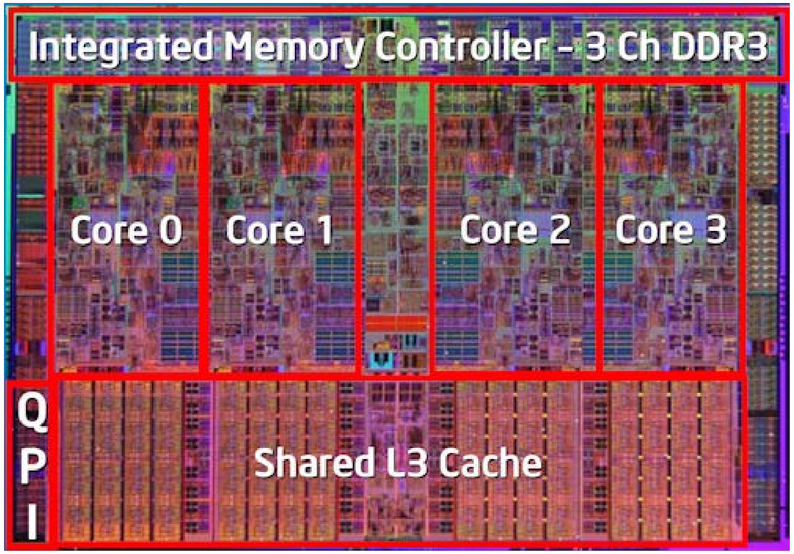
Dynamic Task and Data Placement over NUMA Architectures: an OpenMP Runtime Perspective

François Broquedis, Nathalie Furmento, Brice Goglin,
Raymond Namyst and Pierre-André Wacrenier

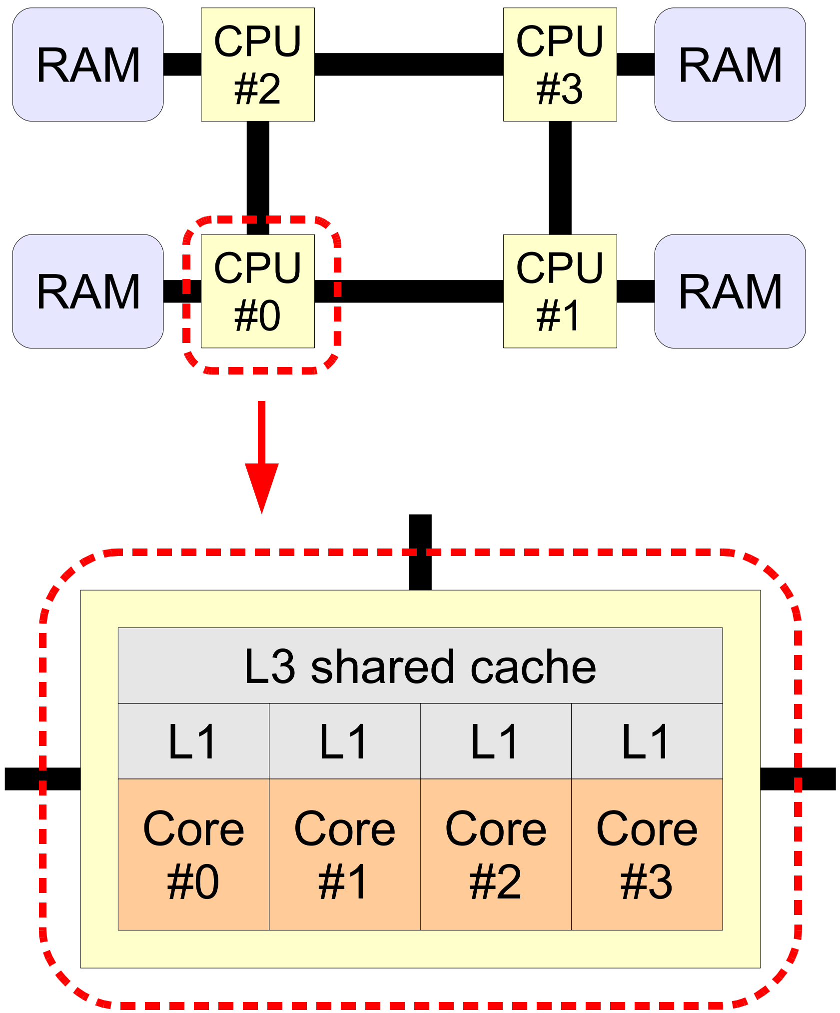
Université Bordeaux 1
RUNTIME Team, LaBRI – INRIA, France

Multi-core is a solid architecture trend

- Multi-core chips
  - Increasing number of cores sharing memory
  - Different from SMPs
    - Hierarchical chips
    - Getting really complex
  - Back to the CC-NUMA era?
    - AMD Hypertransport
    - Intel QuickPath



