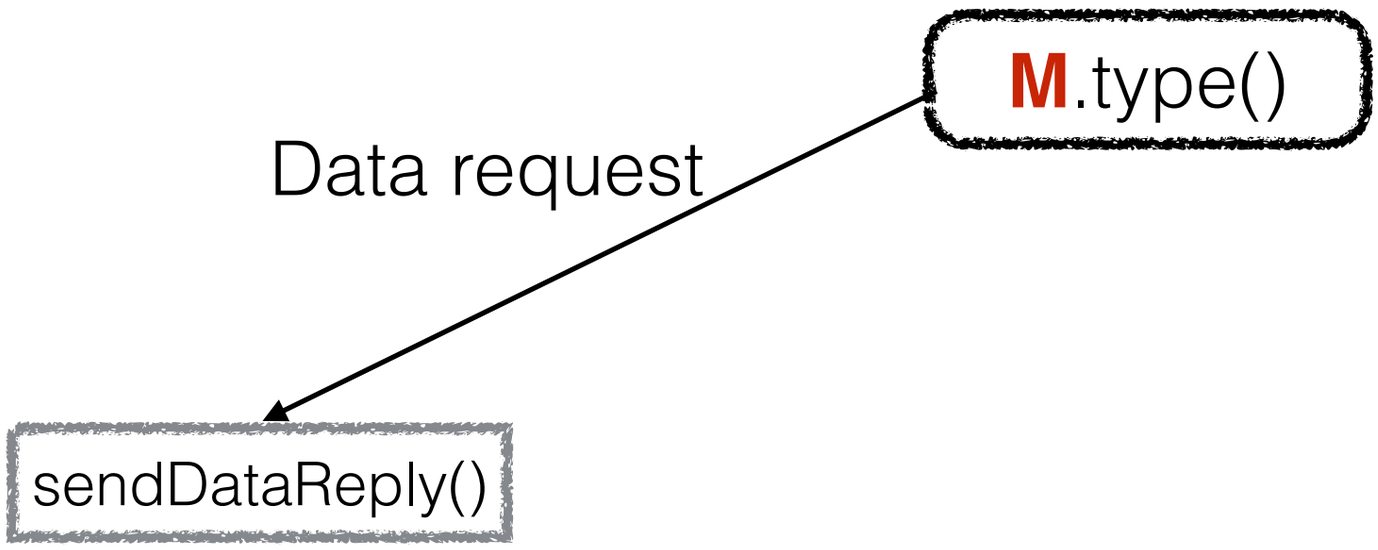
Narrow Phase: Communication Imbalance

Overlapping Computation and Communication

Let's look at the flow on one node

L ← list of potential collision tasks

1. Send data request for the external vertices in **L**
2. On receiving message **M**



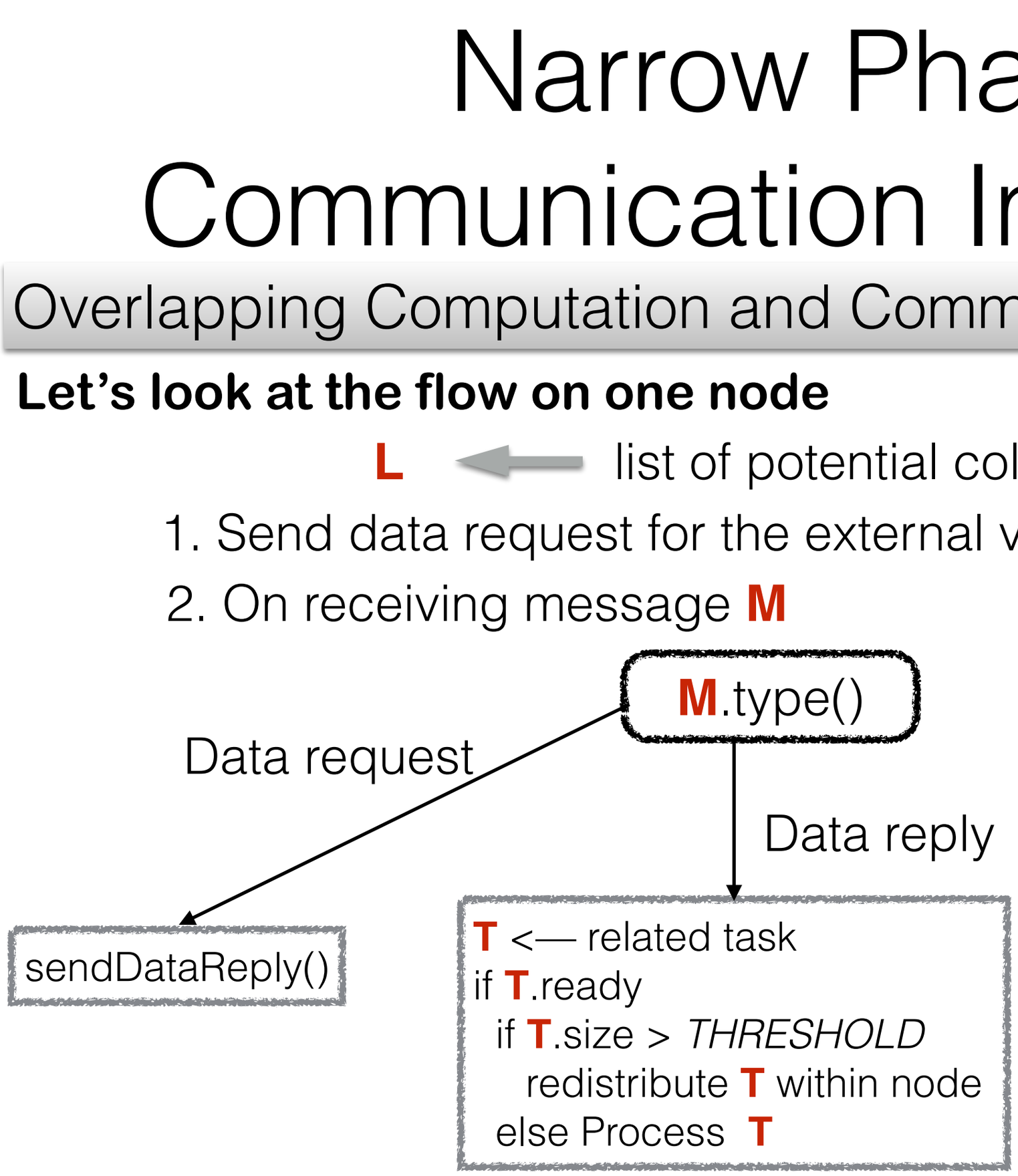
Narrow Phase: Communication Imbalance

Overlapping Computation and Communication

Let's look at the flow on one node

