Speculative Load Balancing

Hassan Eslami
William D. Gropp

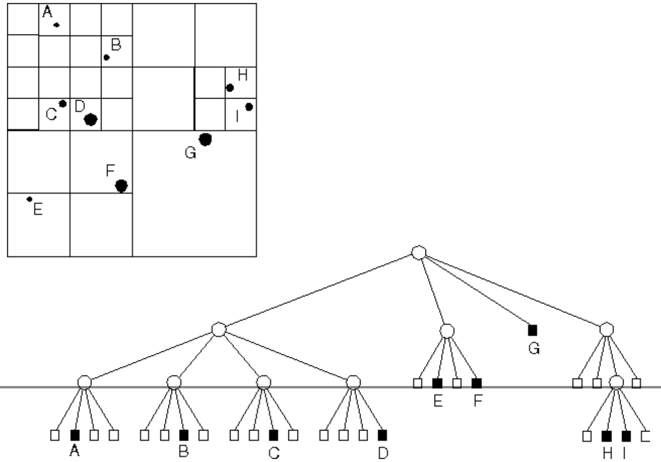
Department of Computer Science
University of Illinois at Urbana Champaign

Continuous Dynamic Load Balancing

- Irregular parallel applications
  - Irregular and unpredictable structure
  - Nested or recursive parallelism
  - Dynamic generation of units of computation
  - Available parallelism heavily depends on input data
  - Require continuous dynamic load balancing

Optimization and search problems

N-Body problems



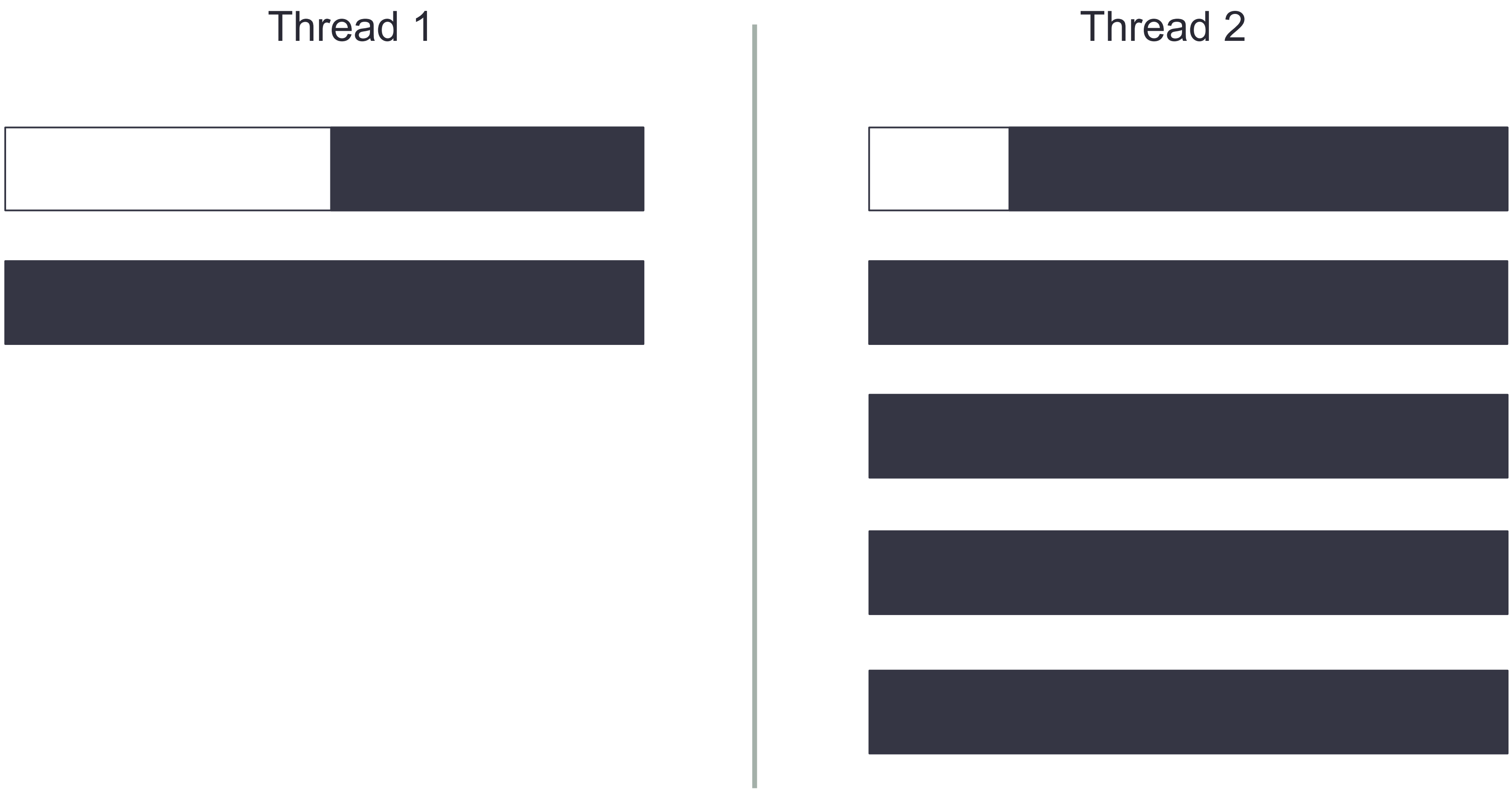
Dynamic Load Balancing Model

```
TaskPool.initialize(initial tasks)
While (t ← TaskPool.get())
      t.execute()
```

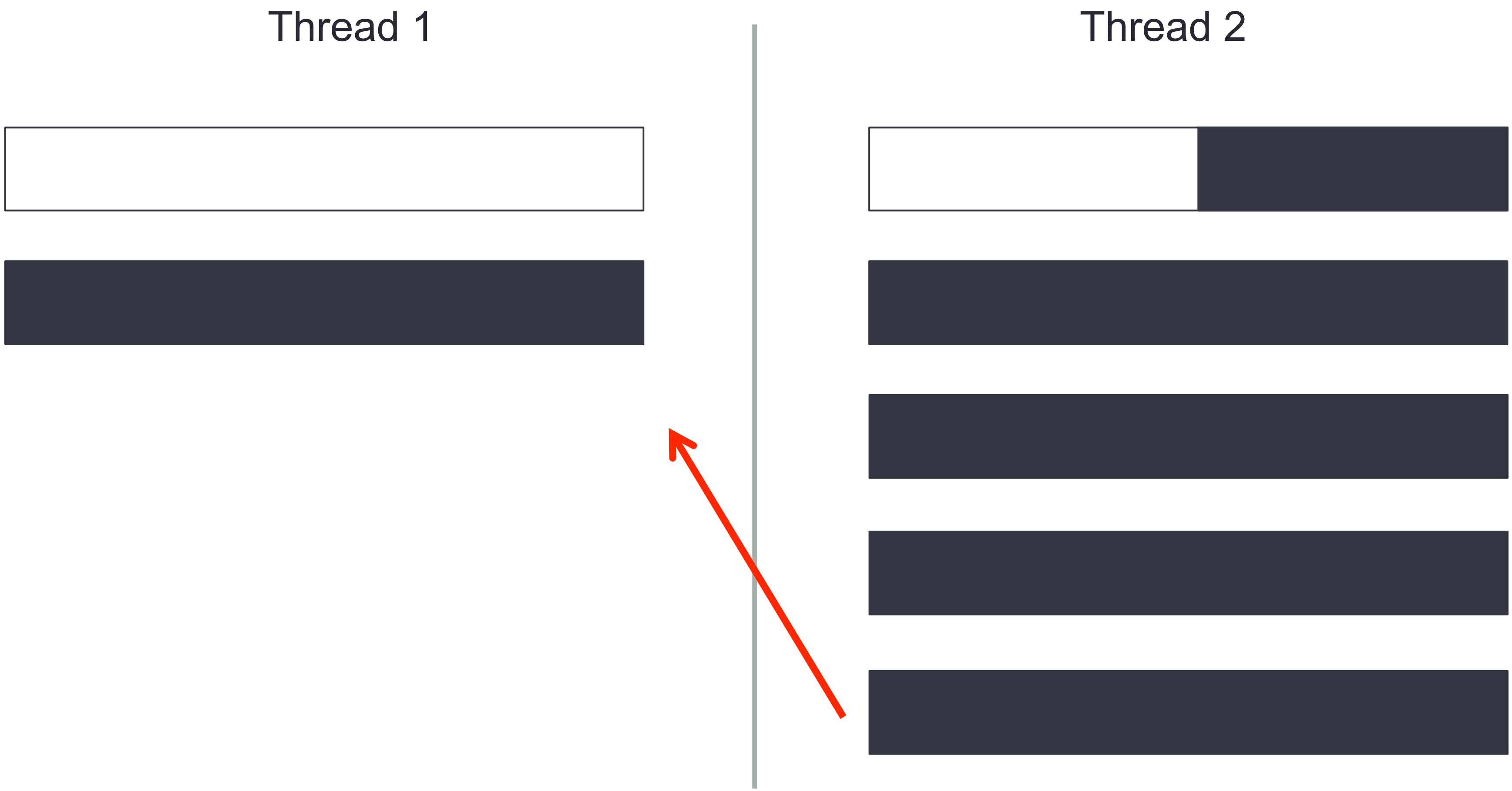
In `execute()`, one may call `TaskPool.put()`

Idle time in `TaskPool.get()`

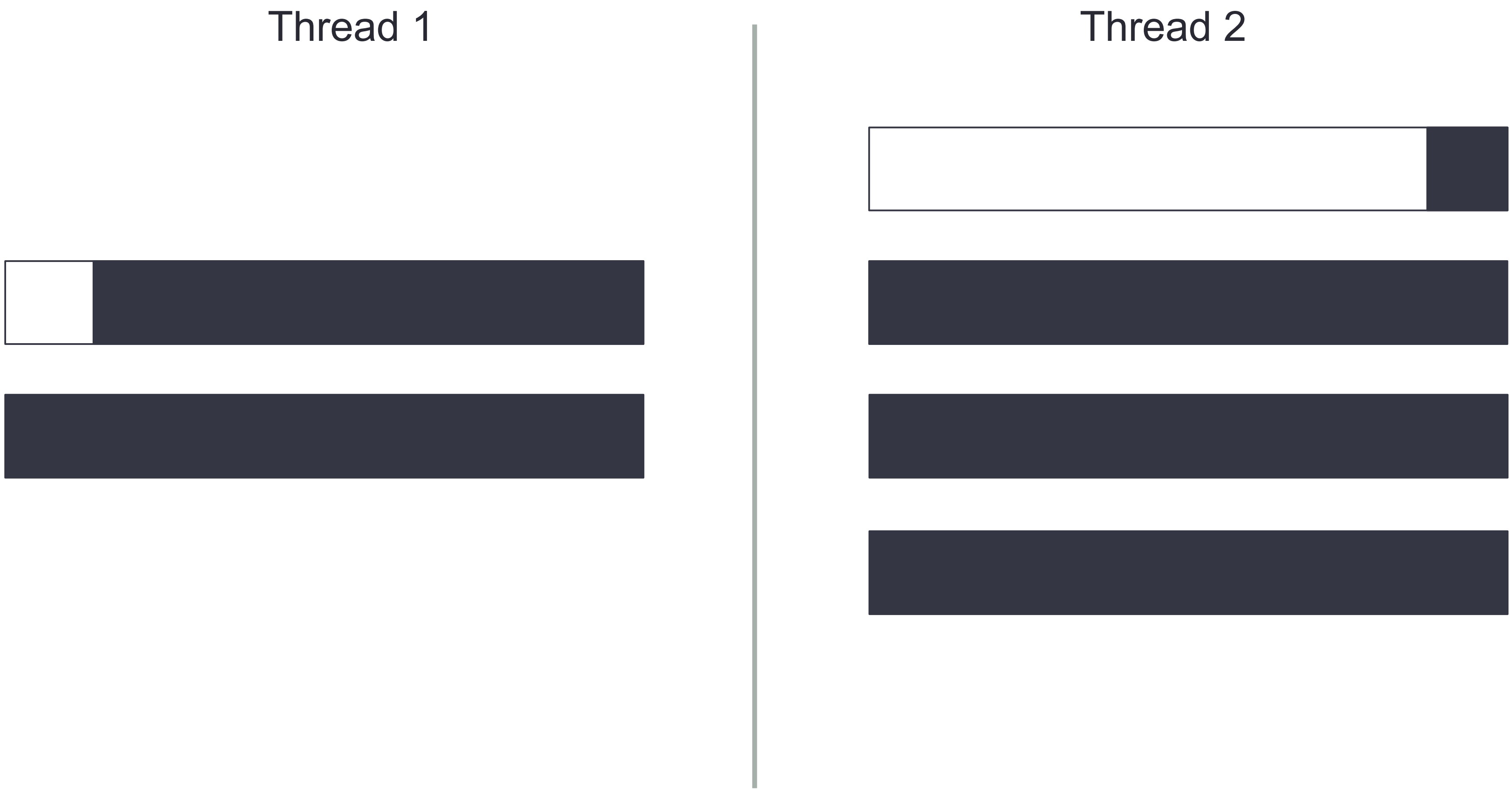
How to Eliminate Idle Time? – Prefetching



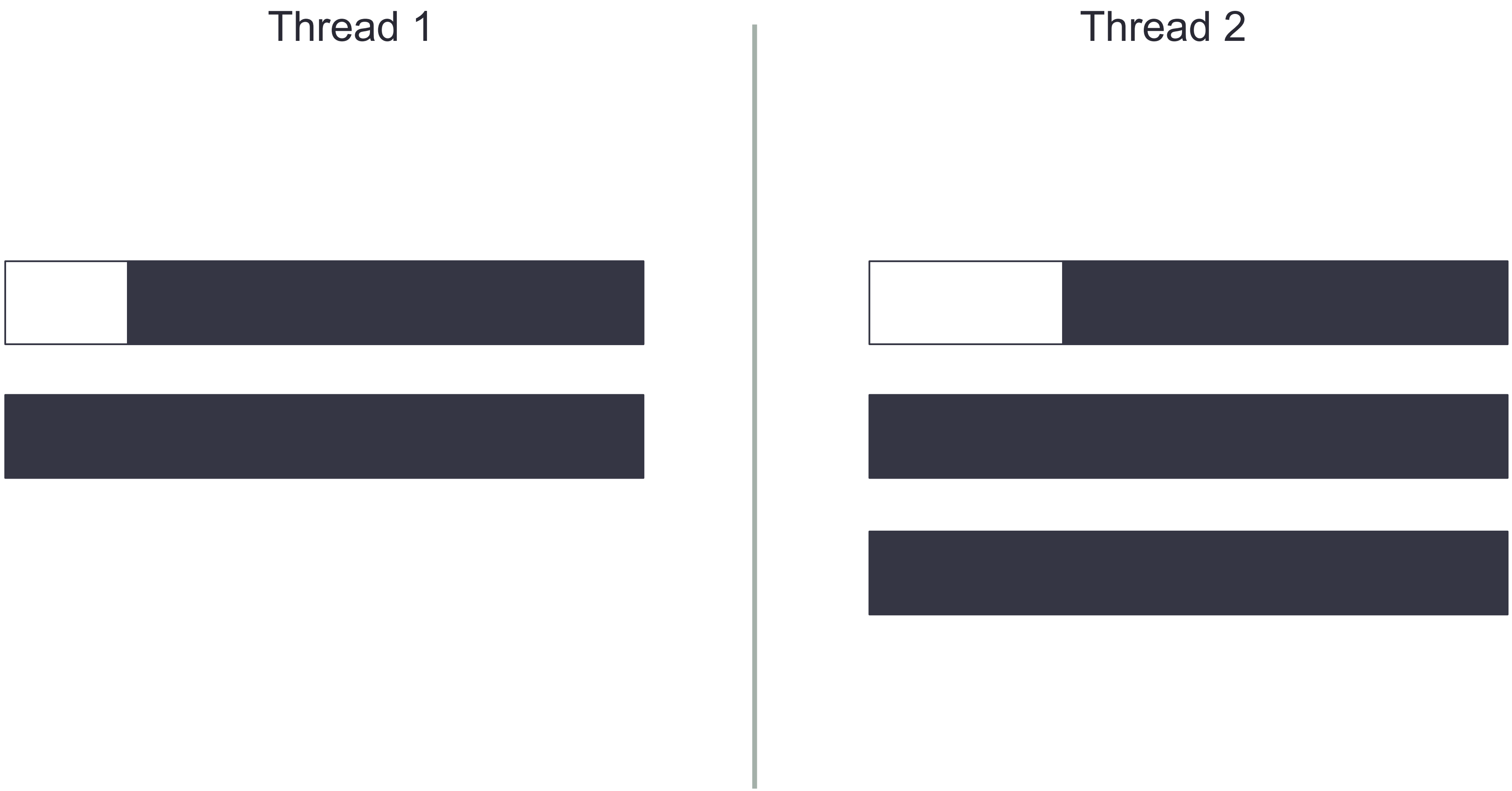
How to Eliminate Idle Time? – Prefetching