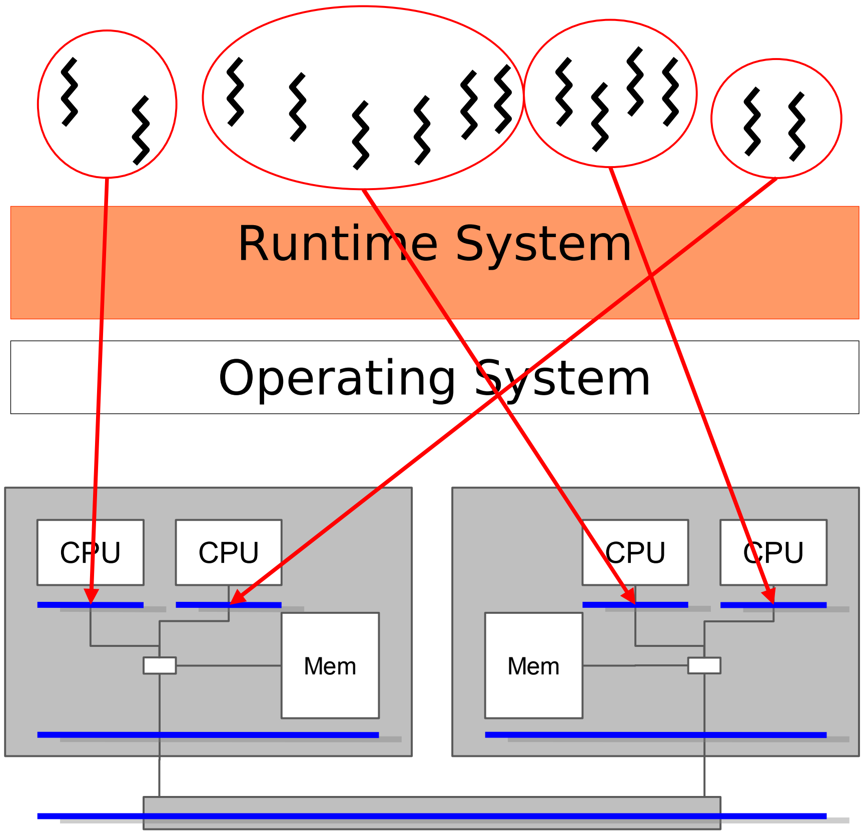
About hierarchical chips



- Hierarchical Chips
  - More than one thread per core
  - Shared resources between cores
- Cache affinity
  - False sharing problem
  - Low cache reuse
  - Solution: related threads on cache-sharing cores
    - Cache memory reuse
    - Better synchronizations

Our background: thread scheduling over multi-core machines

- The Bubble Scheduling concept
  - Capturing application’s structure with nested bubbles
  - Modeling the computer architecture using a tree of runqueues
  - Scheduling = dynamic mapping trees of threads onto a tree of runqueues
- The BubbleSched platform
  - Designing portable NUMA-aware scheduling policies
  - Debugging/tuning scheduling algorithms

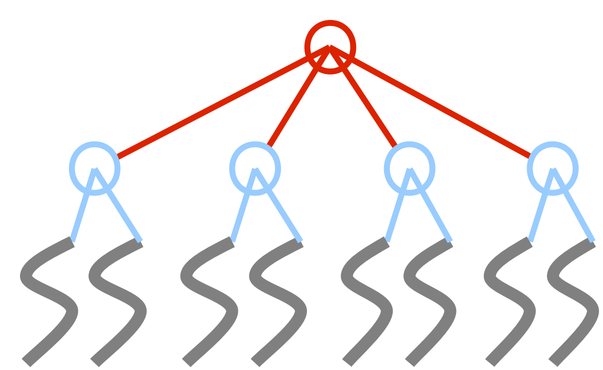


Our background: thread scheduling over multi-core machines

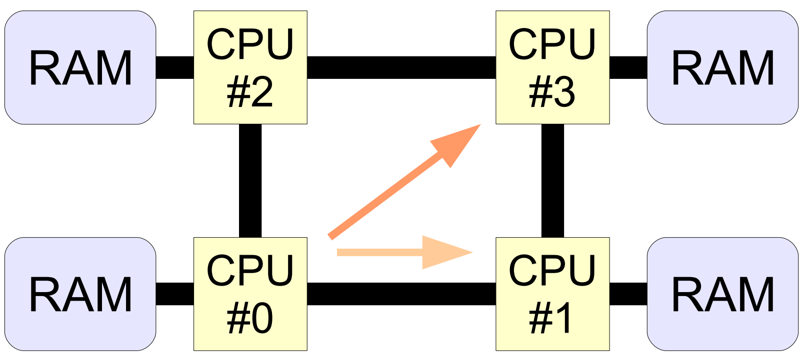
- Designing multi-core-friendly programs with OpenMP
  - Parallel sections generate bubbles
  - Nested parallelism is welcome!
    - Lazy creation of threads
- The ForestGOMP platform
  - Extension of GNU OpenMP
    - Binary compatible with existing applications
  - Excellent speedups with some irregular applications

```
void job()
{
  ...

#pragma omp parallel for
  for (int i=0; i<MAX; i++)
    {
      ...
#pragma omp parallel for
    num_threads (2)
      for (int k=0; k<MAX; k++)
        ...
    }
}
```