L ← list of potential collision tasks

1. Send data request for the external vertices in **L**
2. On receiving message **M**



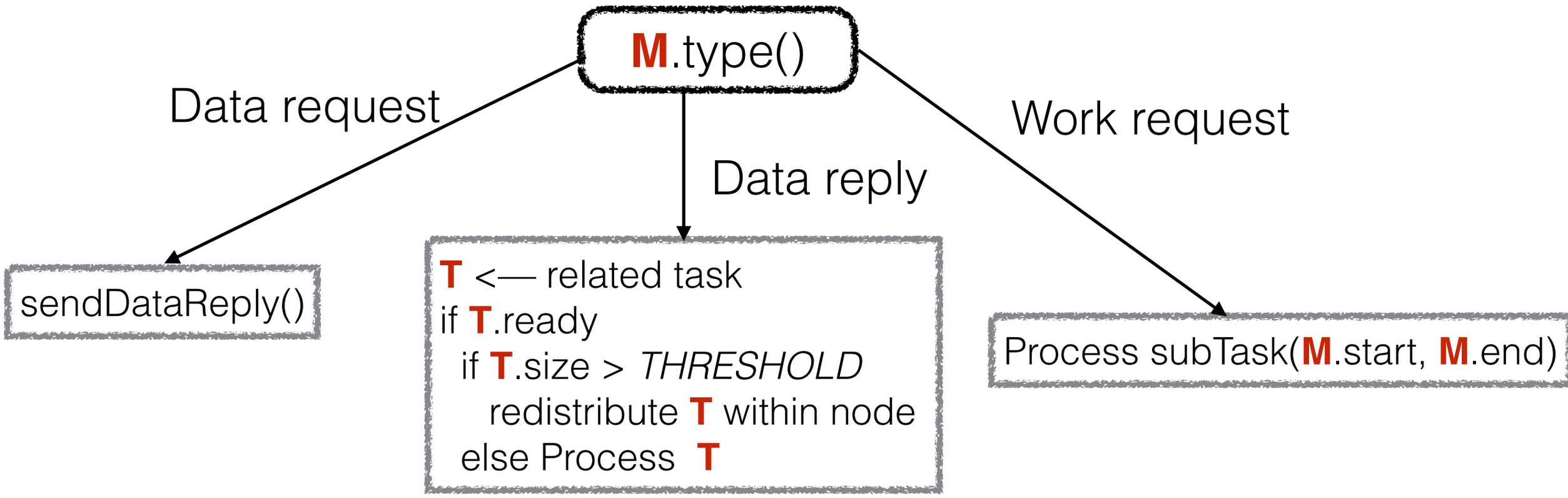
Narrow Phase: Communication Imbalance

Overlapping Computation and Communication

Let's look at the flow on one node

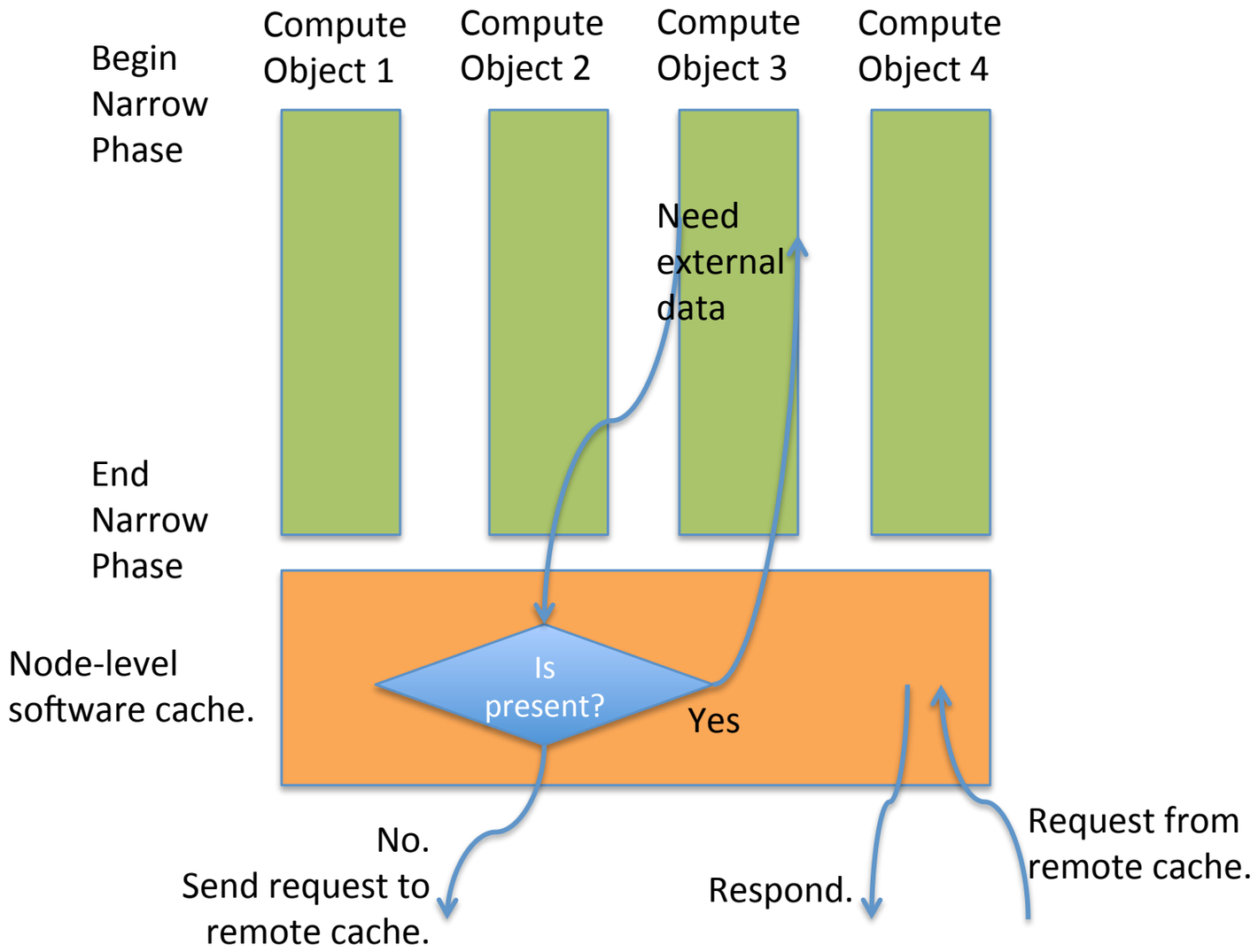
L ← list of potential collision tasks

1. Send data request for the external vertices in **L**