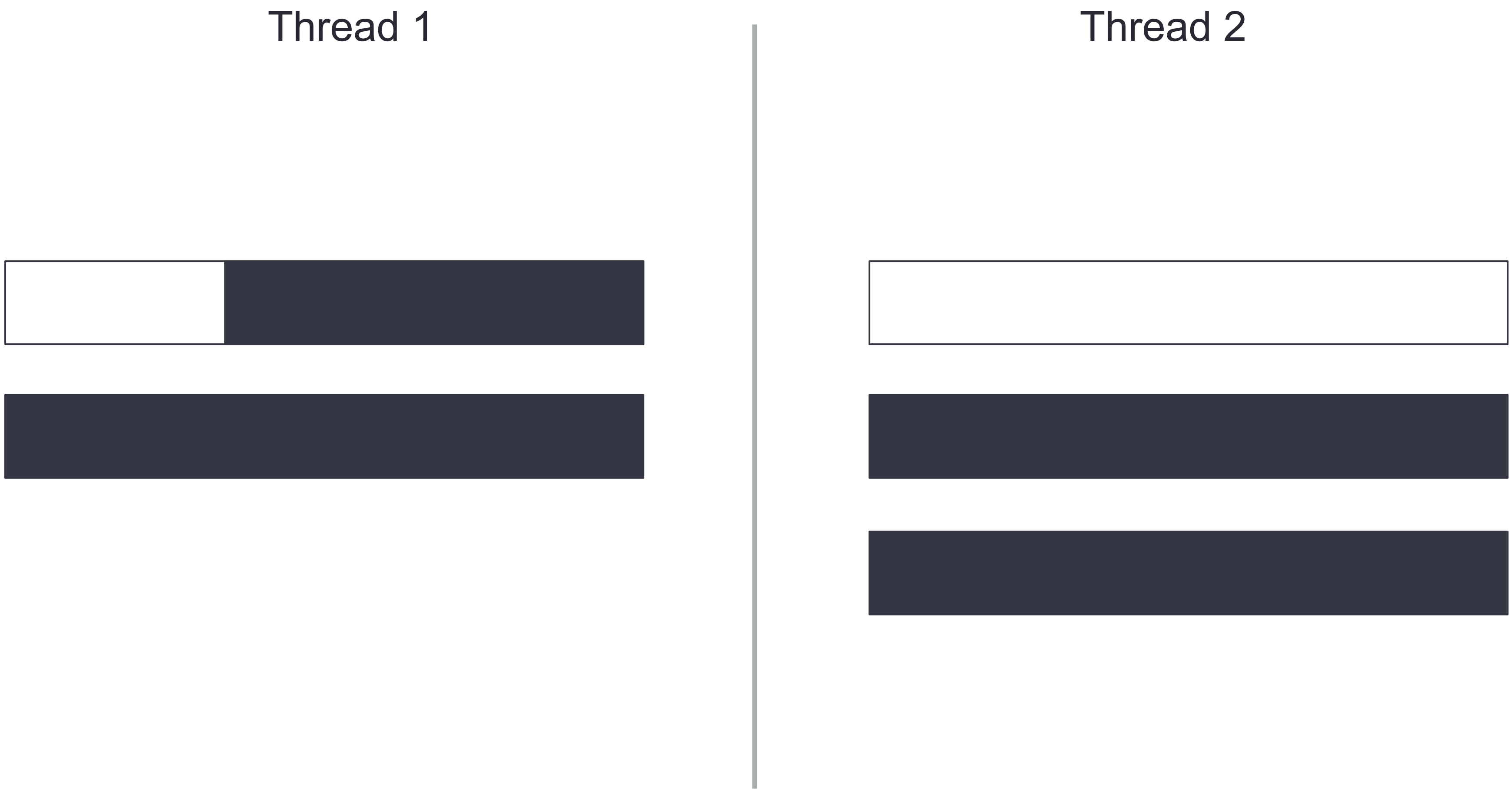
How to Eliminate Idle Time? – Prefetching



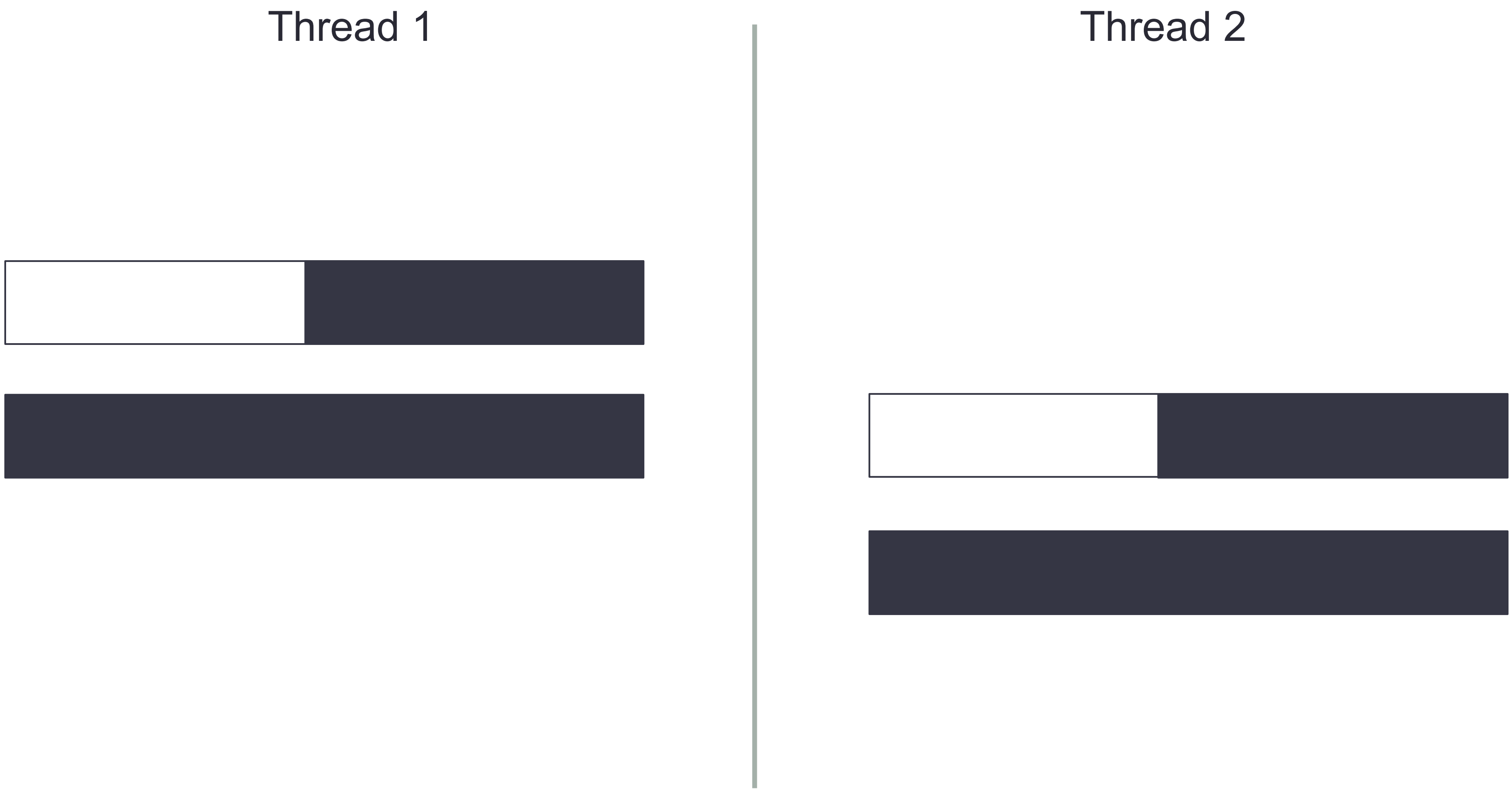
How to Eliminate Idle Time? – Prefetching



How to Eliminate Idle Time? – Prefetching



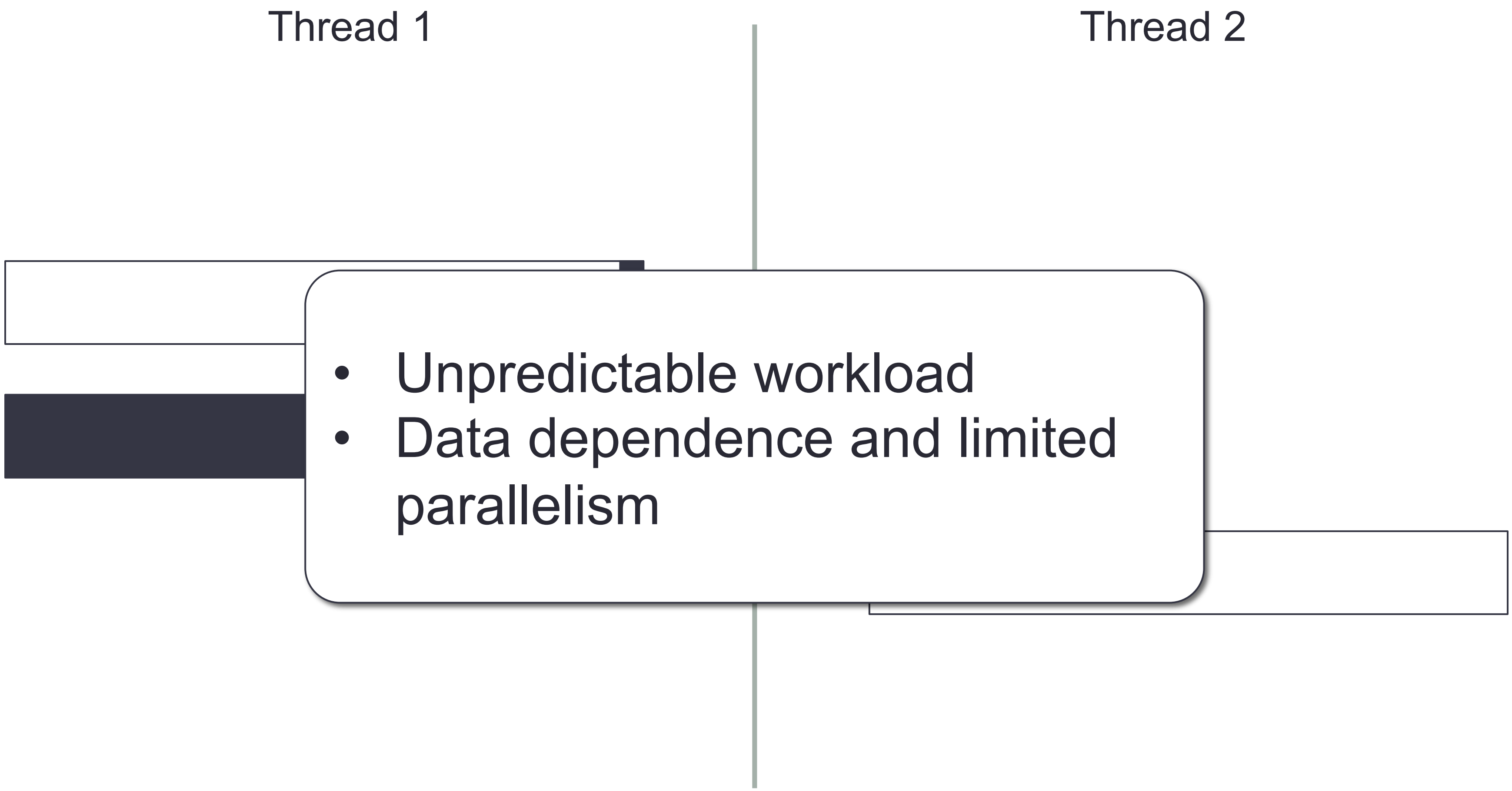
How to Eliminate Idle Time? – Prefetching



How to Eliminate Idle Time? – Prefetching

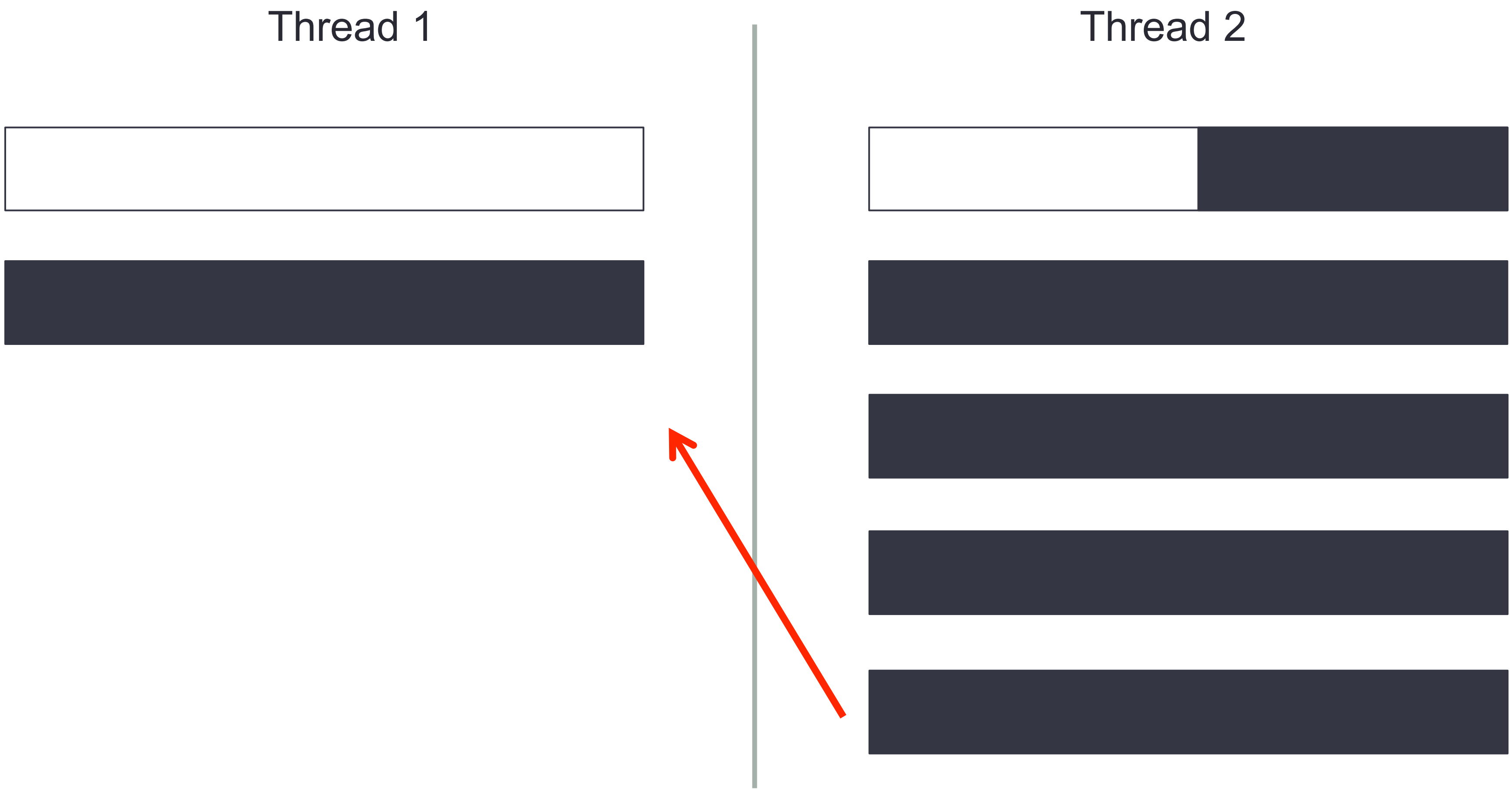
How to Eliminate Idle Time? – Prefetching