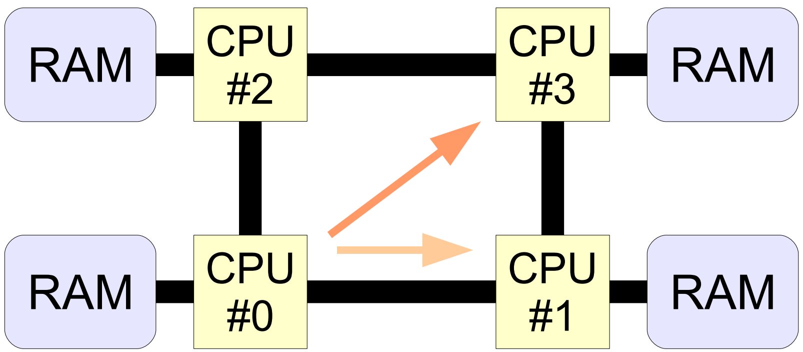
The NUMA problem



Access to...	Local node	Neighbor node	Opposite node
Read	83 ns	98 ns (x1.18)	117 ns (x1.41)
Write	142 ns	177 ns (x1.25)	208 ns (x1.46)

- Non-Uniform Memory Accesses
  - The NUMA factor
  - Write access = more trafic
- Memory affinity
  - Applications run faster if accessing local data
  - Need a carefull distribution of threads and data
    - To avoid NUMA penalties
    - To reduce memory contention

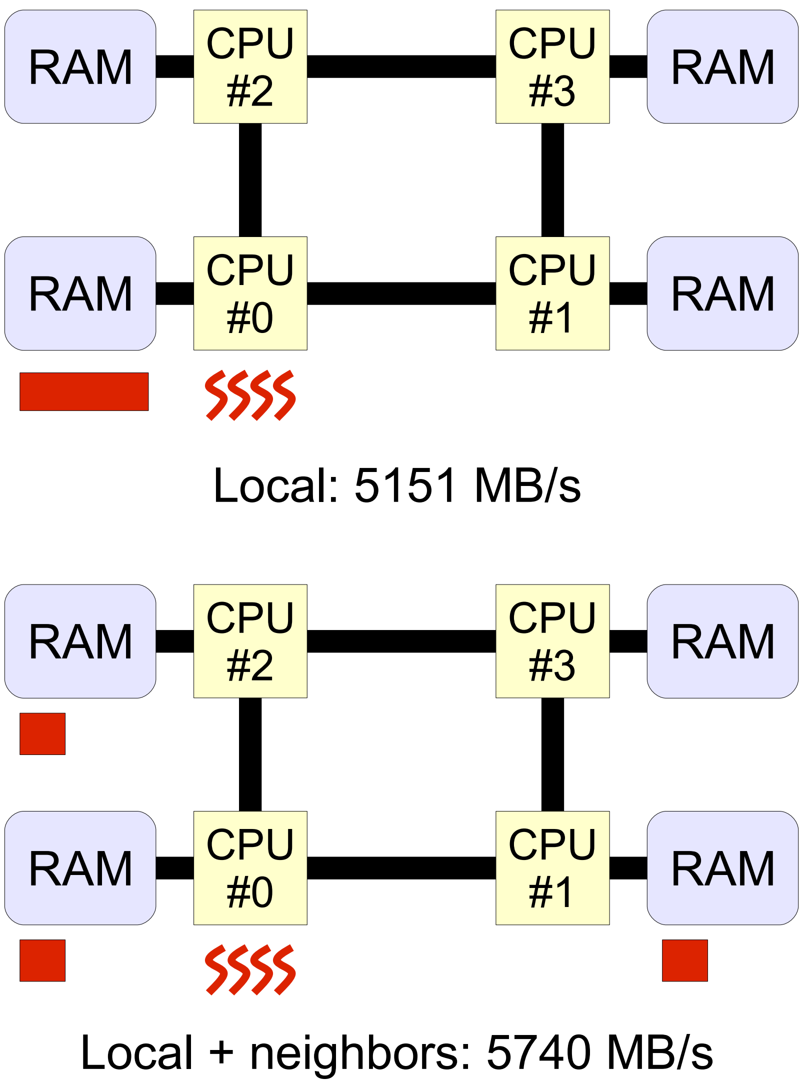
The NUMA problem



Access to...	Local node	Neighbor node	Opposite node
Read	83 ns	98 ns (x1.18)	117 ns (x1.41)
Write	142 ns	177 ns (x1.25)	208 ns (x1.46)