2. On receiving message **M**

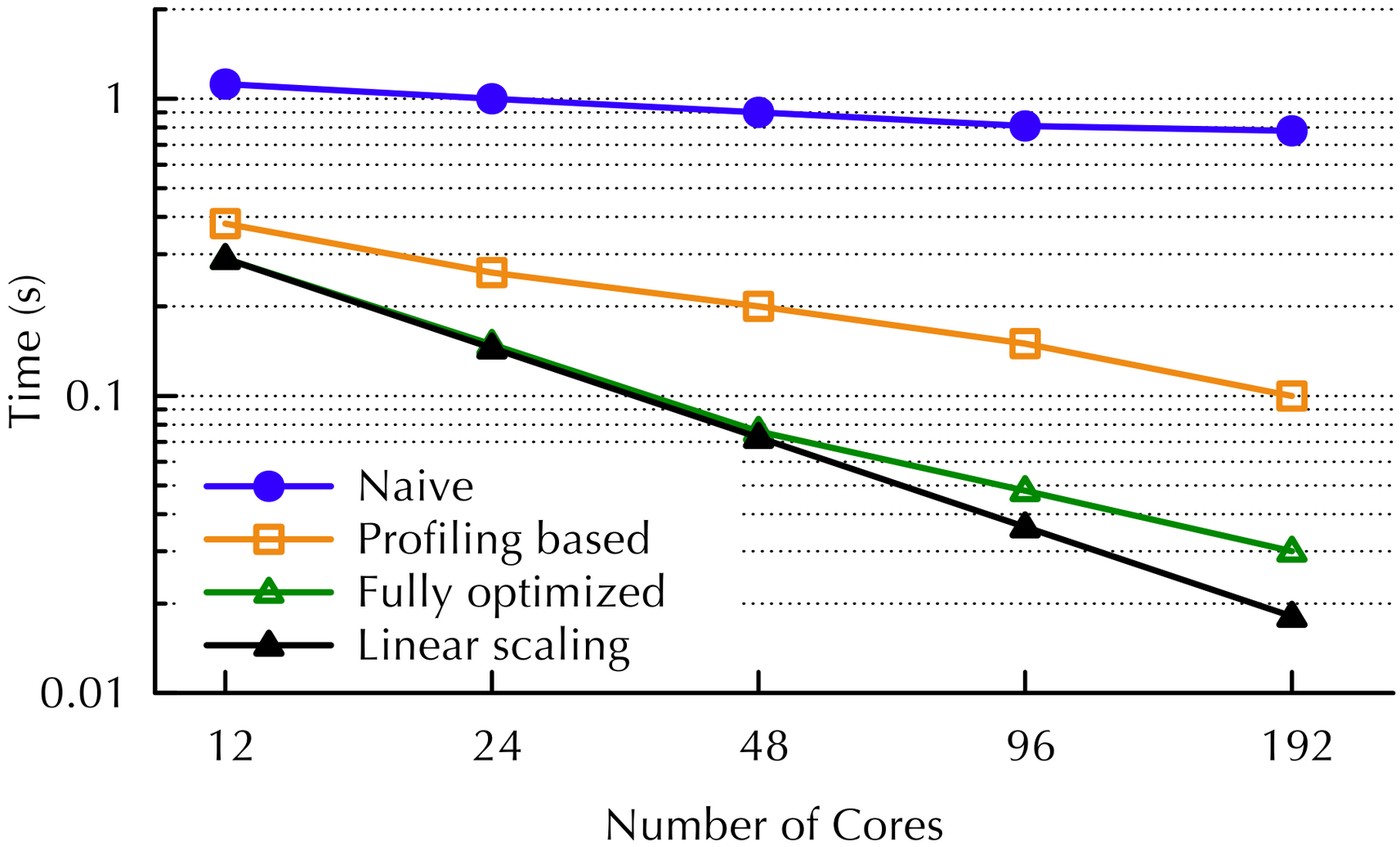


Narrow Phase: Communication Imbalance

Node level data cache

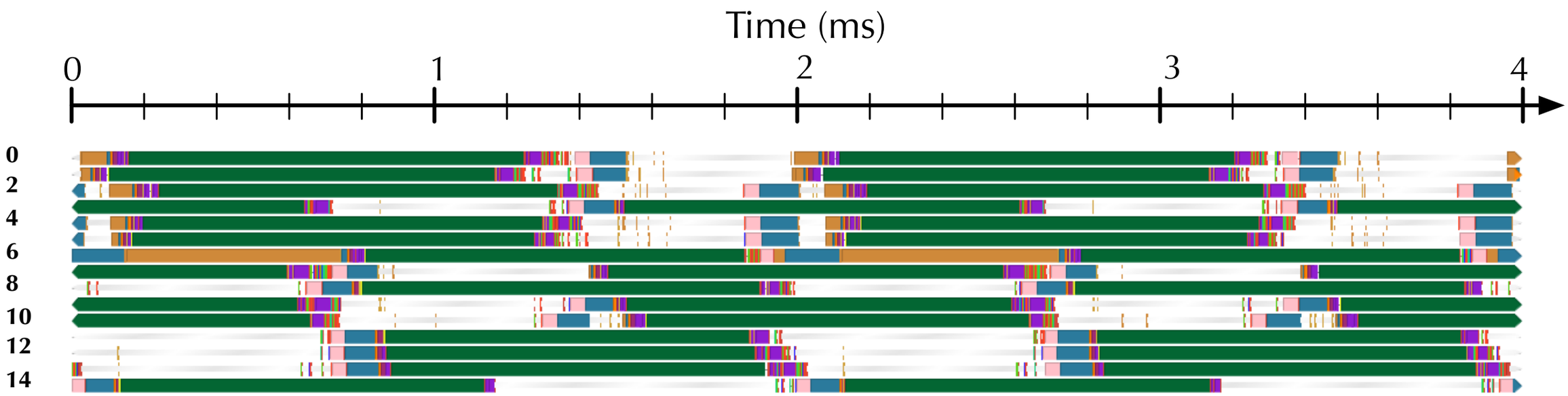


Narrow Phase



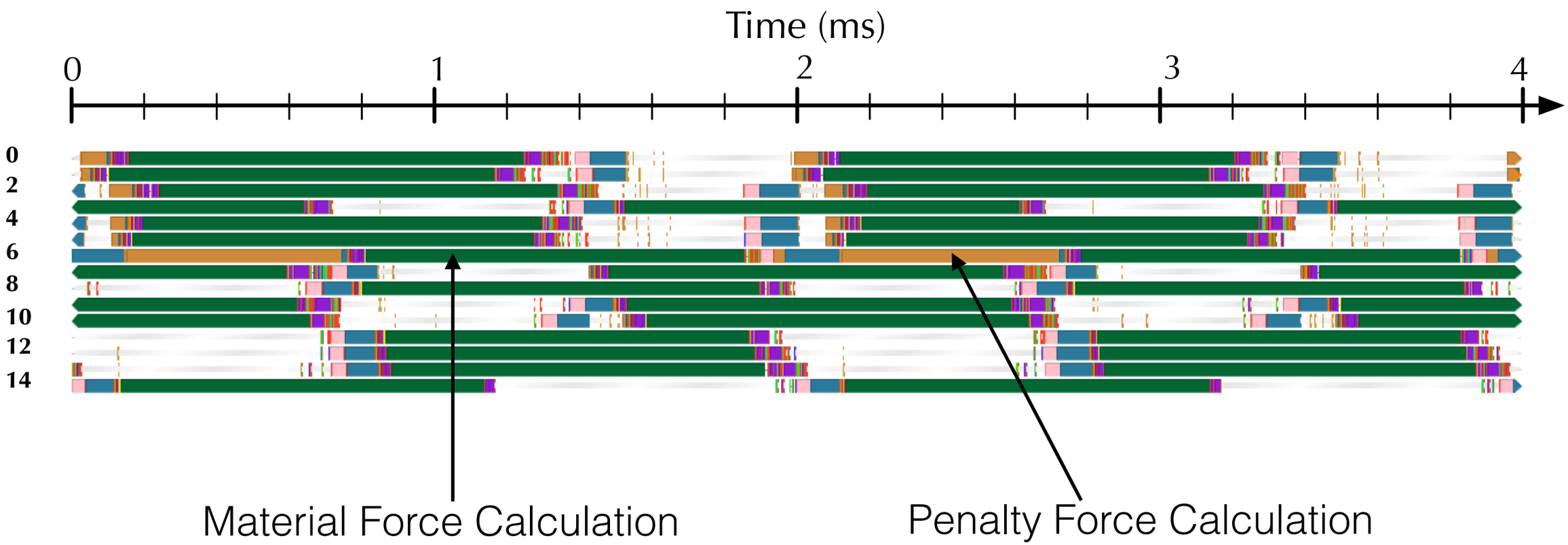
Collision Response

Computation Imbalance



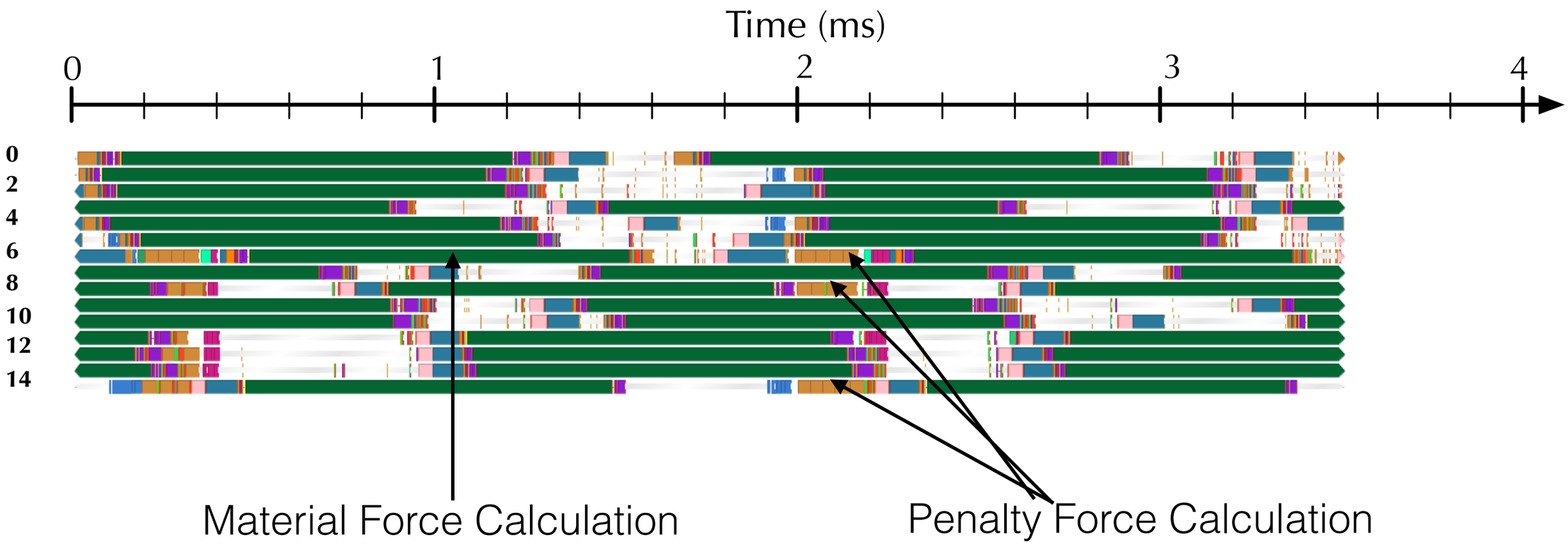
Collision Response

Computation Imbalance



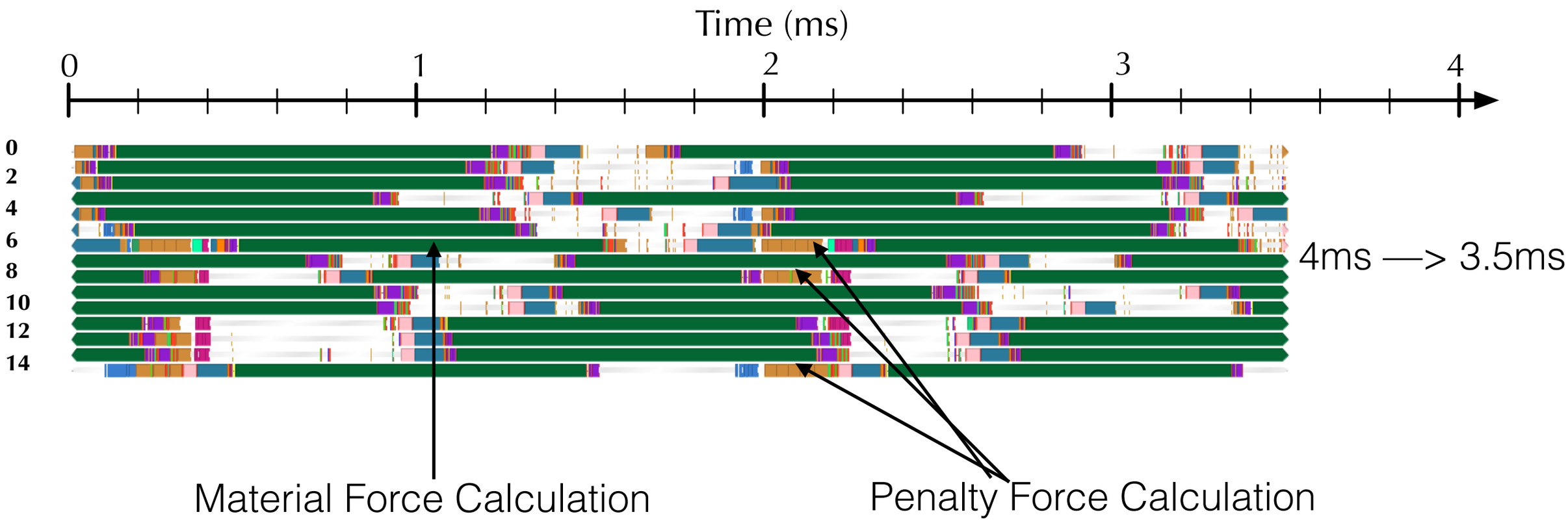
Collision Response