How to Eliminate Idle Time? – Prefetching



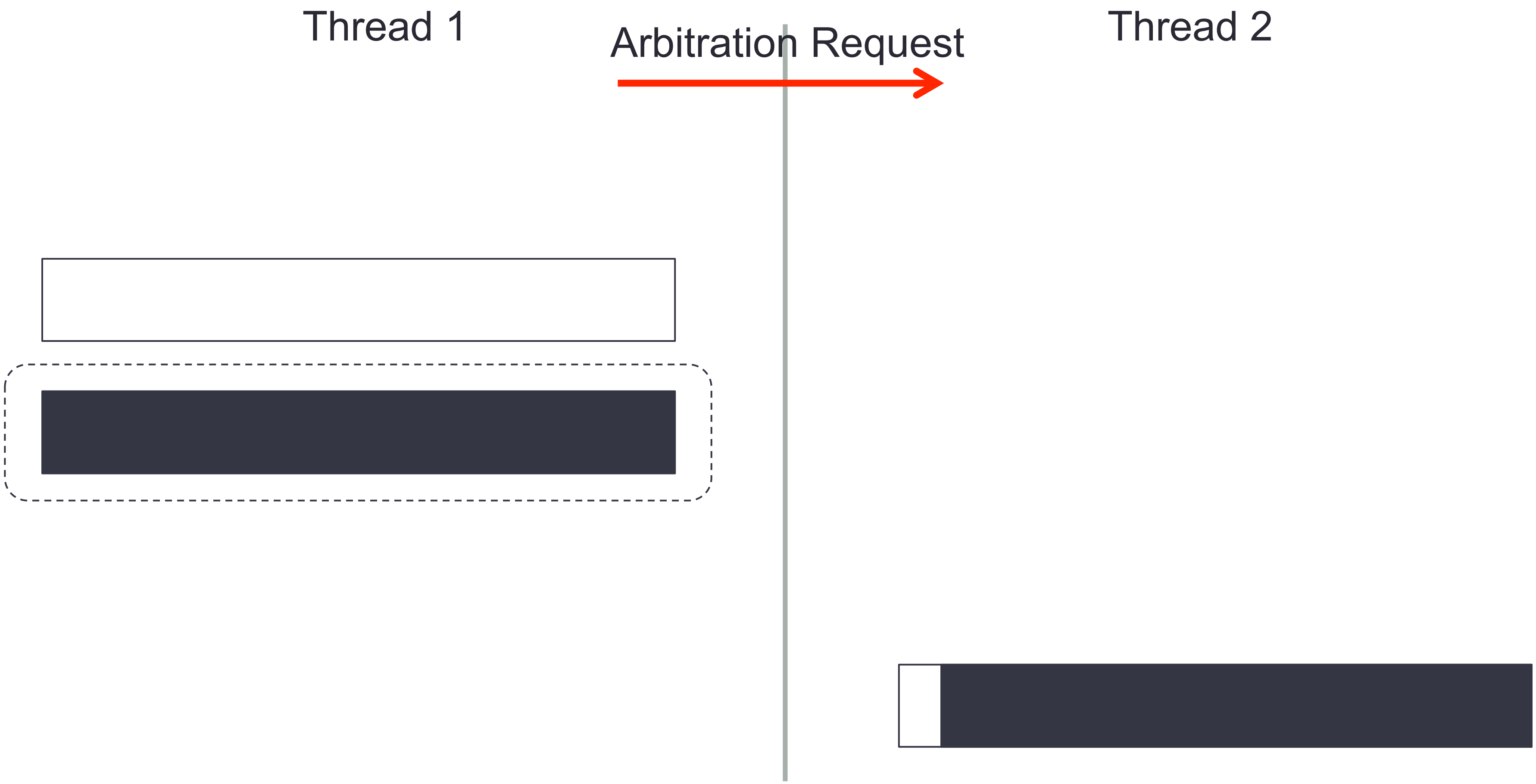
- Unpredictable workload
- Data dependence and limited parallelism

How to Eliminate Idle Time? – Speculation

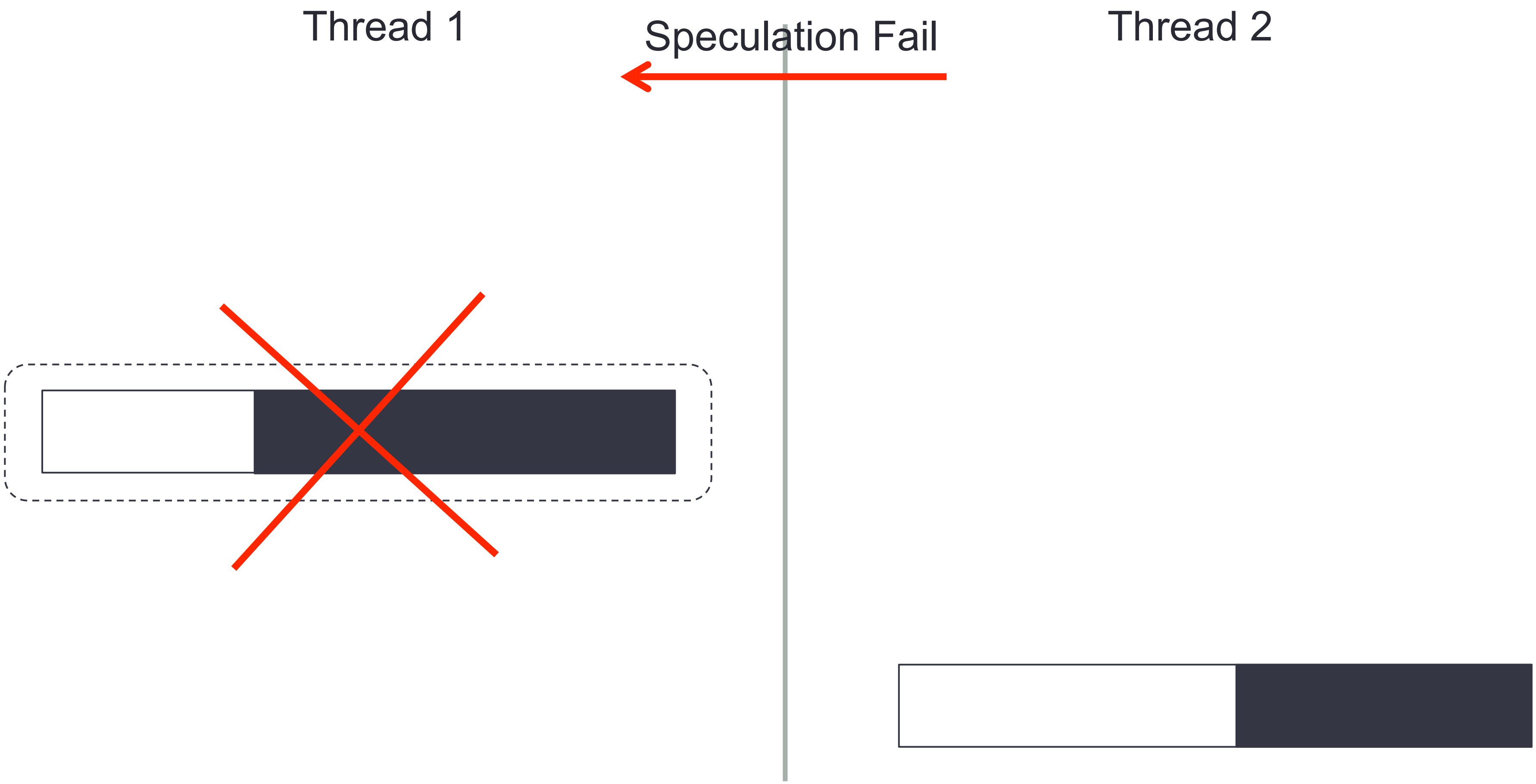


How to Eliminate Idle Time? – Speculation

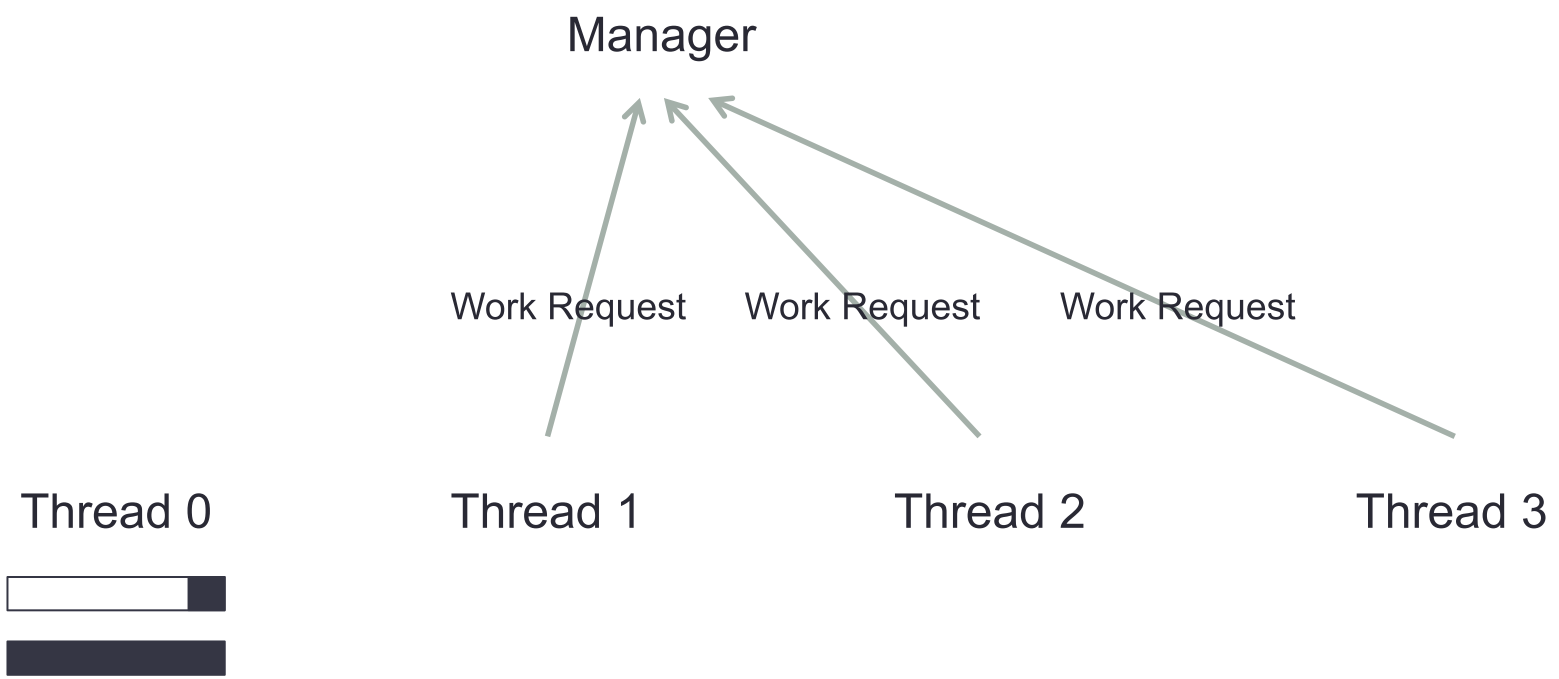
How to Eliminate Idle Time? – Speculation



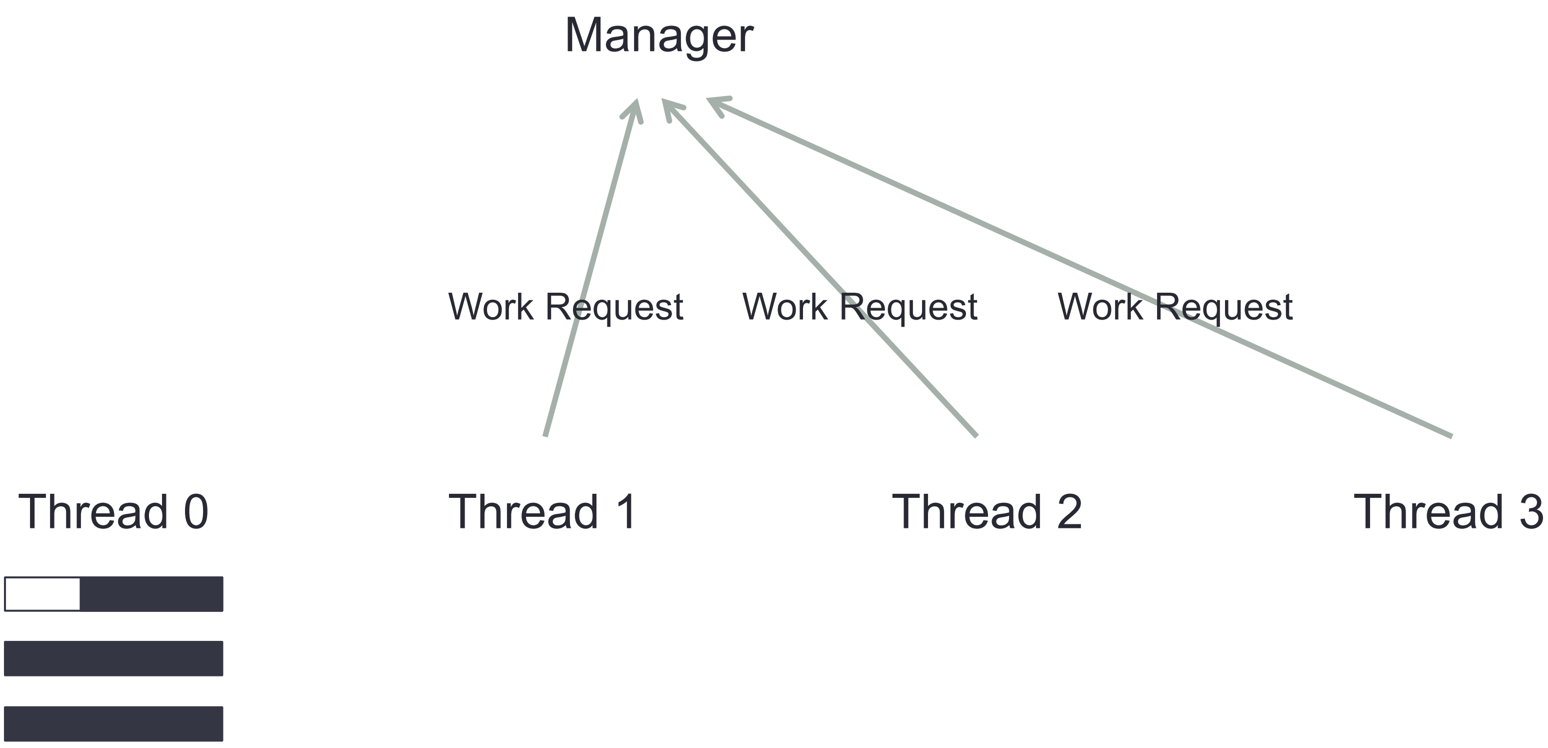
How to Eliminate Idle Time? – Speculation