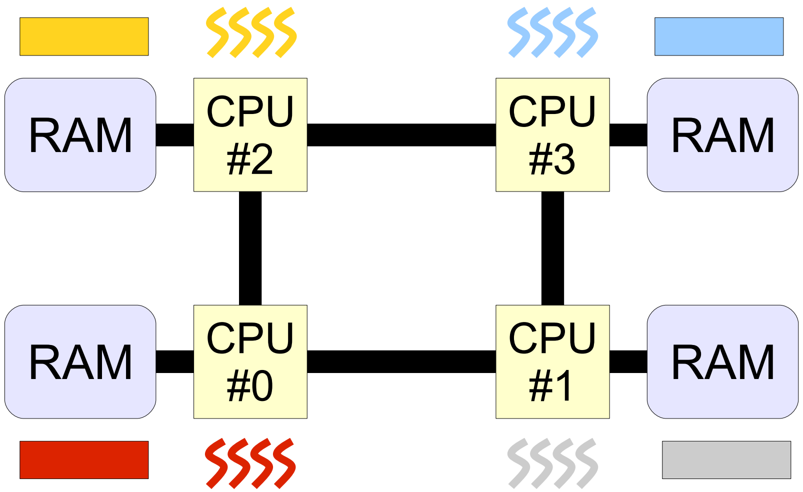
- Software support to deal with data locality
  - The first-touch allocation policy
    - Improves thread/data locality
  - The next-touch allocation policy
    - Improves performance of irregular applications
  - Common issues
    - Ignore the underlying system state
    - Need an « artificial » parallel initialization
    - Undefined behavior in some situations

Preliminary experiment: measuring the contention

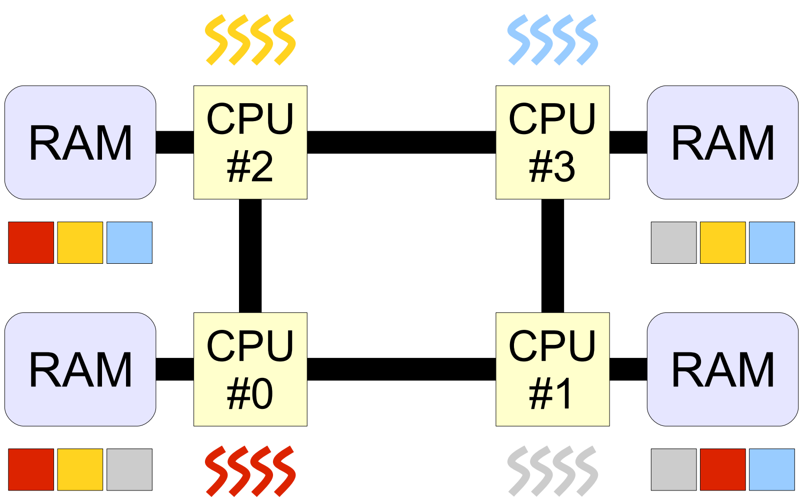


- Synthetic application
  - A few threads randomly accessing a memory area
  - Two different data placement policies
    - Allocate locally
    - Spread the memory pages on the neighbor nodes
- Results
  - Non-loaded computer
    - Spread solution is better

Preliminary experiment: measuring the contention



Local: 4 x 3635 MB/s



Local + neighbors: 4 x 2257 MB/s

- Synthetic application
  - A few threads randomly accessing a memory area
  - Two different data placement policies
    - Allocate locally
    - Spread the memory pages on the neighbor nodes
- Results
  - Non-loaded computer
    - Spread solution is better
  - Loaded computer
    - Local policy is better

Data placement may also depend on the system state

Thwarting the contention and load-induced bottlenecks

- ## The contention problem
  - Best data distribution can change depending on memory contention
  - Architecture-dependent problem
- ## The load problem
  - Can't migrate memory to an out-of-memory node
- ## Are thread-centric policies enough?
  - first-touch, next-touch: thread-centric policies
  - OpenMP threads/tasks are meant to move!
  - Ignore the current system state!