Computation Imbalance



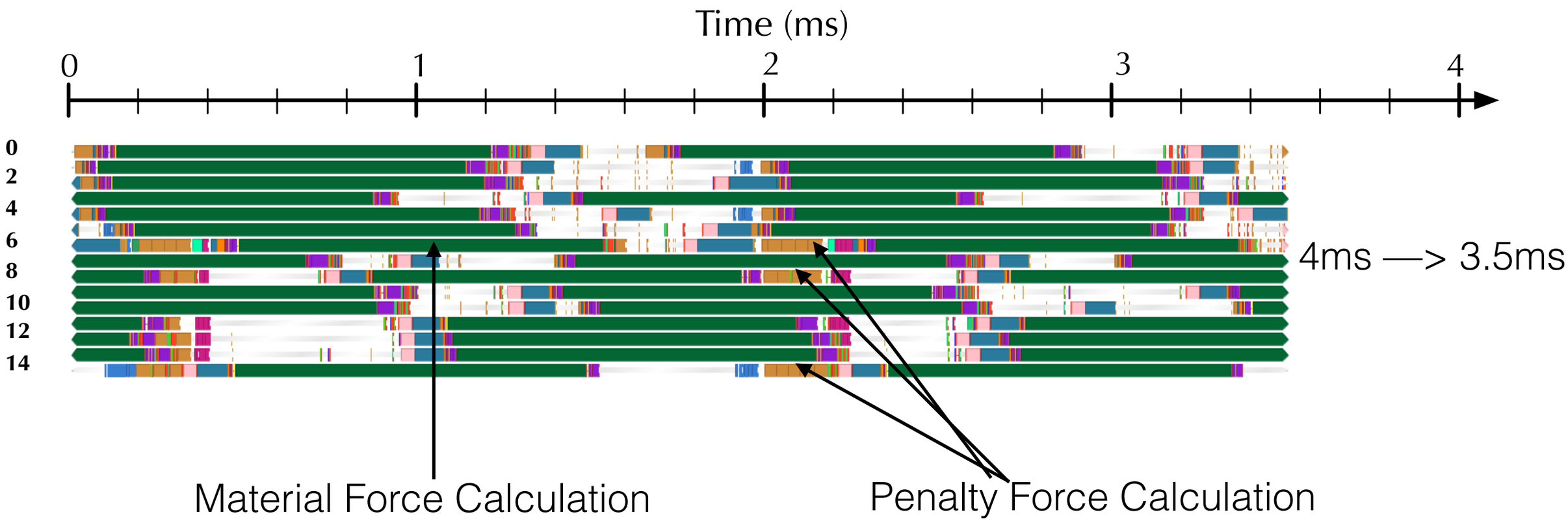
Collision Response

Computation Imbalance



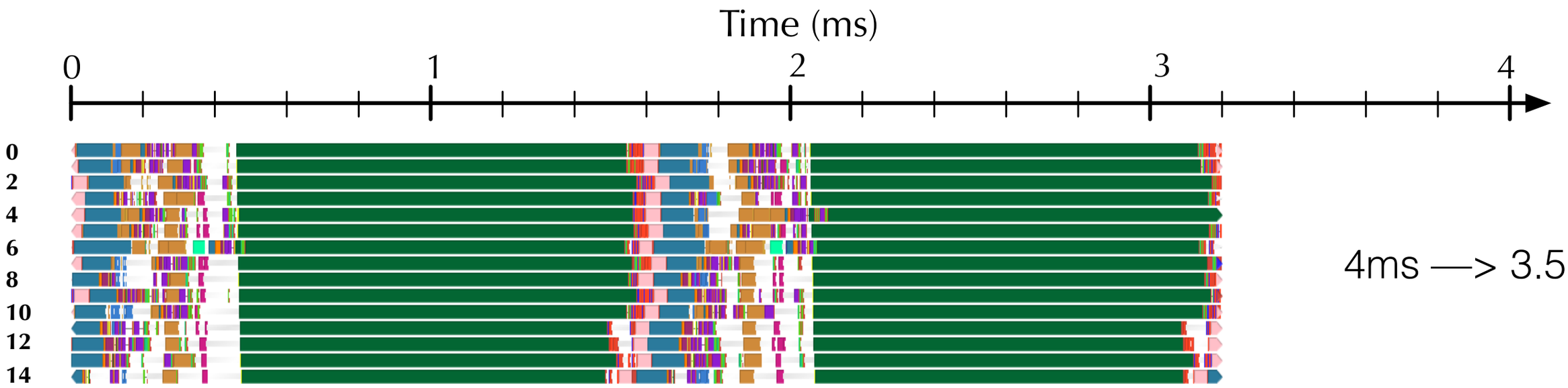
Collision Response

Importance of partial barrier



Collision Response

Importance of partial barrier



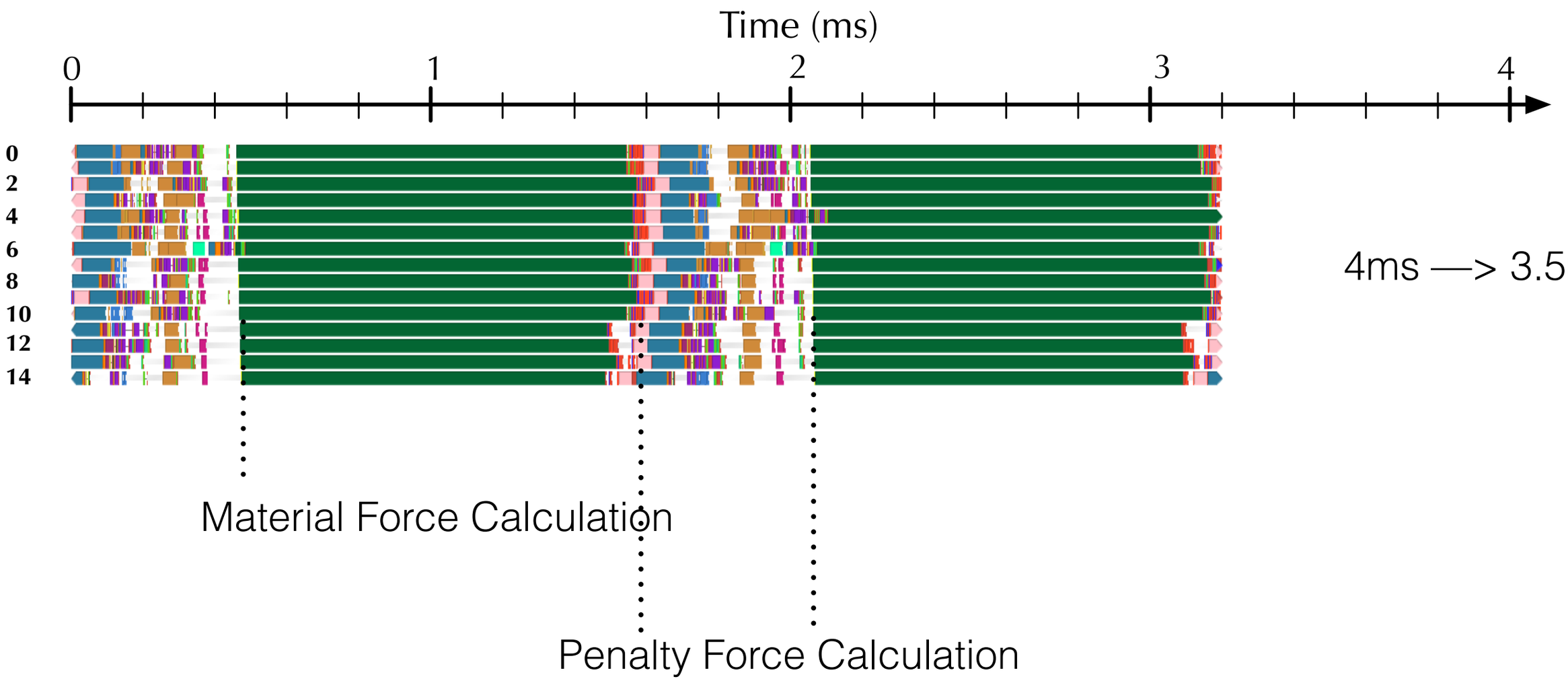
4ms —> 3.5ms

Material Force Calculation

Penalty Force Calculation

Collision Response

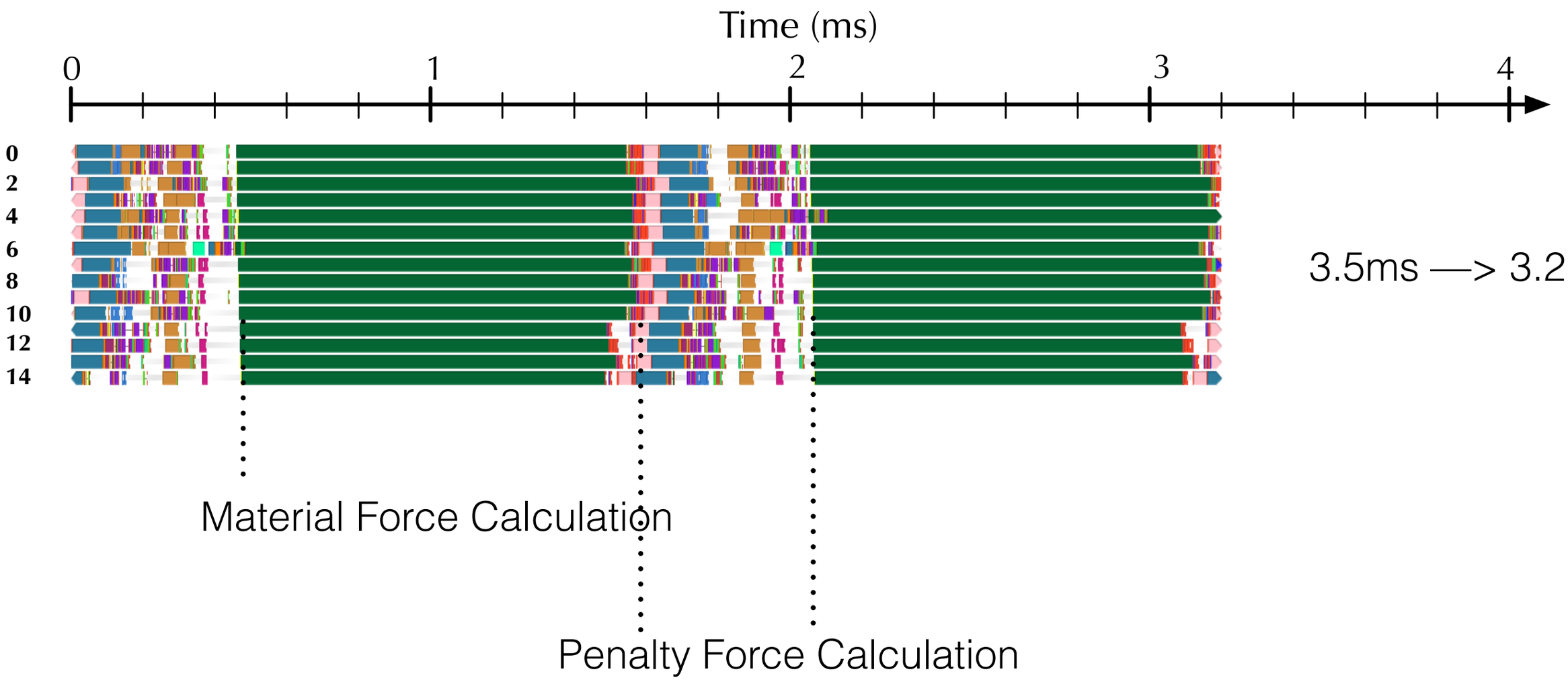