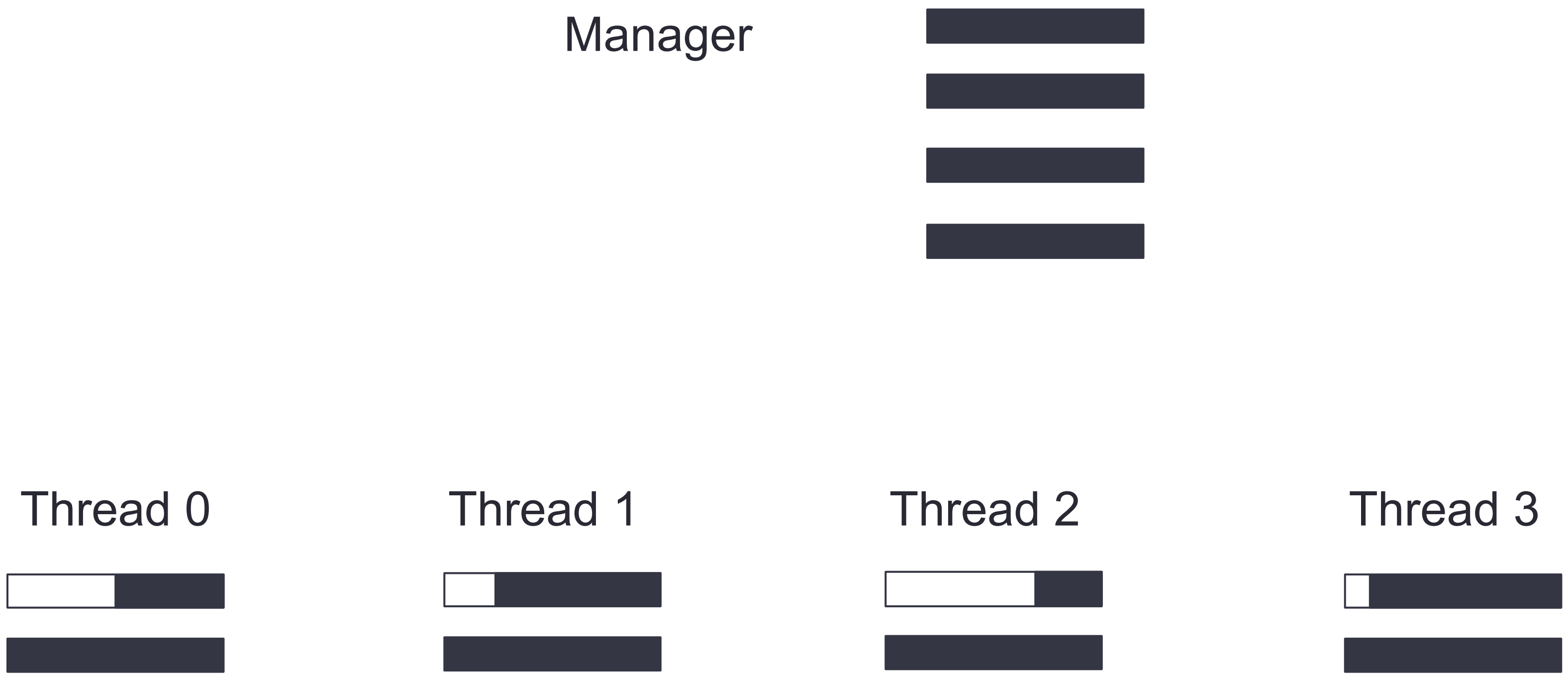
Work Sharing Algorithm



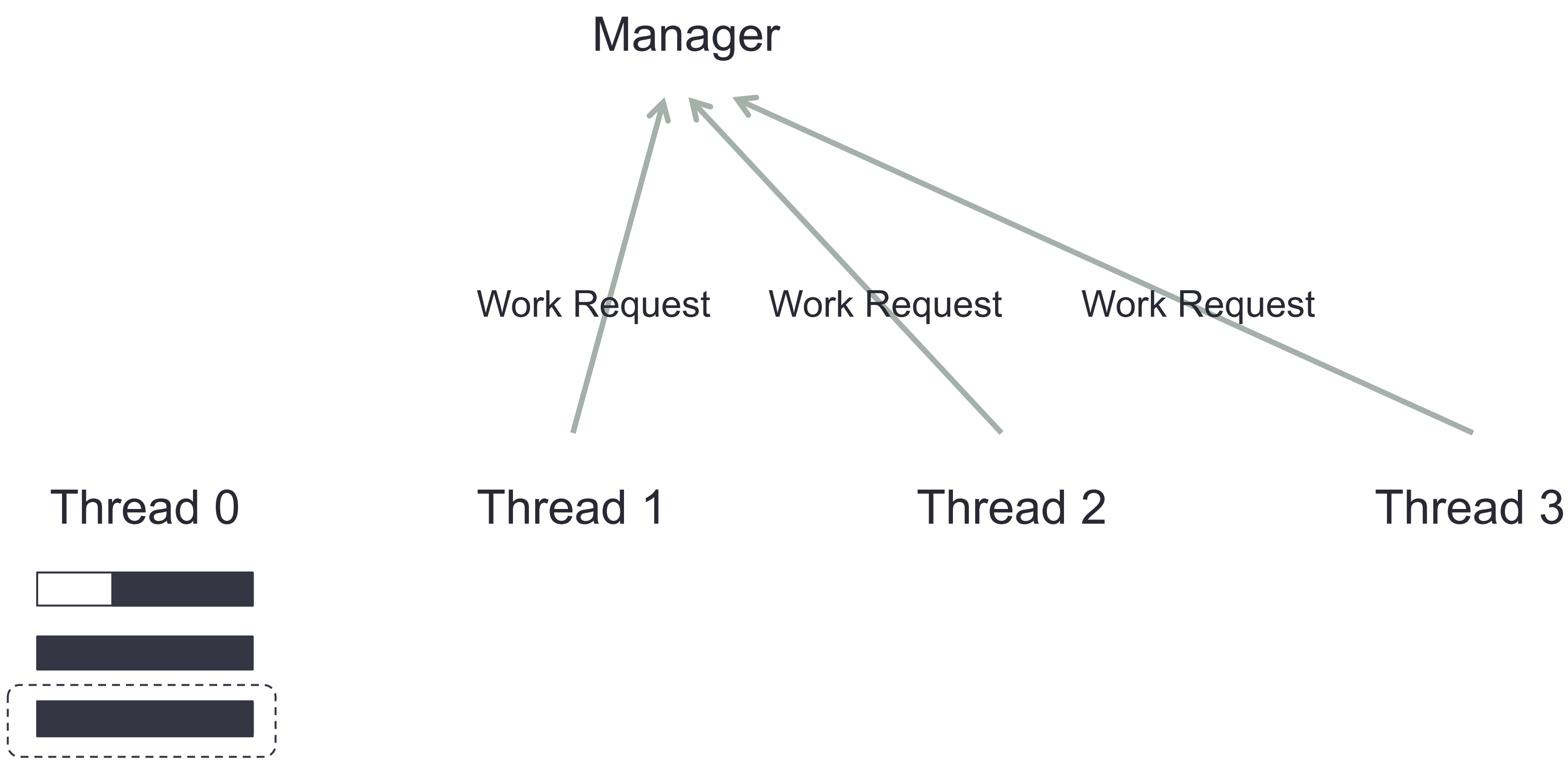
Work Sharing Algorithm



Work Sharing Algorithm



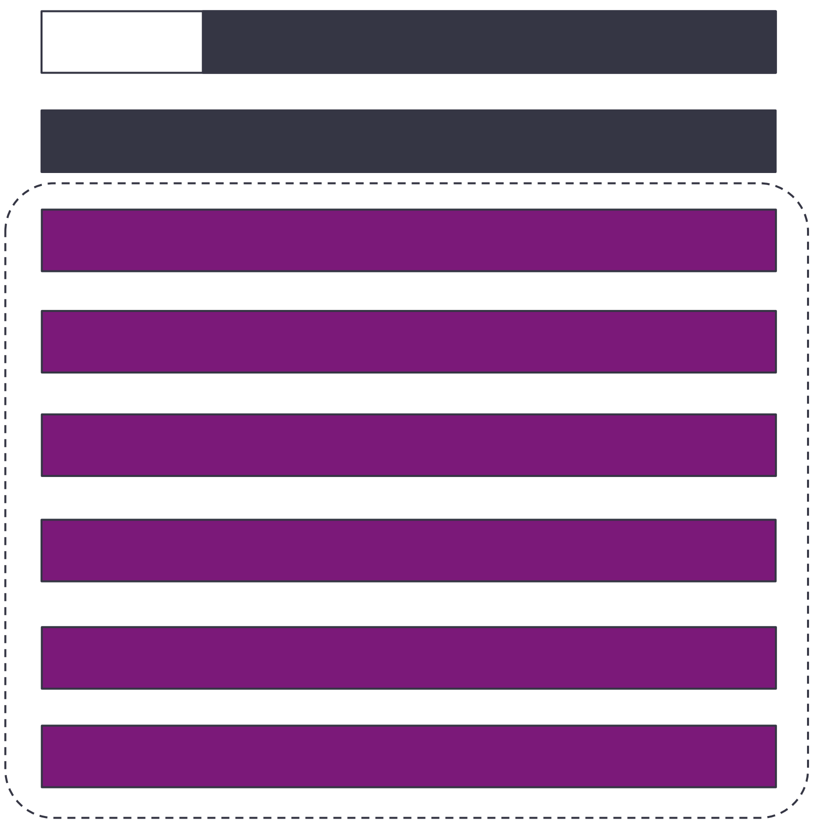
Speculative Work Sharing Algorithm



Speculative Work Sharing Algorithm

Some Worker Thread

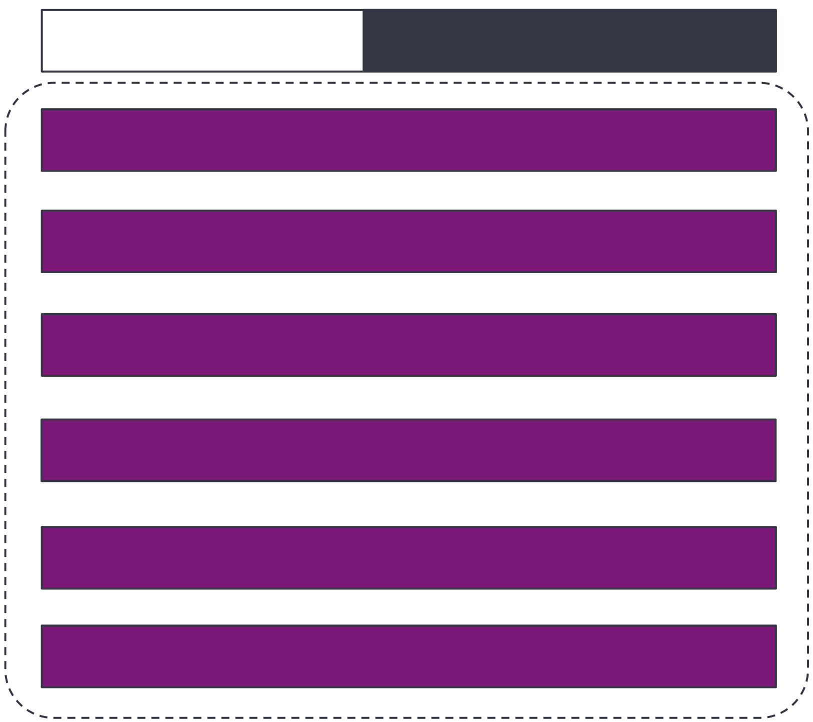
Manager Thread



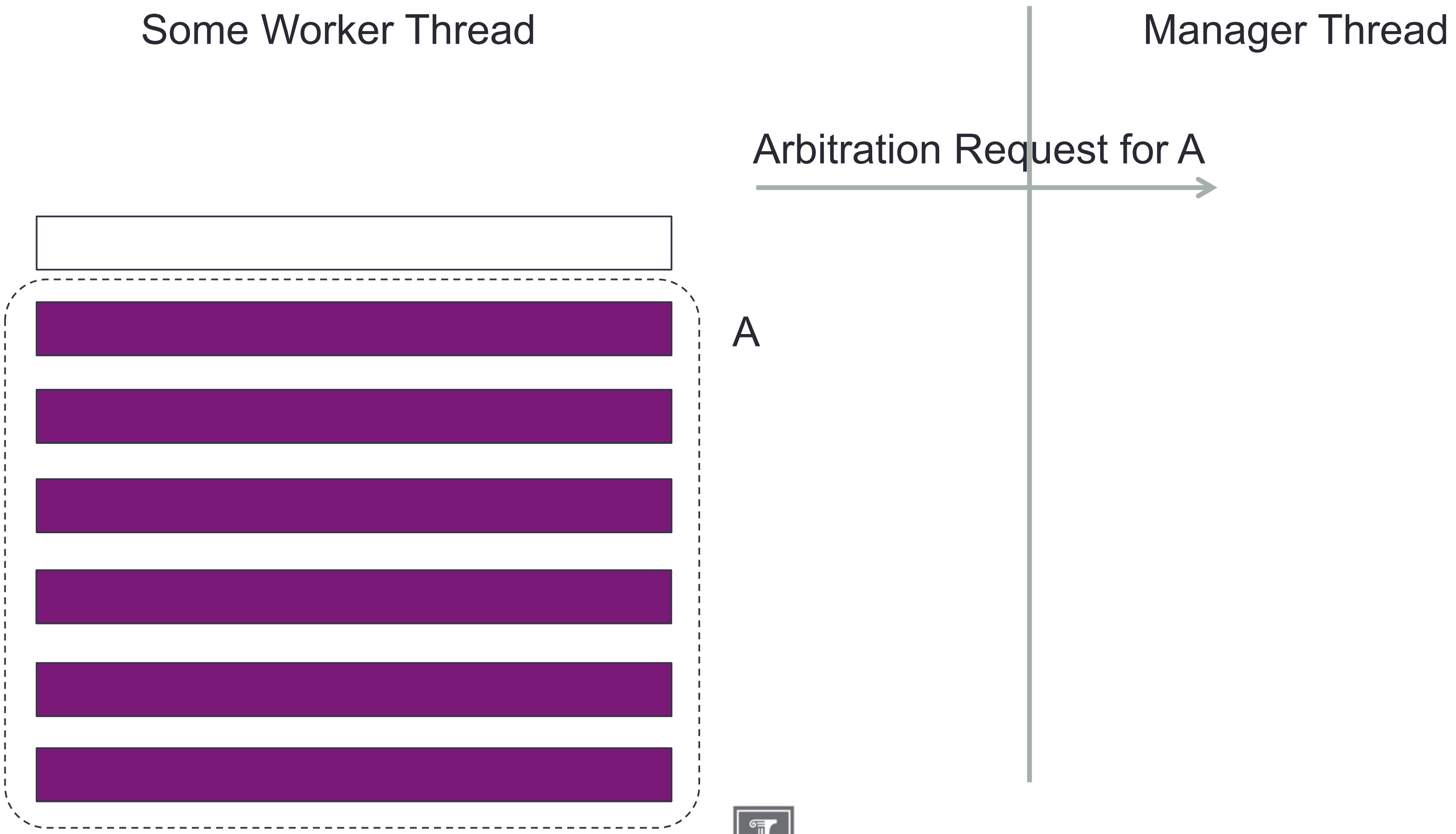
Speculative Work Sharing Algorithm

Some Worker Thread

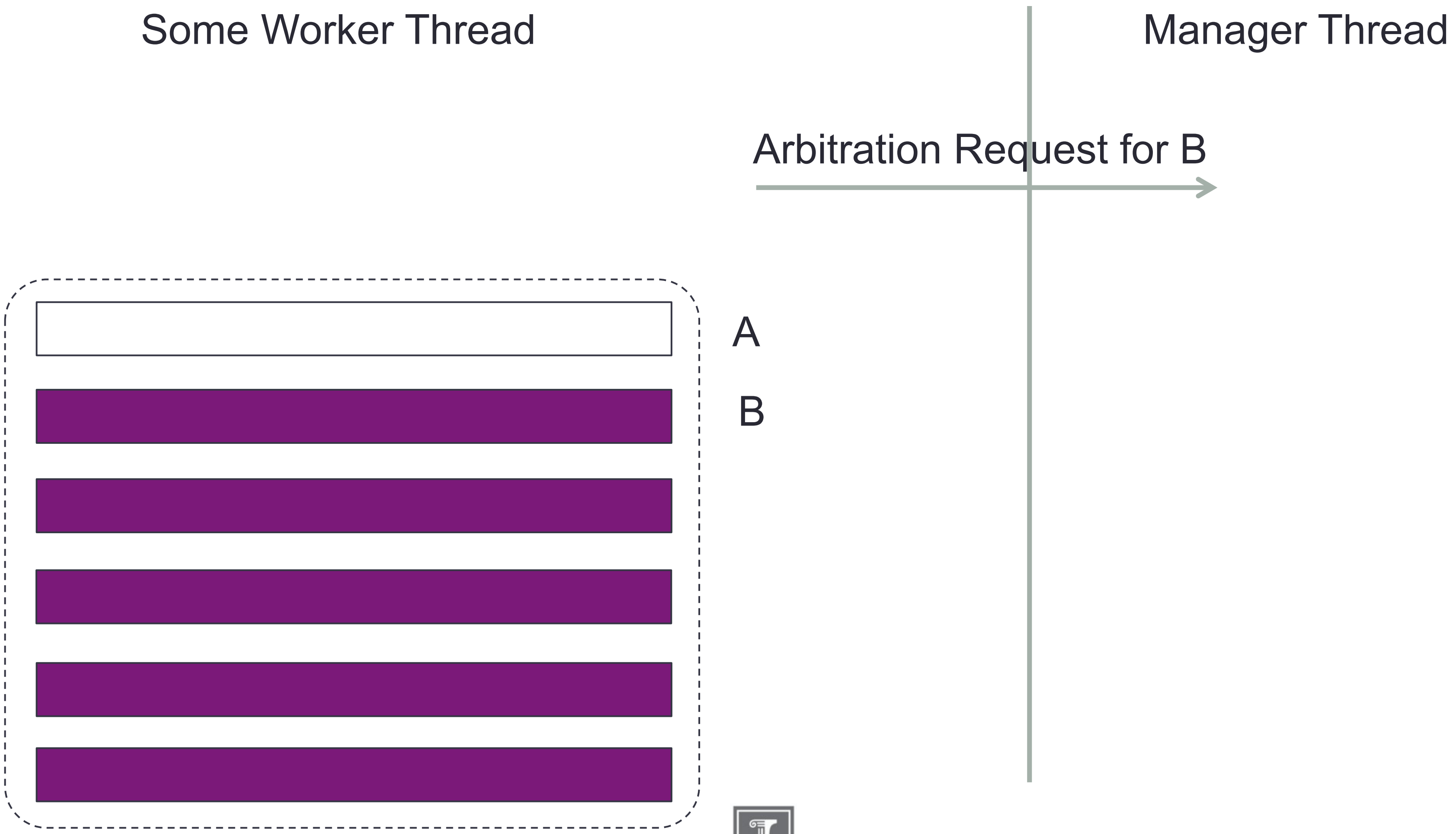
Manager Thread



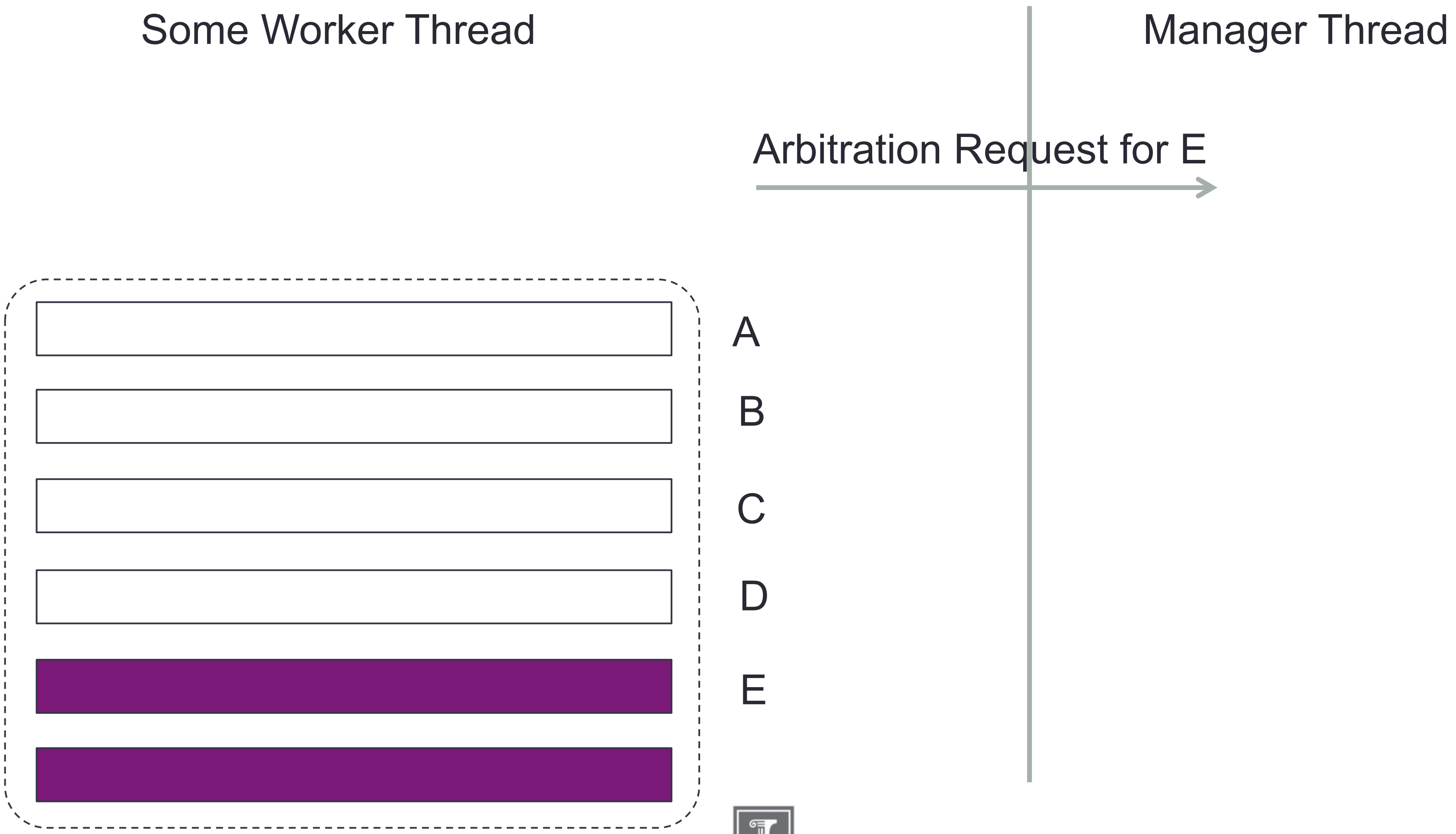
Speculative Work Sharing Algorithm



Speculative Work Sharing Algorithm



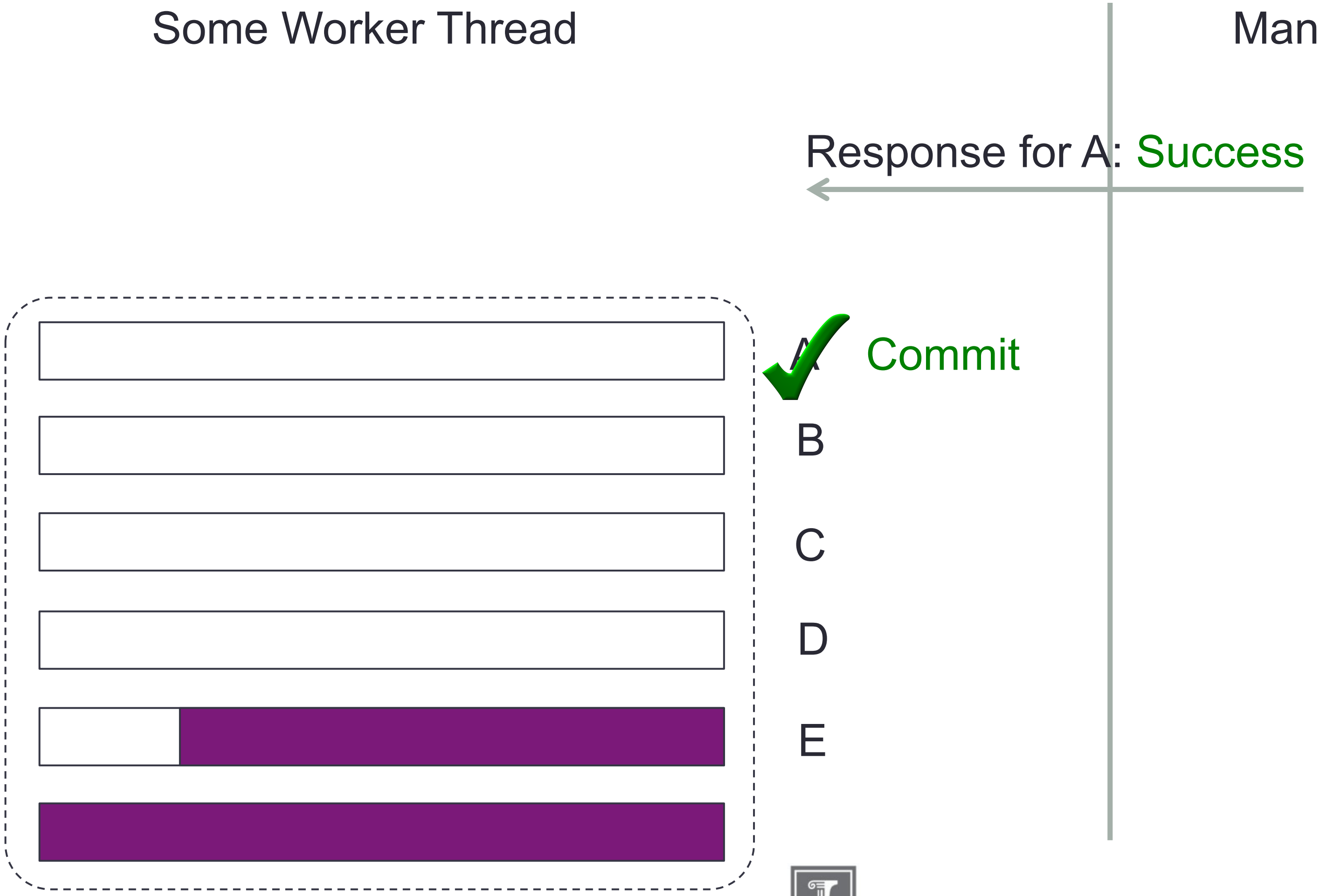
Speculative Work Sharing Algorithm



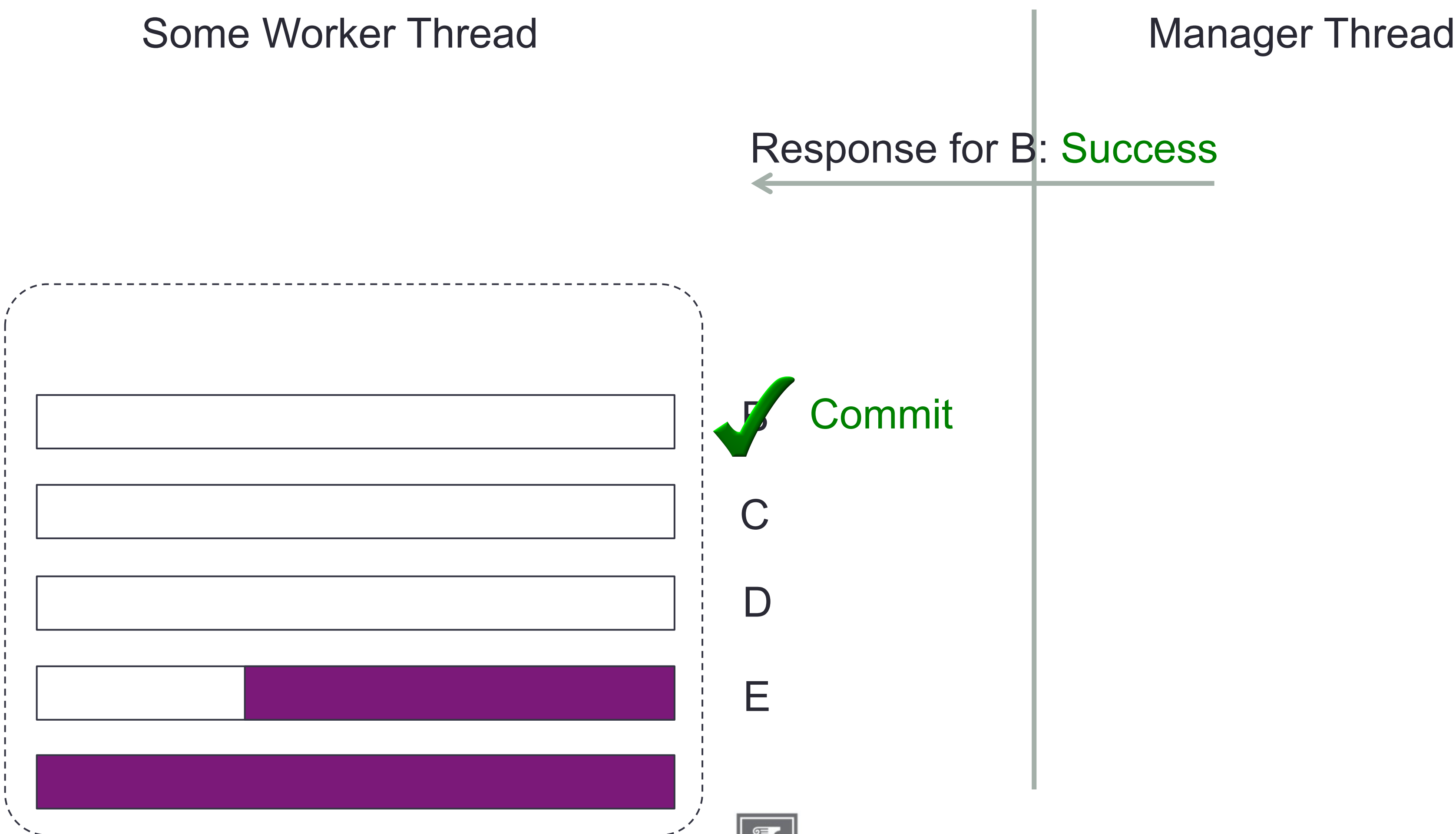