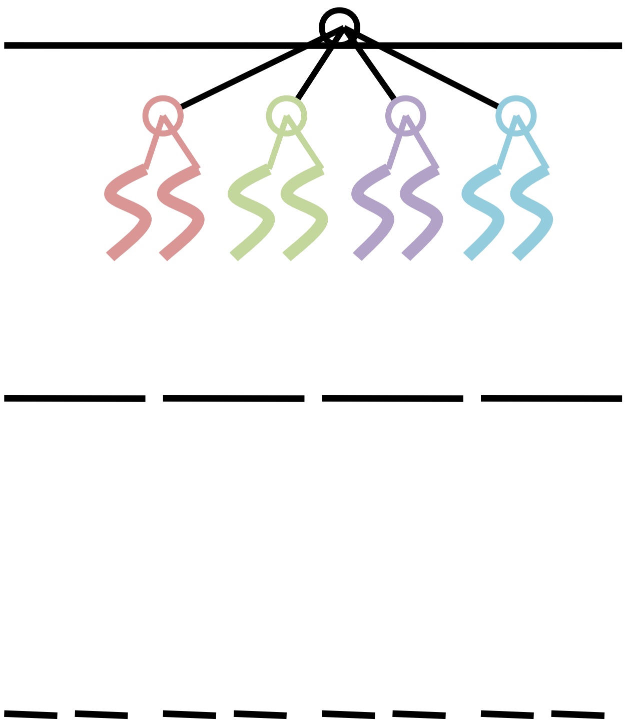
Our approach: Let the runtime system in charge!

A runtime approach to account for NUMA

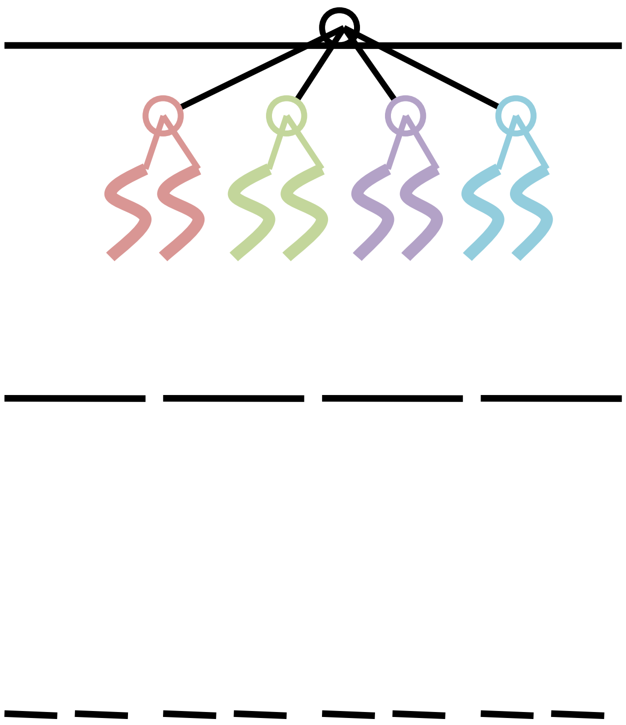
- Express memory affinity
  - Transmit the application programmer knowledge to the runtime system
- Schedule threads according to their memory affinity
  - Keep threads and attached data together the longer we can
  - Steal work considering the computer load and NUMA nodes state
- Technical contribution
  - A programming interface to express memory affinity
  - A specific bubble scheduler for coordinate task/data placement

Memory: an affinity-aware bubble scheduler

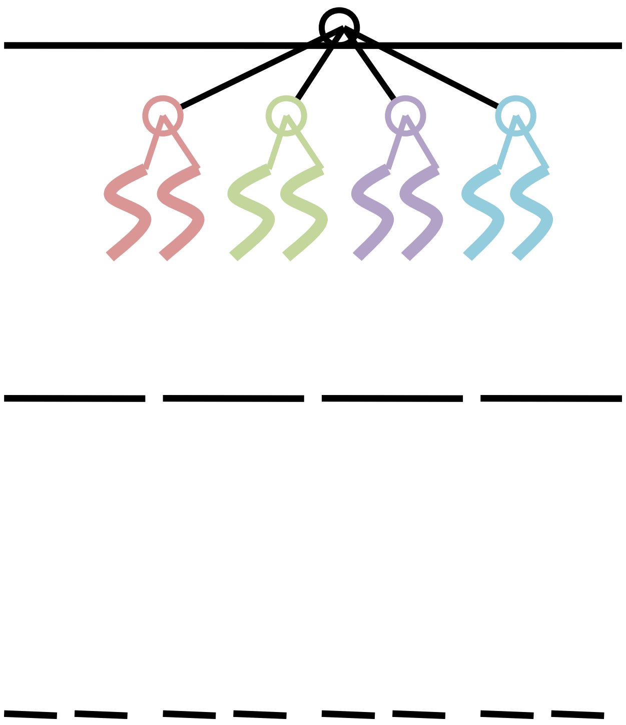
Main goal: make every thread access to local memory