Importance of partial barrier



4ms —> 3.5ms

Collision Response

Importance of partial barrier



3.5ms —> 3.2ms

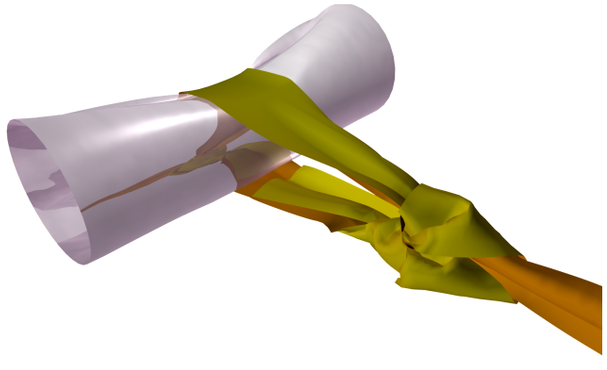
Results

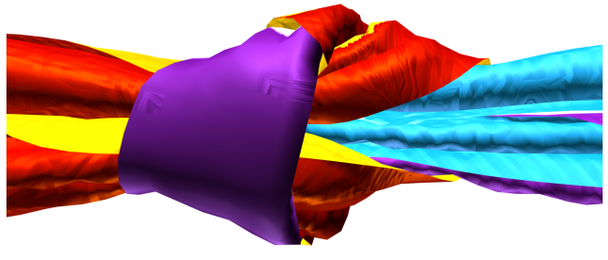
Machines

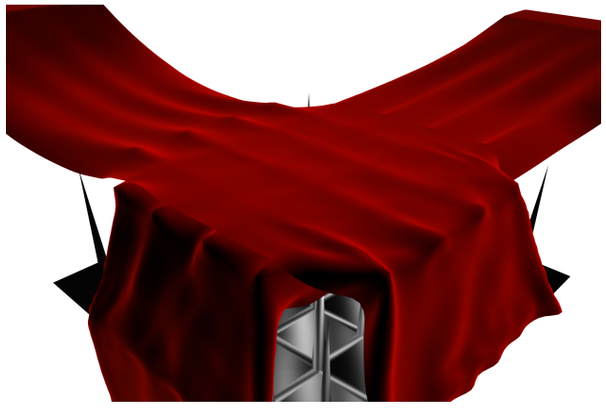
Edison: a Cray XC30, Intel E5-2695@2.4GHz, 12 core Ivy Bridge

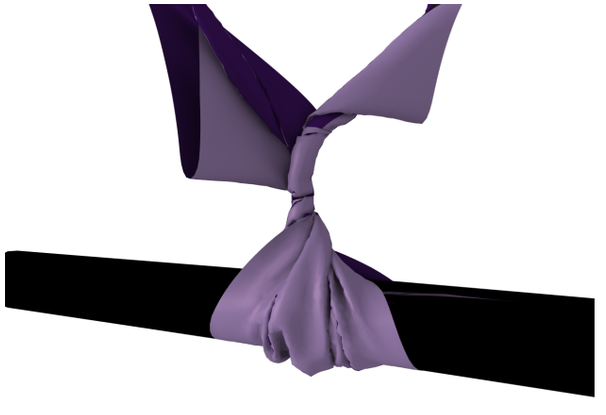
Brickland: a 4 socket system with Intel E7-4890@2.8GHz, 15 core Ivy Bridge