Speculative Work Sharing Algorithm

Some Worker Thread

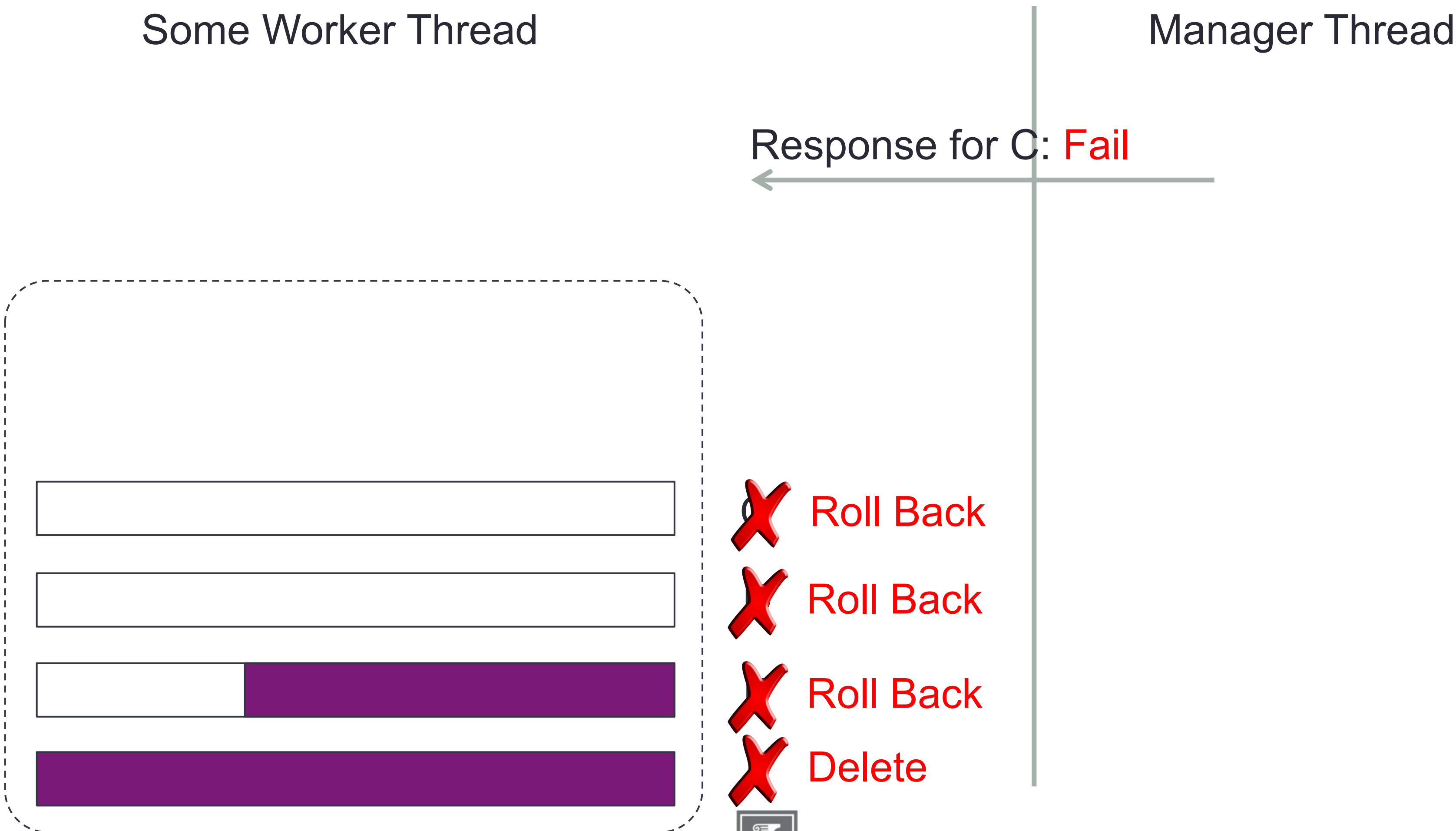
Manager Thread



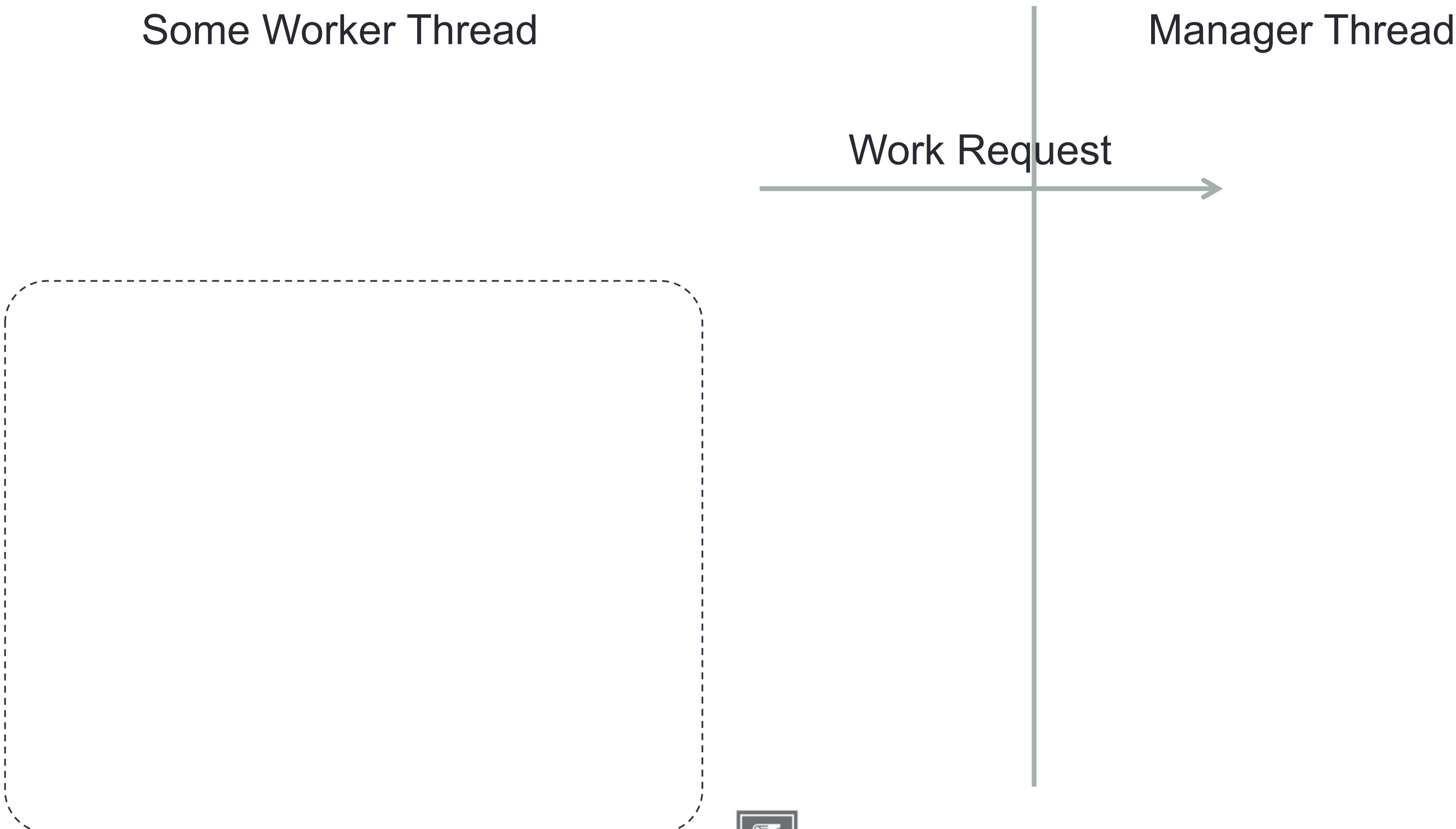
Speculative Work Sharing Algorithm



Speculative Work Sharing Algorithm



Speculative Work Sharing Algorithm



Unbalanced Tree Search (UTS)

- Counting nodes in randomly generated tree
- Tree generation is based on separable cryptographic random number generator

```
childCount = f(nodeId)
childId = SHA1(nodeId, childIndex)
```

- Different types of trees
  - Binomial (probability q, # of child m)
  - Geometric (depth limit d, branching factor is geometric distribution with mean b)

Work Sharing in UTS

- A *node* in tree is a unit of work