Test case #1



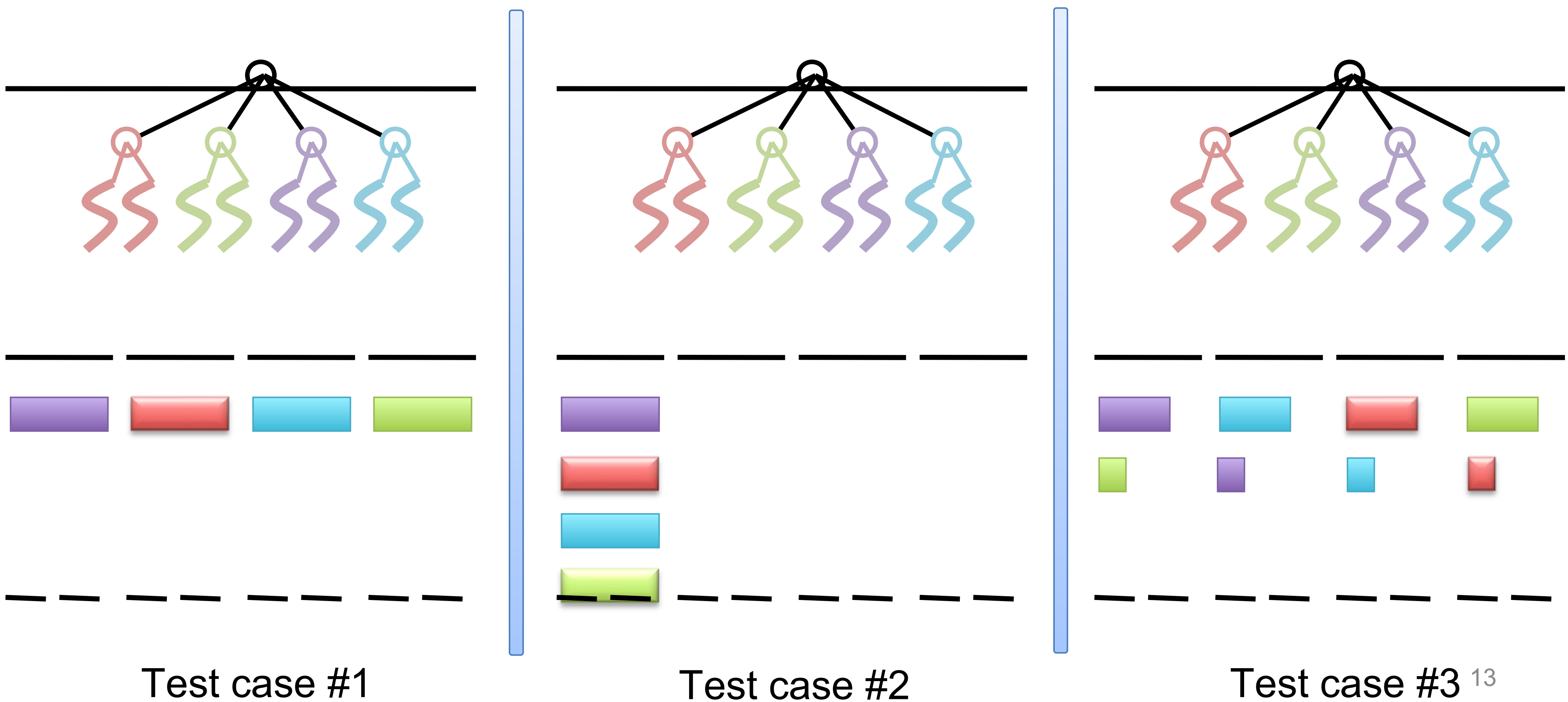
Test case #2



Test case #3

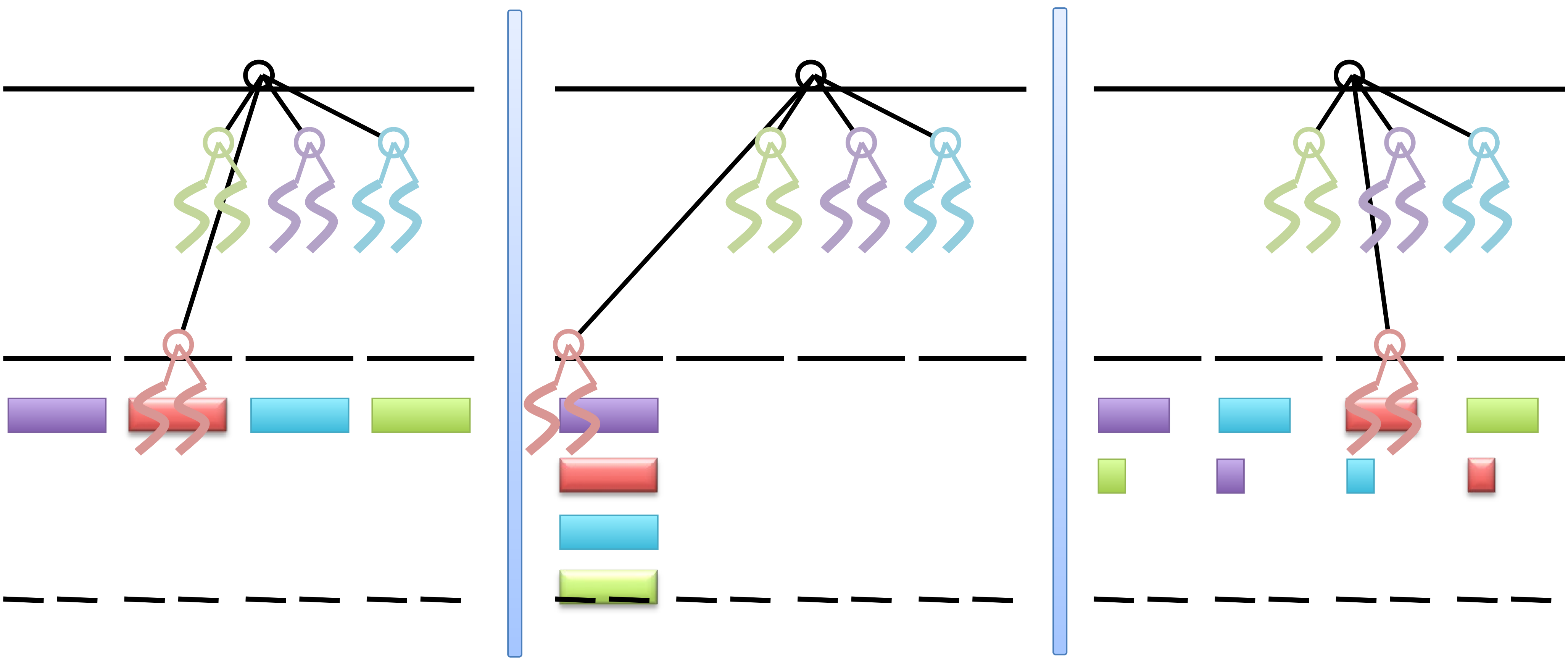
Memory: an affinity-aware bubble scheduler