Examples

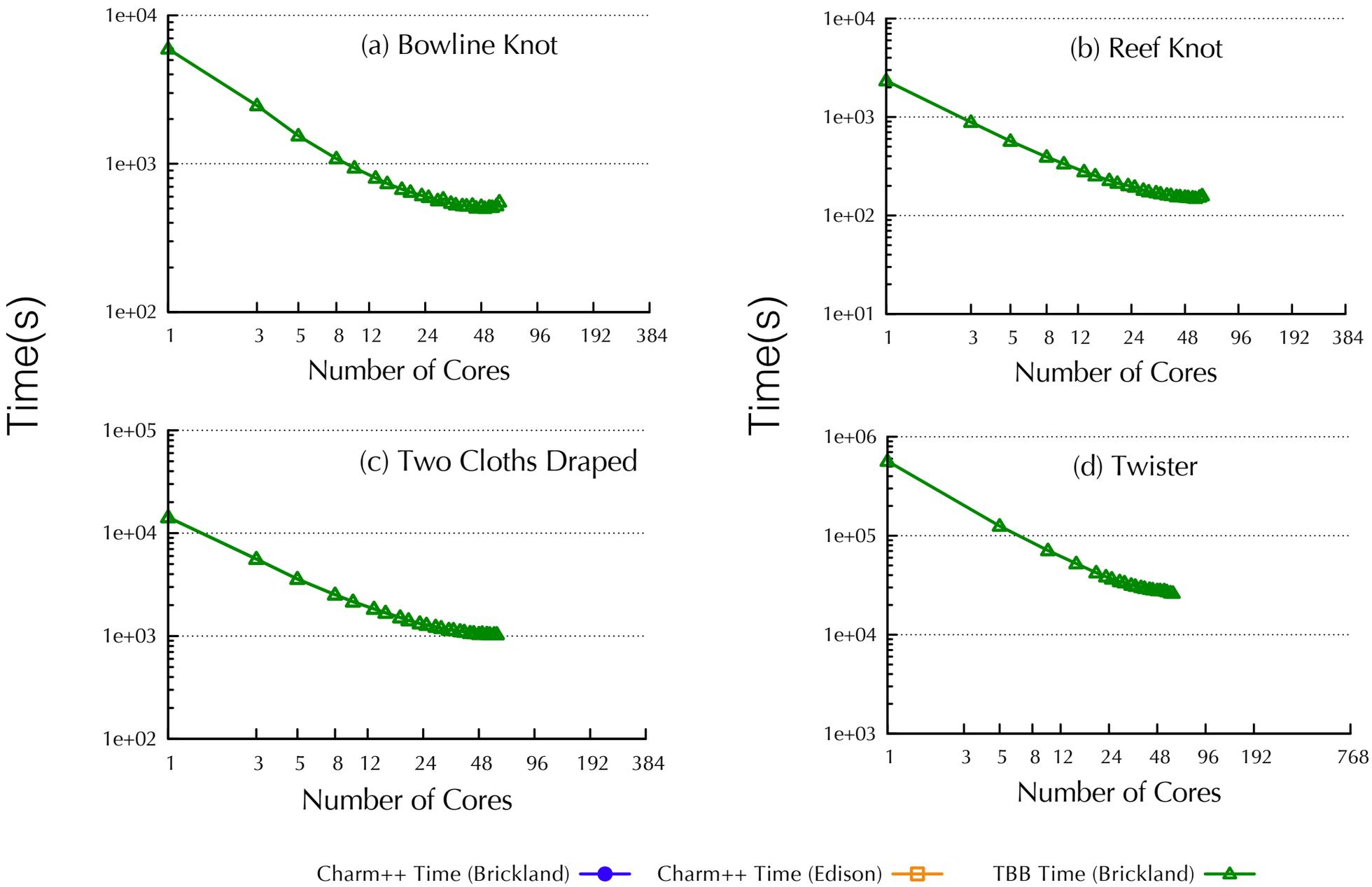




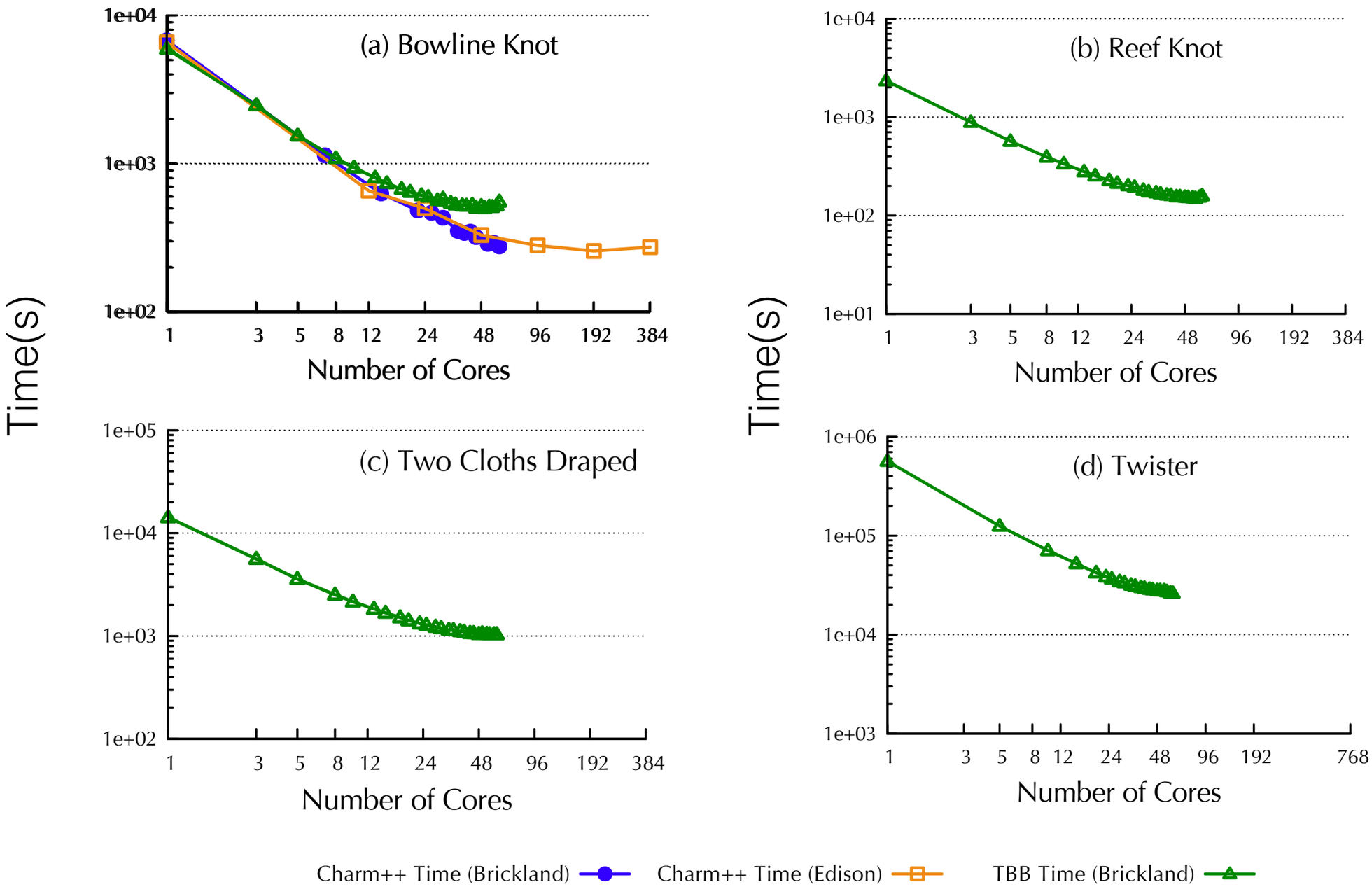




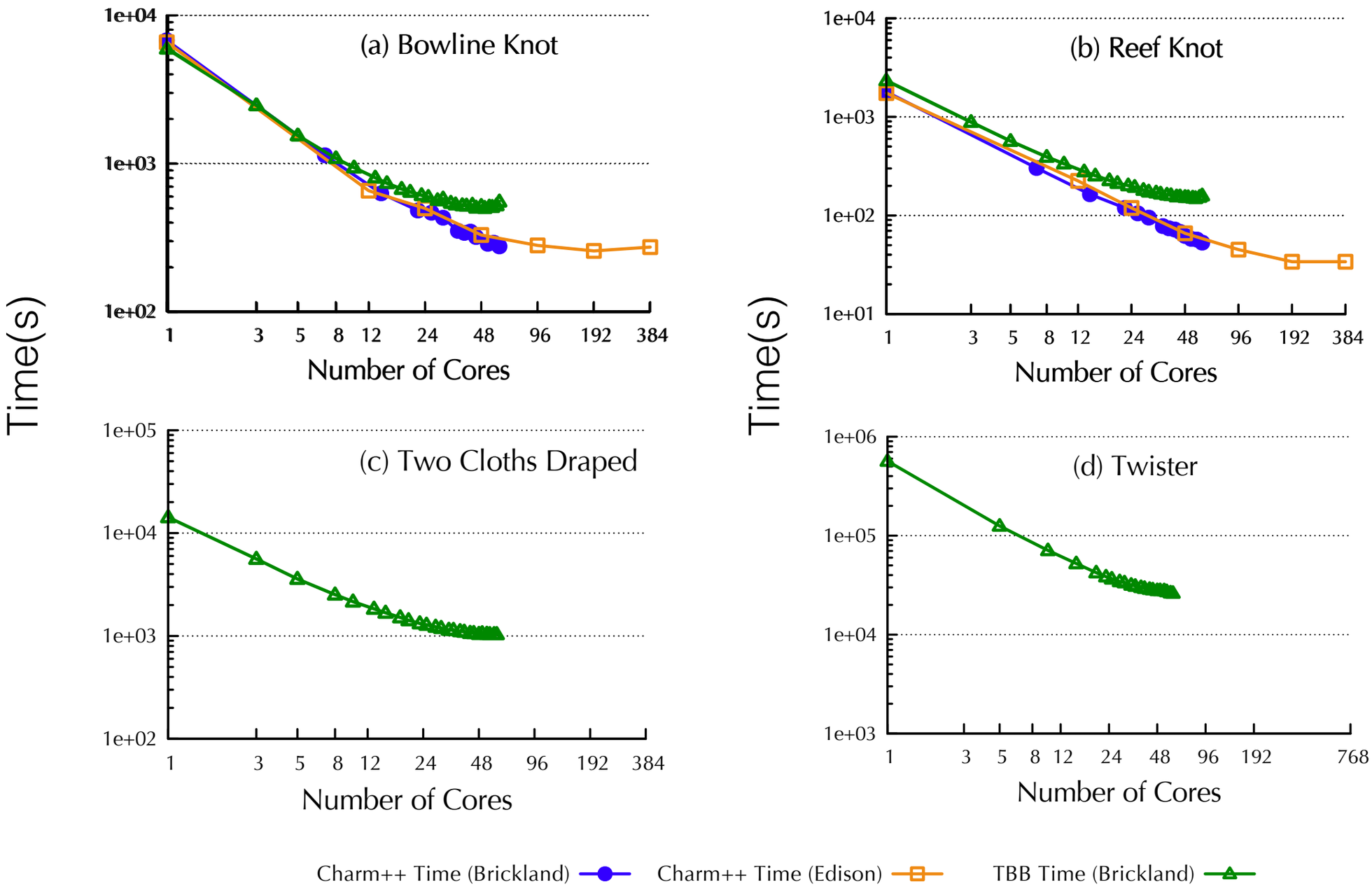
Results


(a) Bowline Knot
(b) Reef Knot
(c) Two Cloths Draped
(d) Twister
Time (s)
Number of Cores
1e+01
1e+02
1e+03
1e+04
1e+05
1e+06
1
3
5
8
12
24
48
96
192
384
768
Charm++ Time (Brickland)
Charm++ Time (Edison)
TBB Time (Brickland)