- A *chunk* is a set of nodes, and minimum transferrable unit
- *Release interval* is the frequency with which a worker releases a *chunk* to the manager

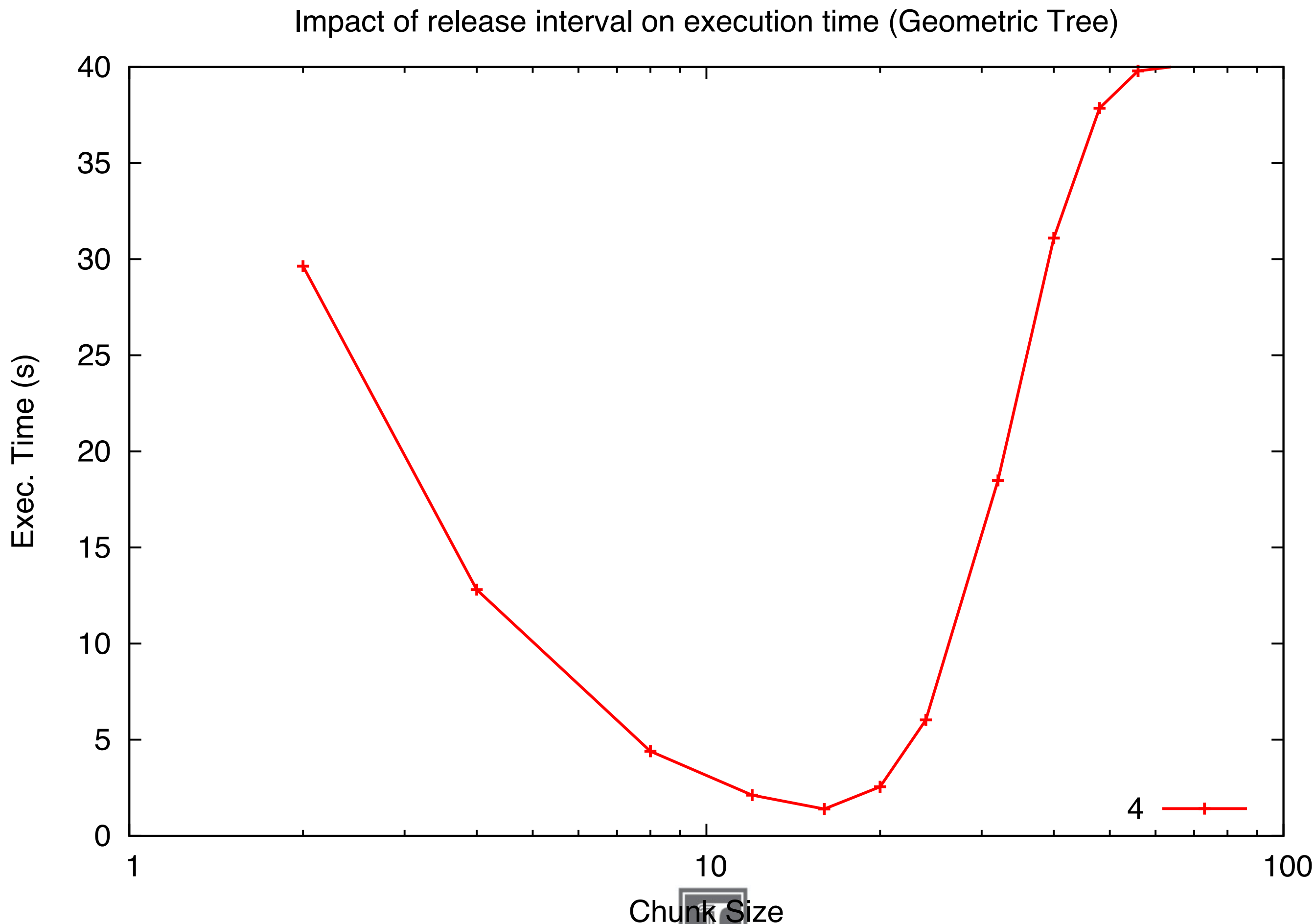
```
If (HasSurplusWork() and
    NodesProcessed % release_inerval == 0)
{
     ReleaseWork()
}
```

Experimental Setup and Inputs

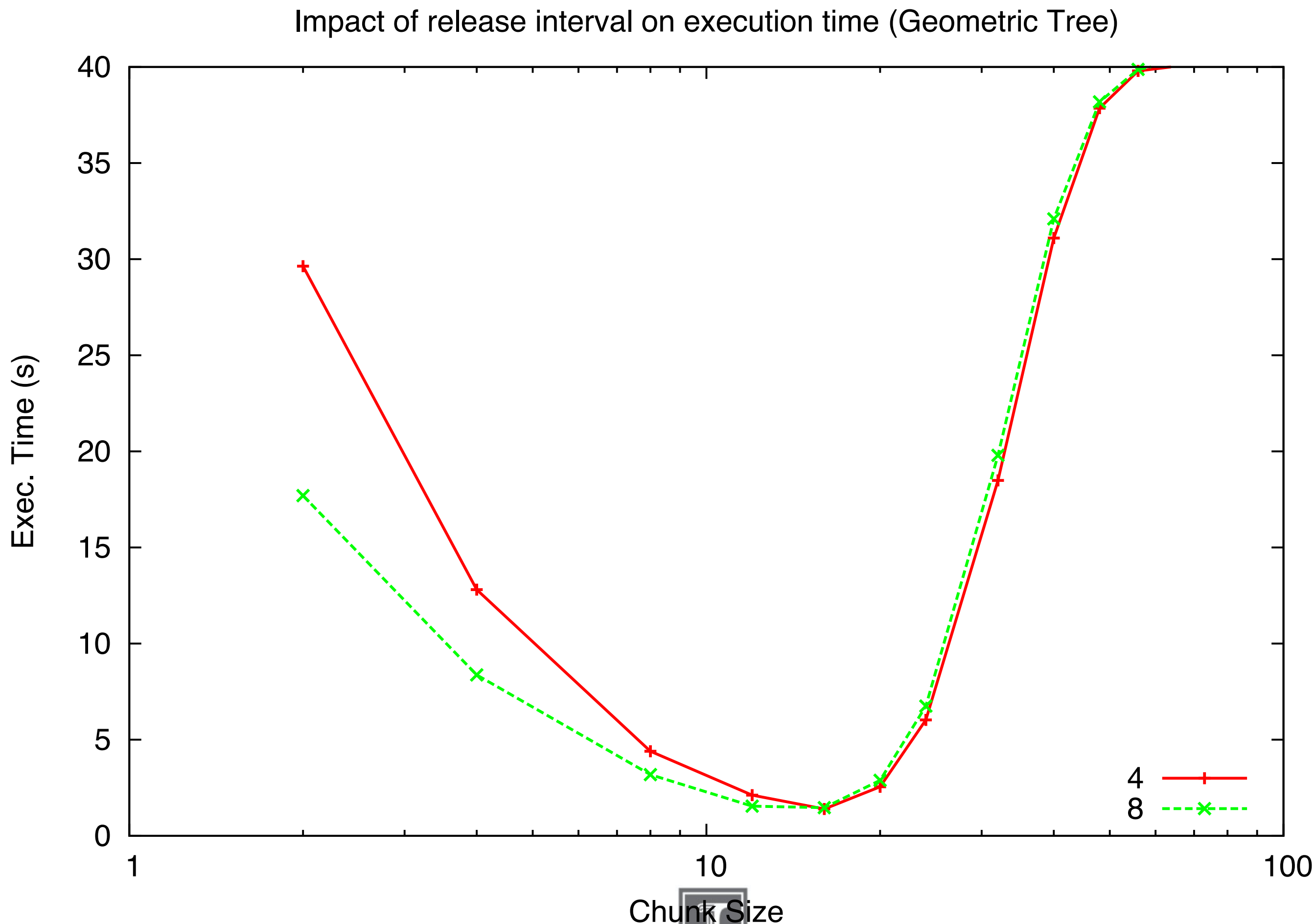
- Illinois Campus Cluster
  - Cluster of HP ProLiant Servers
  - 2 Intel X5650 2.66Ghz 6Core Processors per Node
  - High Speed Infiniband Cluster Interconnect

	Binomial (10^9 Nodes)	Geometry (10^9 Nodes)
Small	0.111	0.102
Medium	2.79	1.64
Large	10.6	4.23

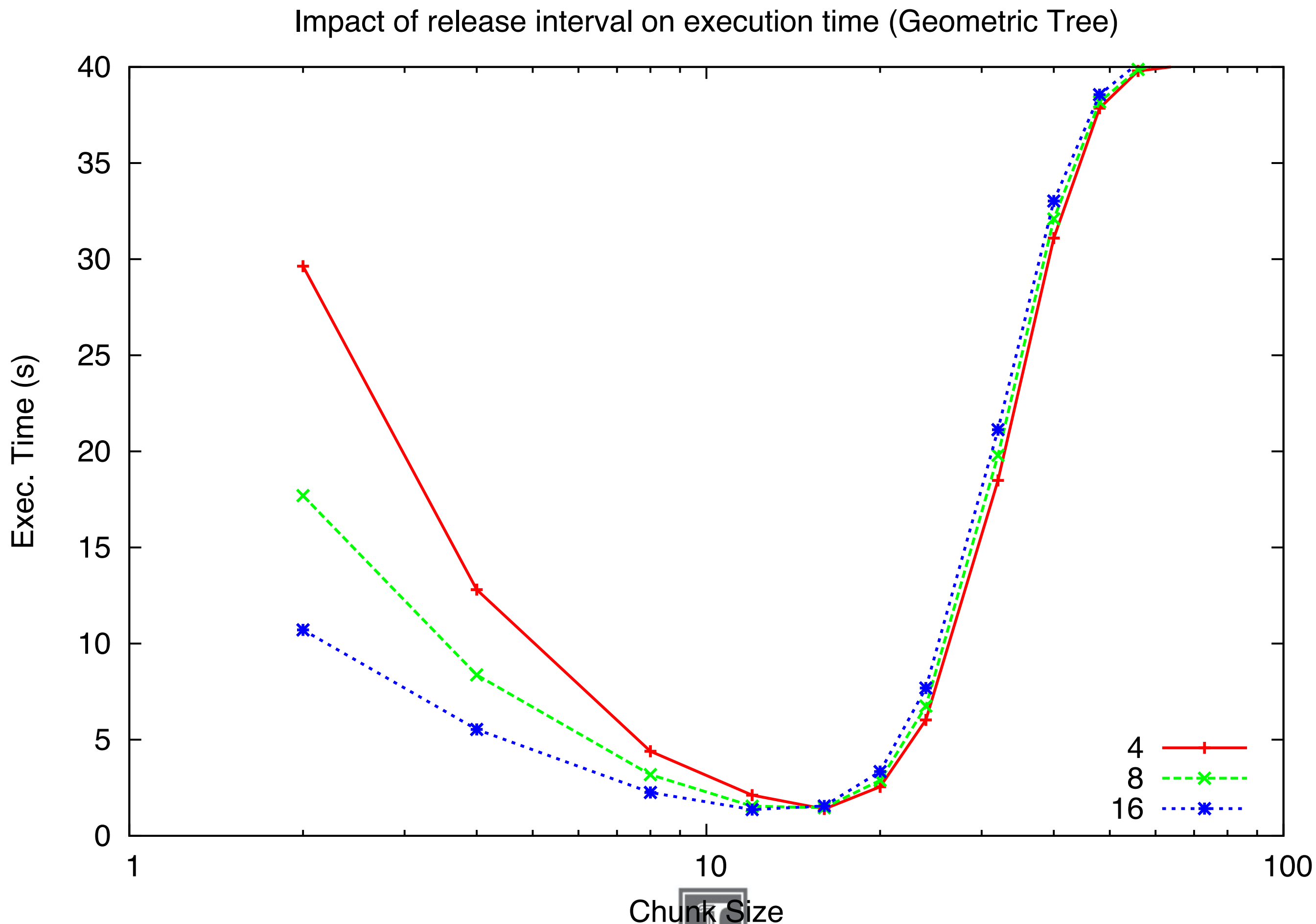
Tuning of Original Algorithm – Small Input (on 4 nodes, 12 cores each)