Each test case has a different initial data distribution



Memory: an affinity-aware bubble scheduler

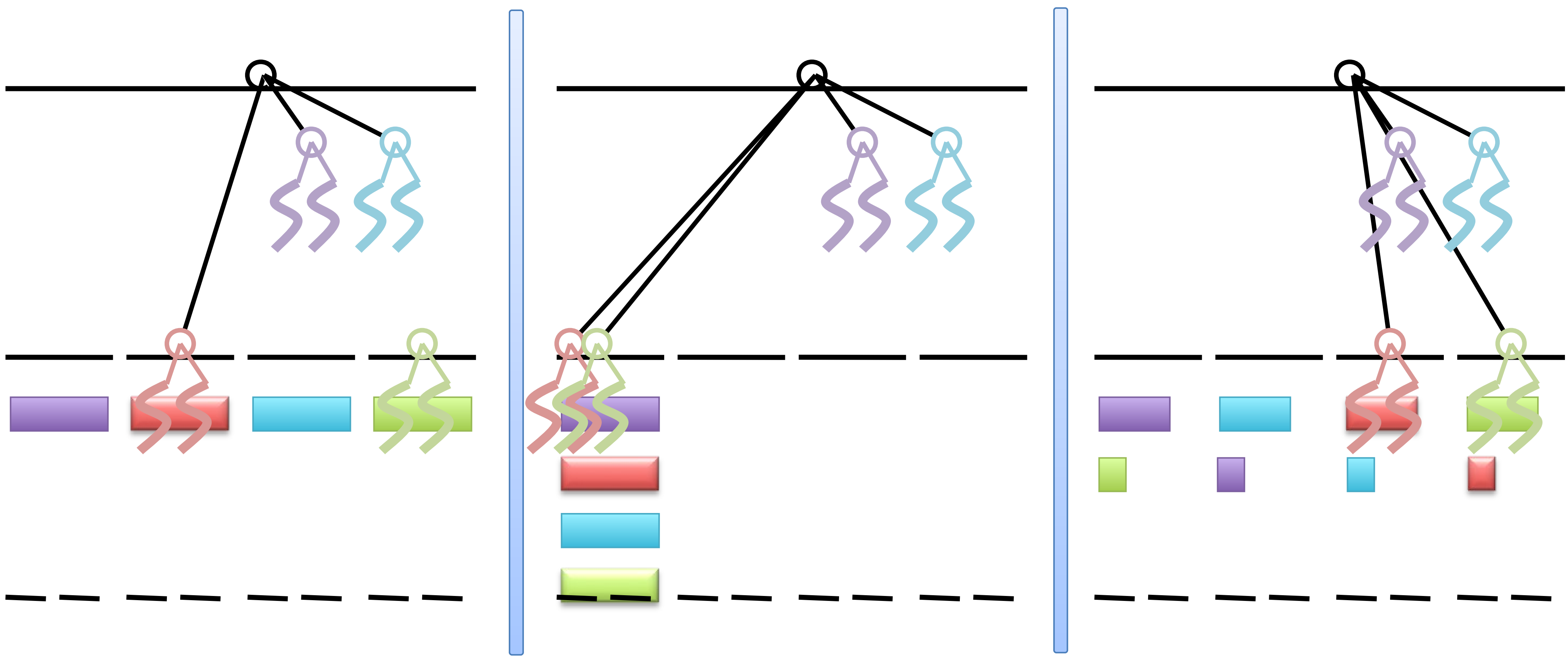
First phase: Draw the threads to the node holding the bigger amount of their data