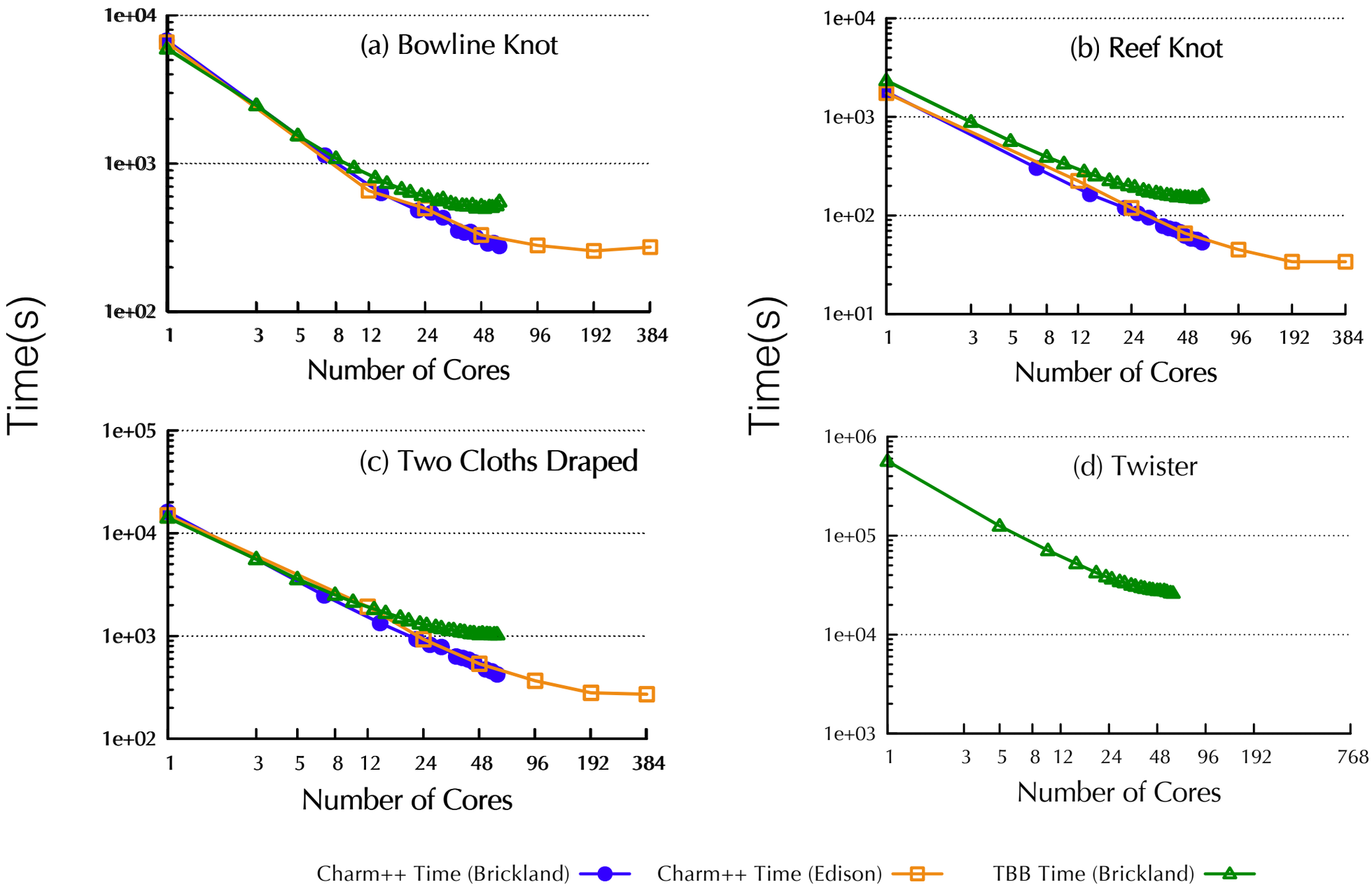
Results


(a) Bowline Knot
(b) Reef Knot
(c) Two Cloths Draped
(d) Twister
Time (s)
Number of Cores
1e+01
1e+02
1e+03
1e+04
1e+05
1e+06
1
3
5
8
12
24
48
96
192
384
768
Charm++ Time (Brickland)
Charm++ Time (Edison)
TBB Time (Brickland)

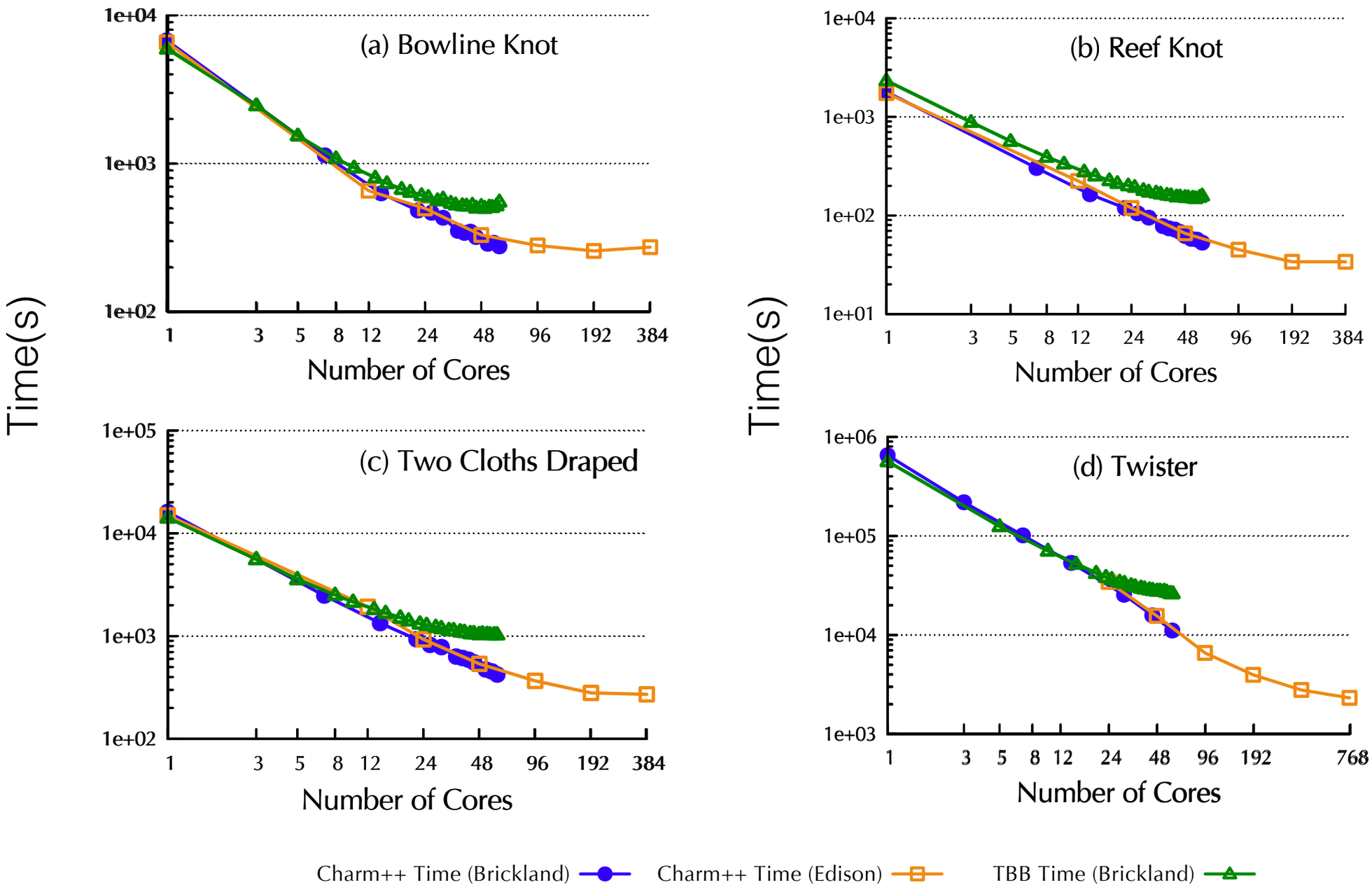
Results


(a) Bowline Knot
(b) Reef Knot
(c) Two Cloths Draped
(d) Twister
Time (s)
Number of Cores
1e+01
1e+02
1e+03
1e+04
1e+05
1e+06
1
3
5
8
12
24
48
96
192
384
768
Charm++ Time (Brickland)
Charm++ Time (Edison)
TBB Time (Brickland)

Results


(a) Bowline Knot
(b) Reef Knot
(c) Two Cloths Draped
(d) Twister
Time (s)
Number of Cores
1e+01
1e+02
1e+03
1e+04
1e+05
1e+06
1
3
5
8
12
24
48
96
192
384
768
Charm++ Time (Brickland)
Charm++ Time (Edison)
TBB Time (Brickland)