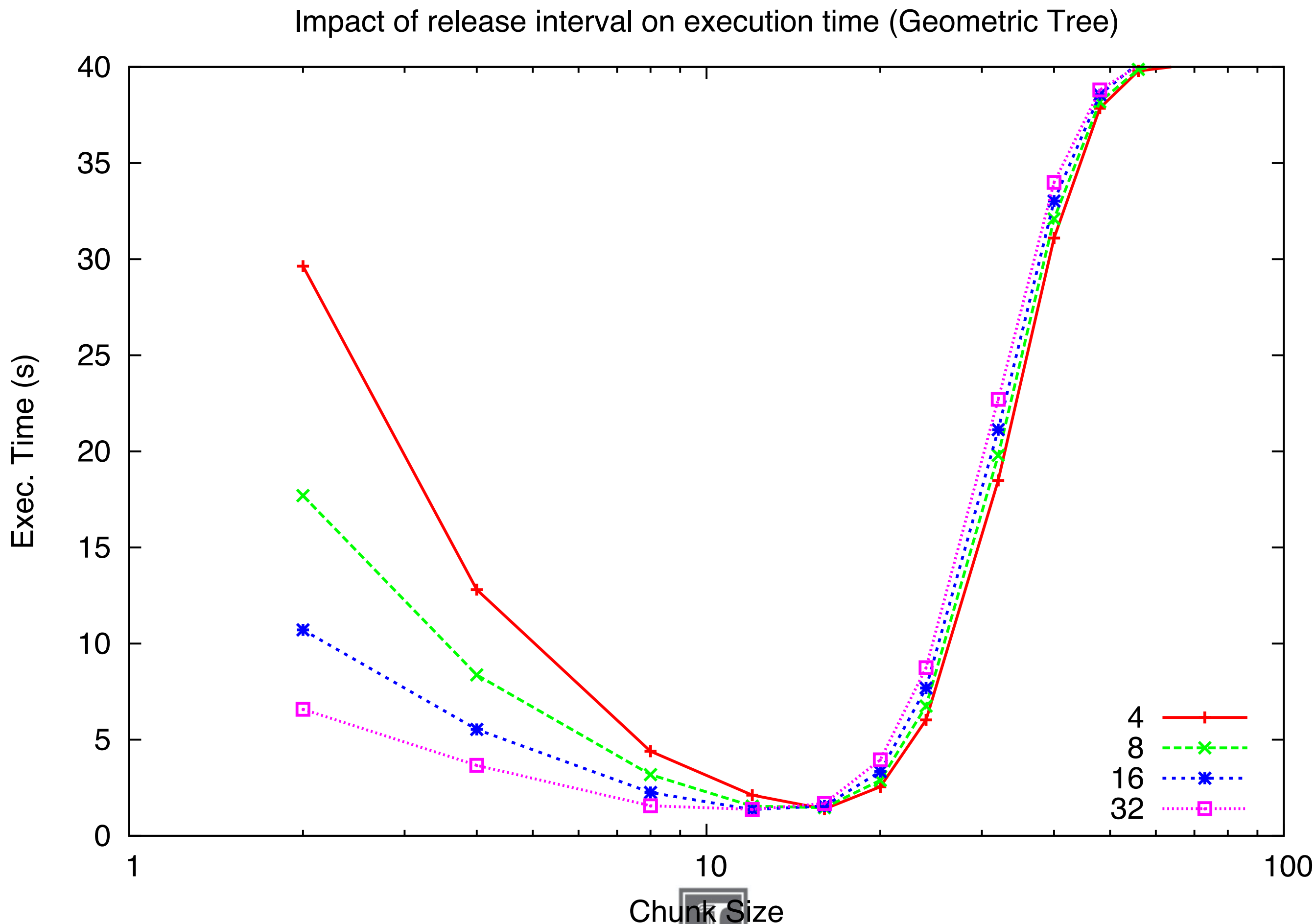
Tuning of Original Algorithm – Small Input (on 4 nodes, 12 cores each)



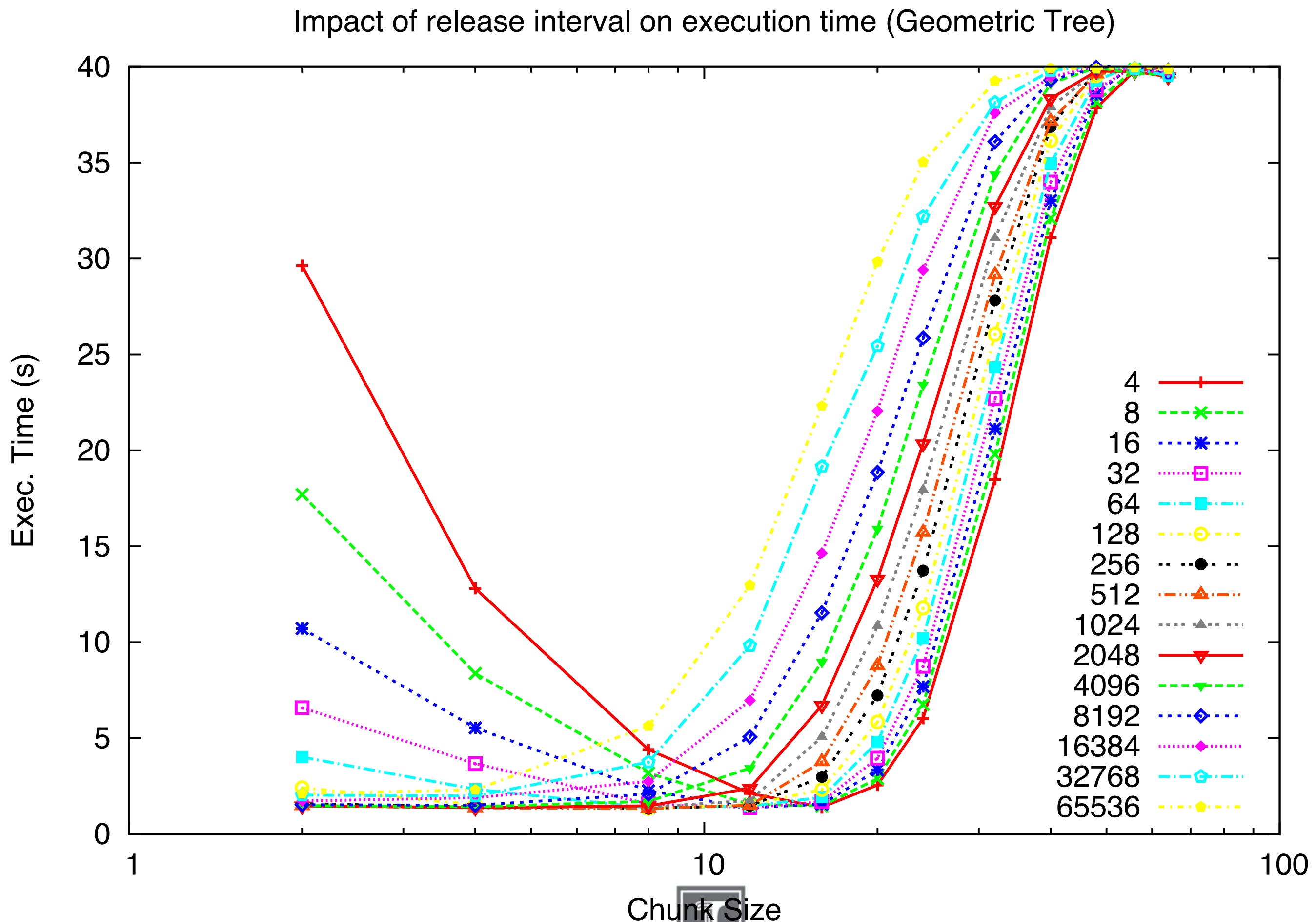
Tuning of Original Algorithm – Small Input (on 4 nodes, 12 cores each)



Tuning of Original Algorithm – Small Input (on 4 nodes, 12 cores each)

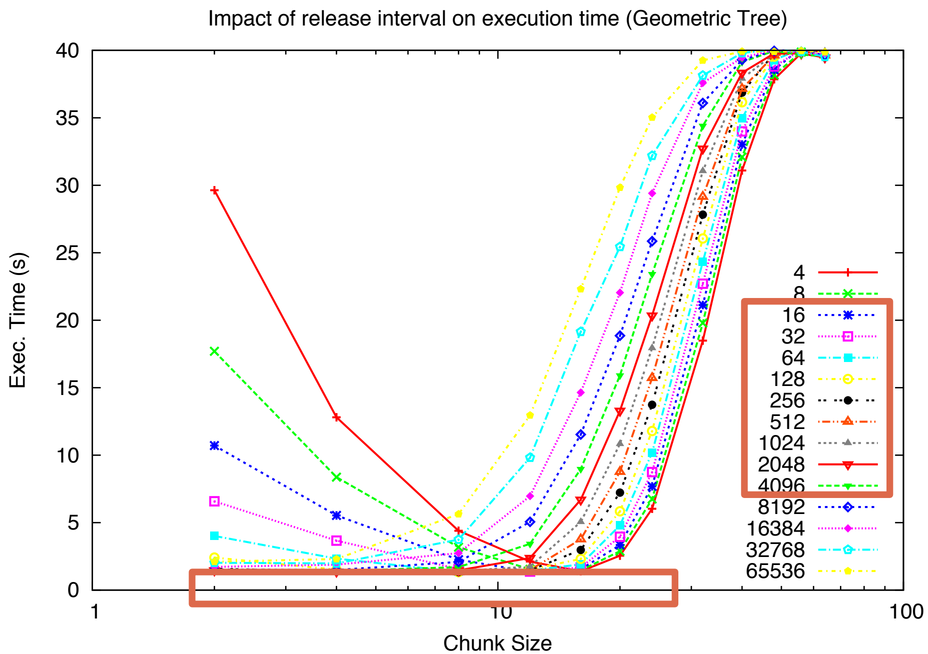


Tuning of Original Algorithm – Small Input (on 4 nodes, 12 cores each)

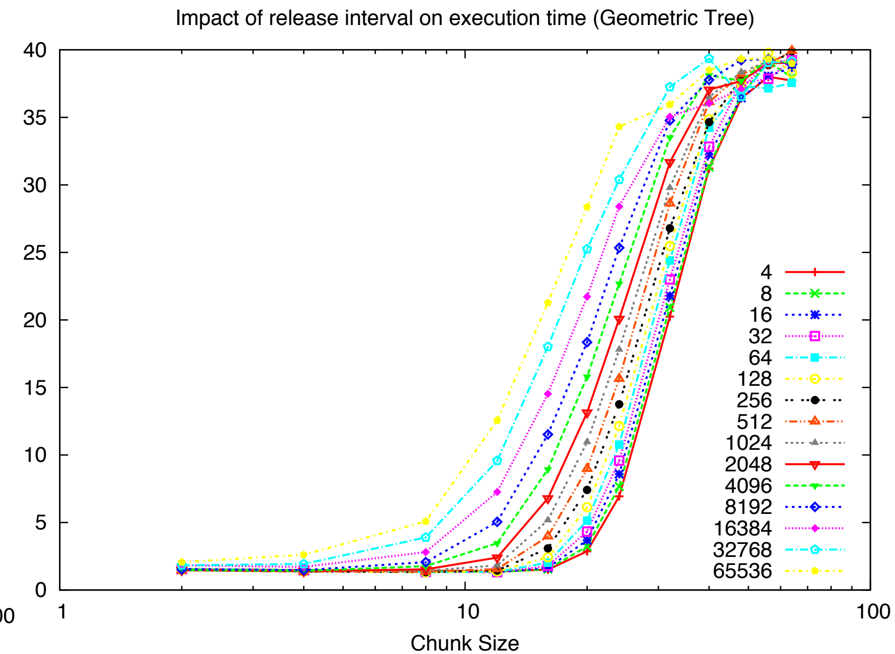


Original vs. Speculative Algorithm – Small Input (on 4 nodes, 12 cores each)