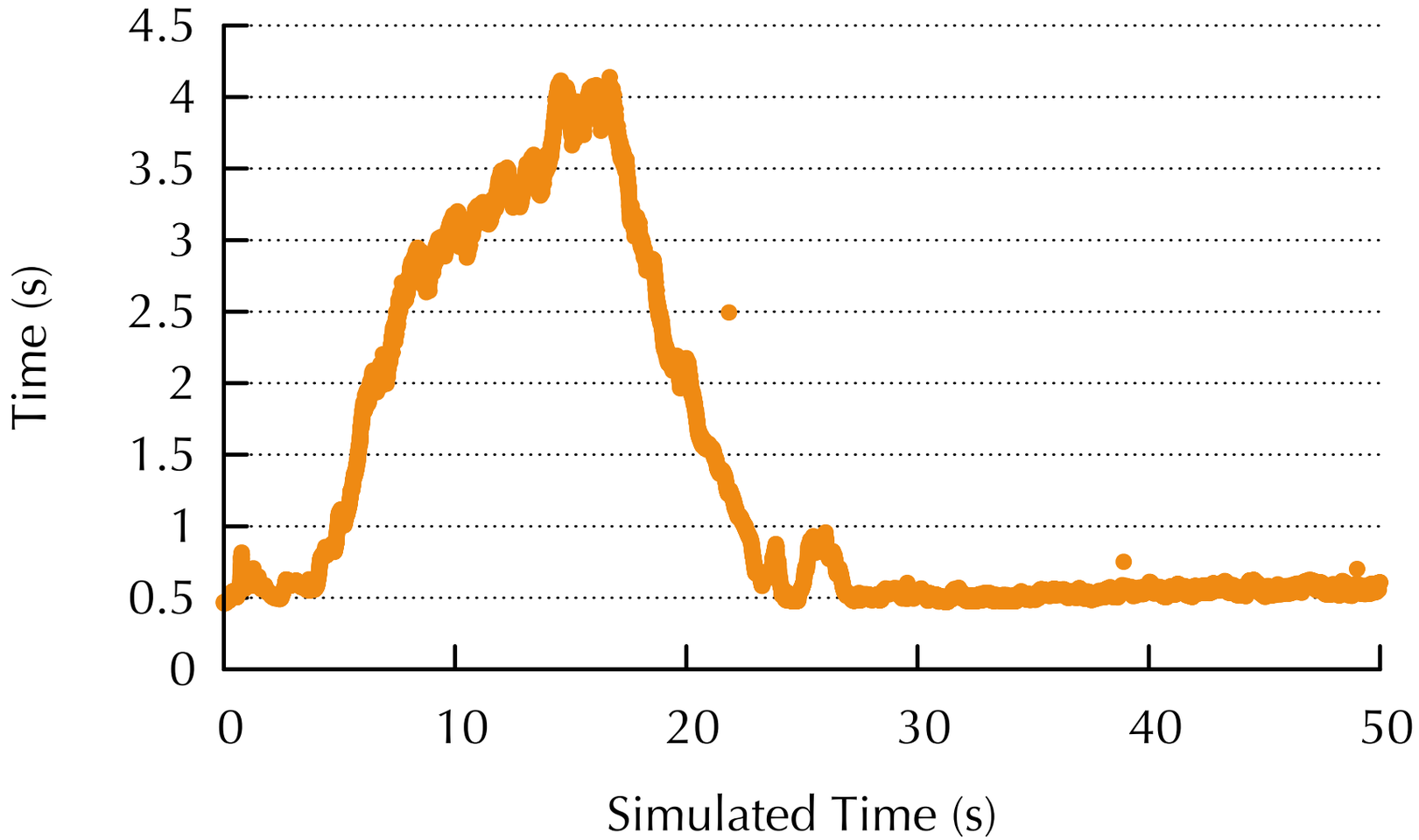
Results


(a) Bowline Knot
(b) Reef Knot
(c) Two Cloths Draped
(d) Twister
Time (s)
Number of Cores
1e+01
1e+02
1e+03
1e+04
1e+05
1e+06
1
3
5
8
12
24
48
96
192
384
768
Charm++ Time (Brickland)
Charm++ Time (Edison)
TBB Time (Brickland)

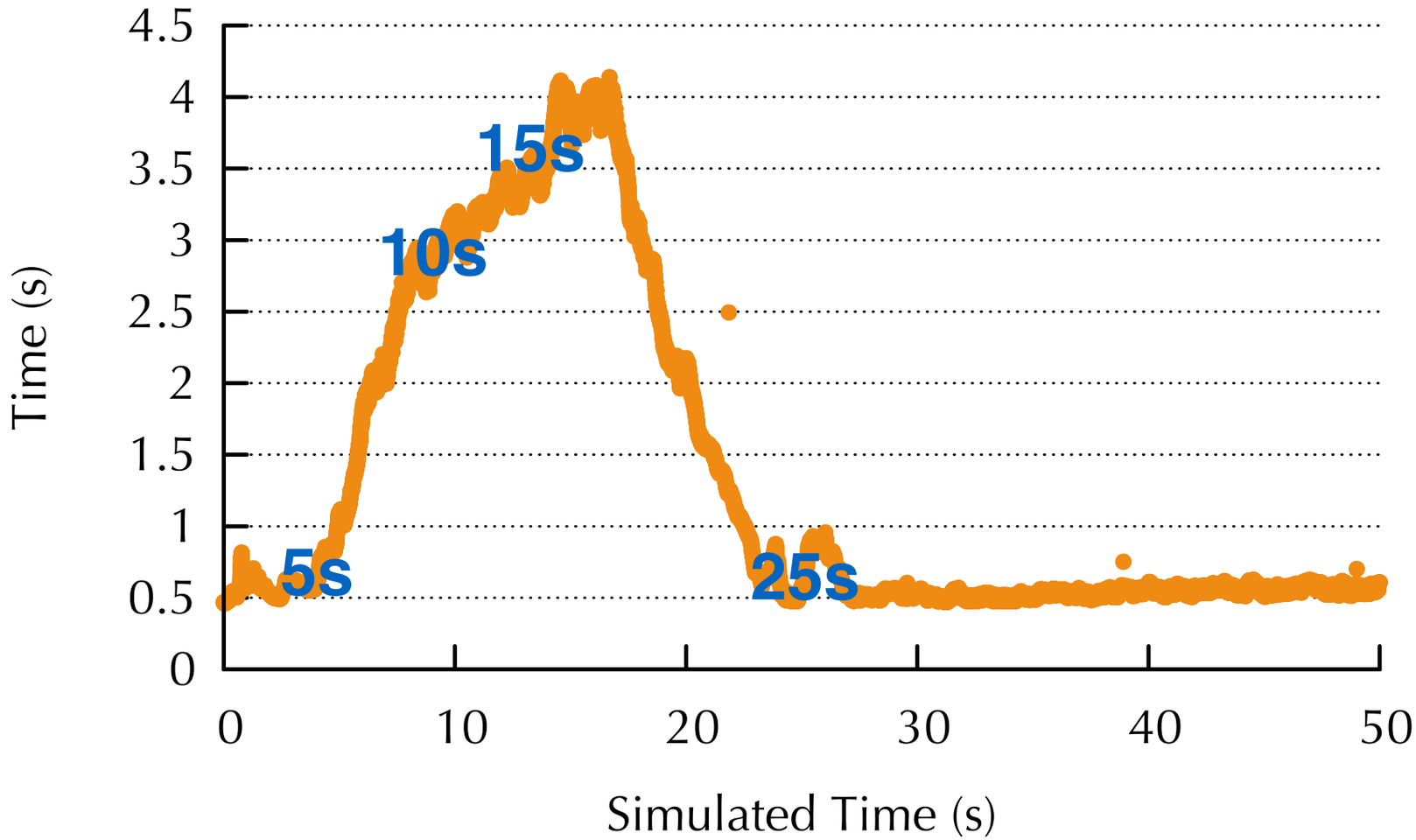
Long Twister

Long Twister

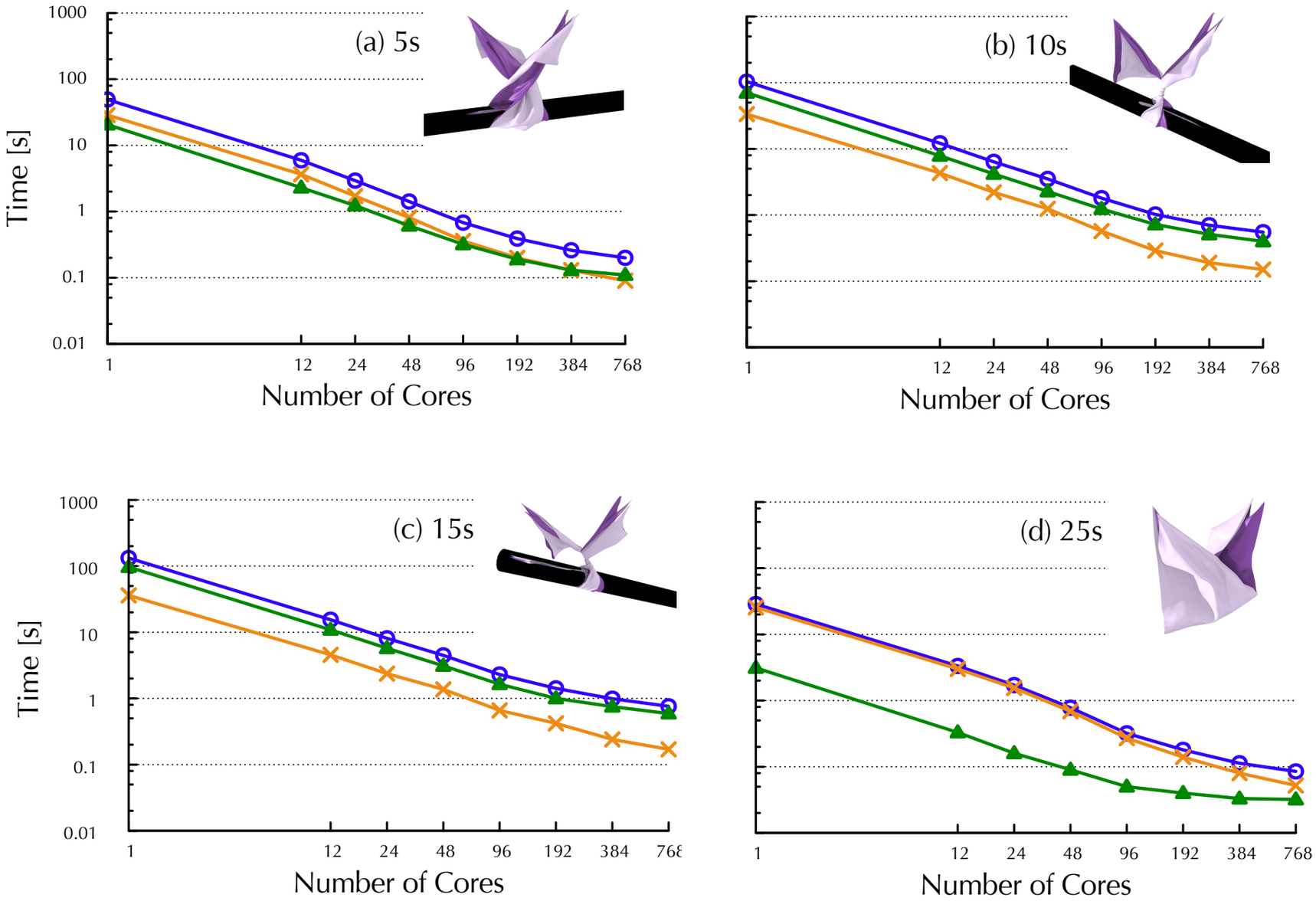
Long Twister



Long Twister



Long Twister



Conclusion

- Strong scaling to 384 cores on Edison
- More than 10x speedup compared to the TBB shared memory results
- Charm++ is well-suited for dynamic irregular applications like ACM
- Message-driven feature helps the overlapping of communication and computation