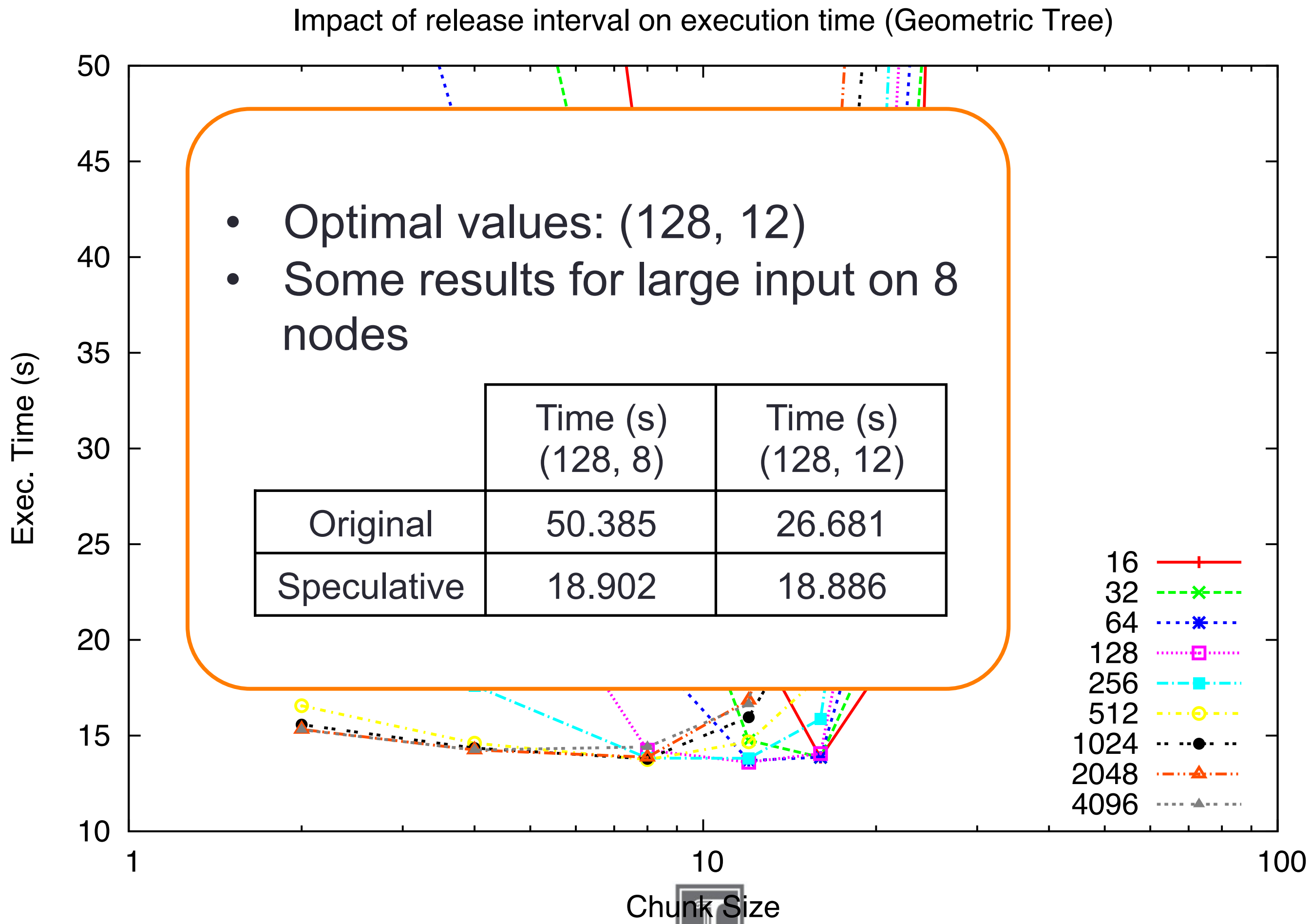
Original



Speculative



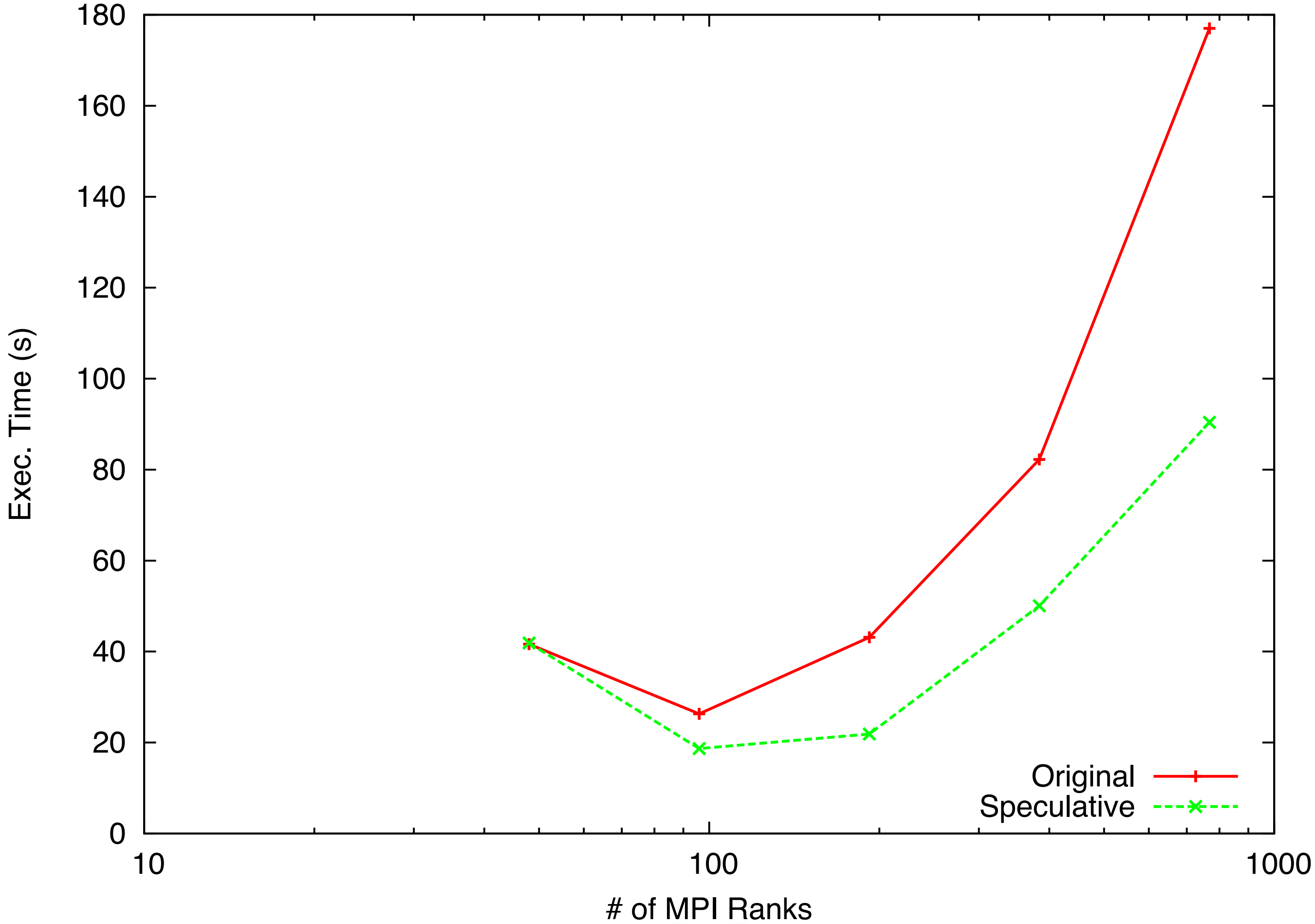
Tuning of Original Algorithm – Medium Input (on 4 nodes, 12 cores each)



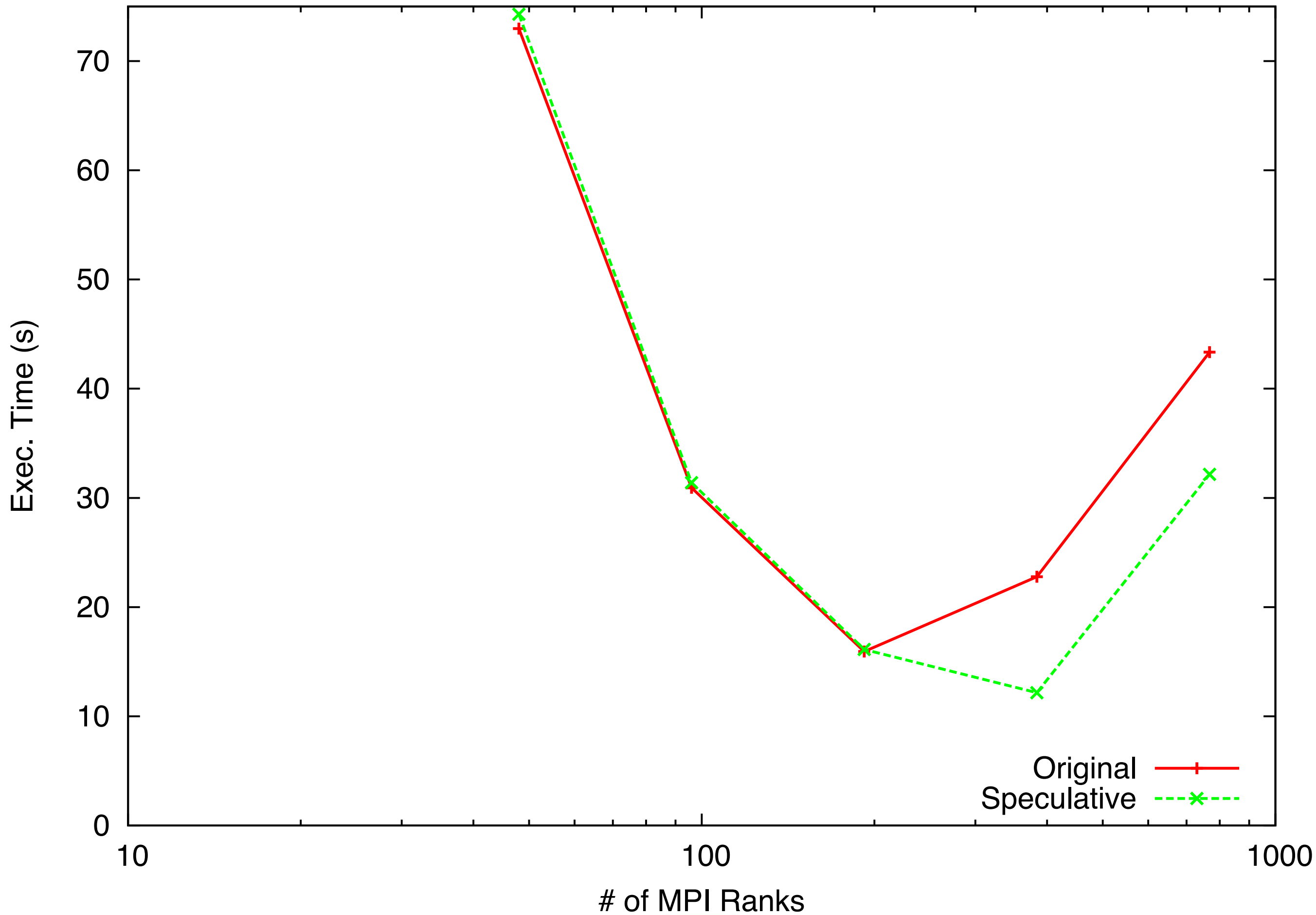
- Optimal values: (128, 12)
- Some results for large input on 8 nodes

	Time (s) (128, 8)	Time (s) (128, 12)
Original	50.385	26.681
Speculative	18.902	18.886

Scalability Study – Geometric Tree



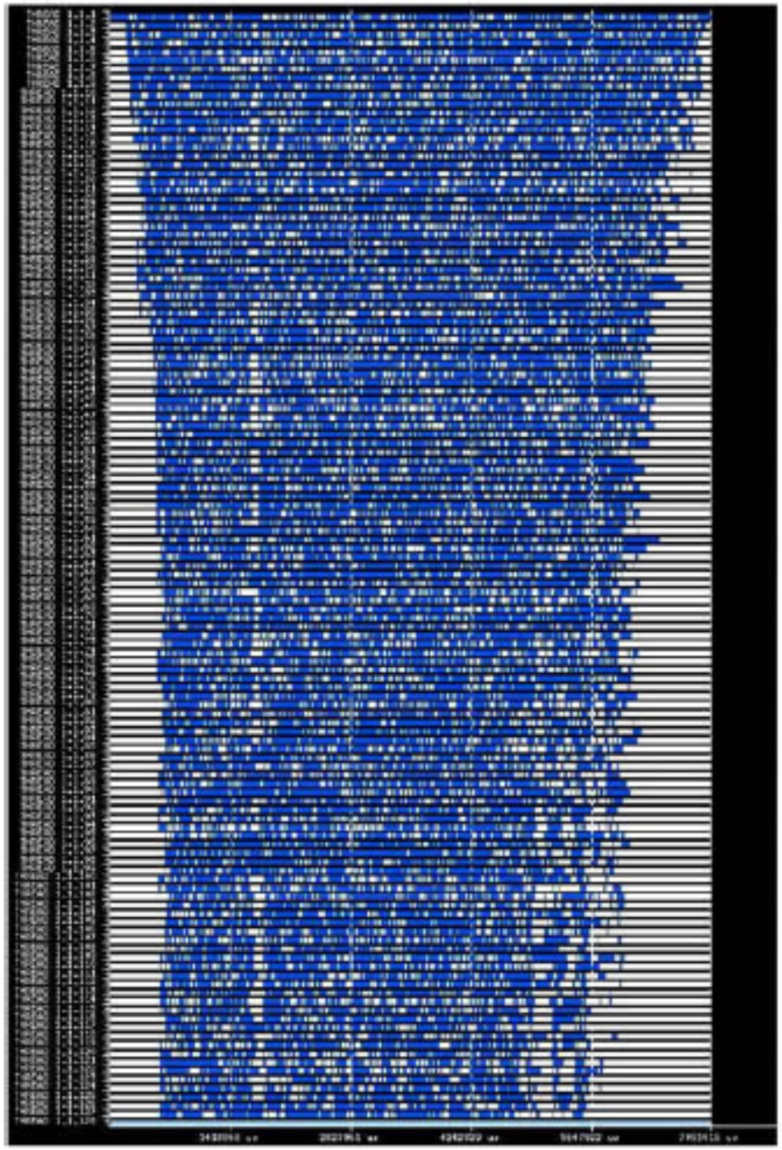
Scalability Study – Binomial Tree

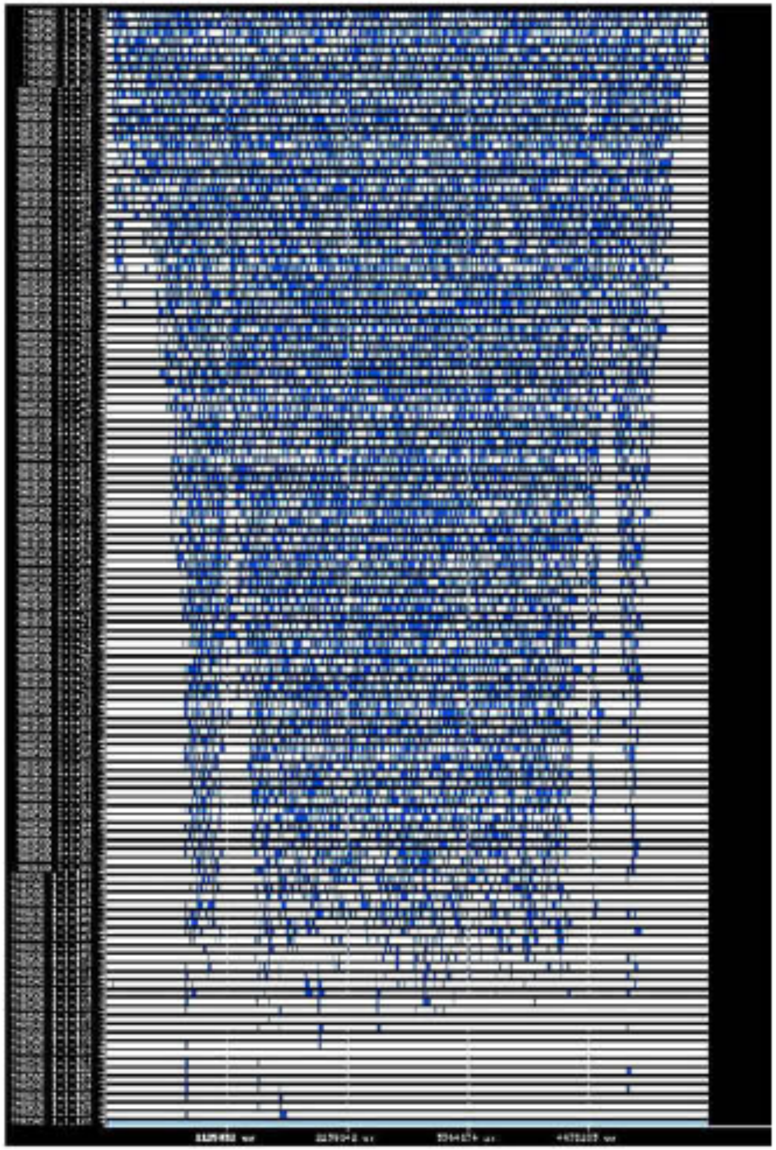


Conclusion

- Speculation
  - Is a **light-weight technique** in load-balancing algorithms
  - Is a potential solution to **eliminate idle time**
  - **Reduces sensitivity** of a load-balancing algorithm to parameters
  - Helps to **reduce tuning efforts**
  - Exhibits a **higher scalability**

Back Up Slides





Design Guidelines

- The time it takes to process a speculative task is far less than the time it takes to get response of an arbitration
  - A worker may need multiple speculative tasks at a time
- Low overhead algorithm to get speculative task
  - Minimal speculative task transfer (i.e. minimizing speculative task destroy)
- Quality of an speculative task decreases over time
  - Move actual task a worker has, less speculative task it should carry
- Quality of an speculative task increases as it goes deeper in its owner’s actual queue

Does Speculation Help Work Stealing?

- Base-line algorithm + speculative algorithm guidelines = speculative work stealing (Algorithm A)
- Speculative work stealing + replacing speculative messages with prefetching = optimized prefetch-based work stealing (Algorithm B)
- “A” has a slight performance benefit over “B” (less than 5 percent overall)
  - Reason: Even the base-line does not have too much idle time in UTS
- … But, speculative work stealing is helpful in problems where there is a limited parallelism due to data dependence
  - Example: Depth-first traversal of a graph