Test case #1 Test case #2 Test case #3

Memory: an affinity-aware bubble scheduler

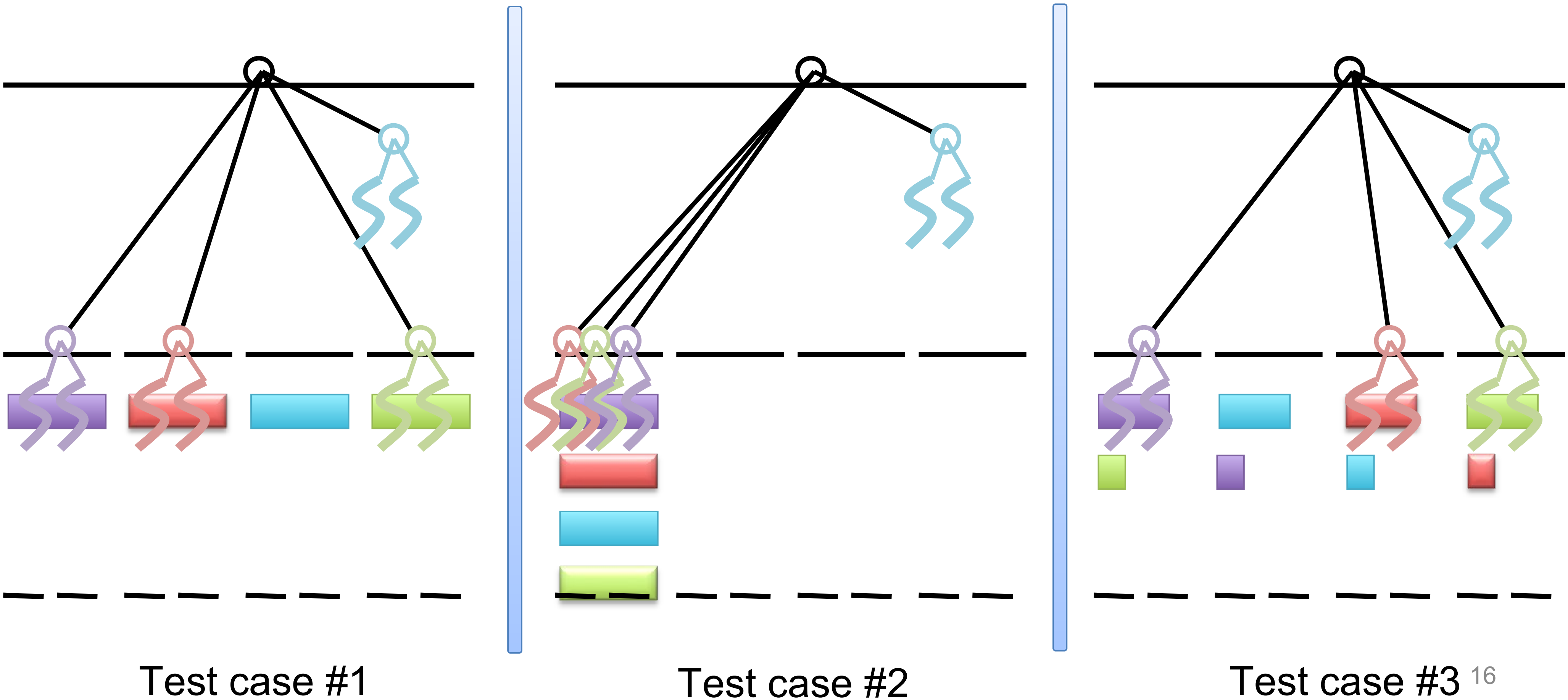
First phase: Draw the threads to the node holding the bigger amount of their data



Test case #1 Test case #2 Test case #3

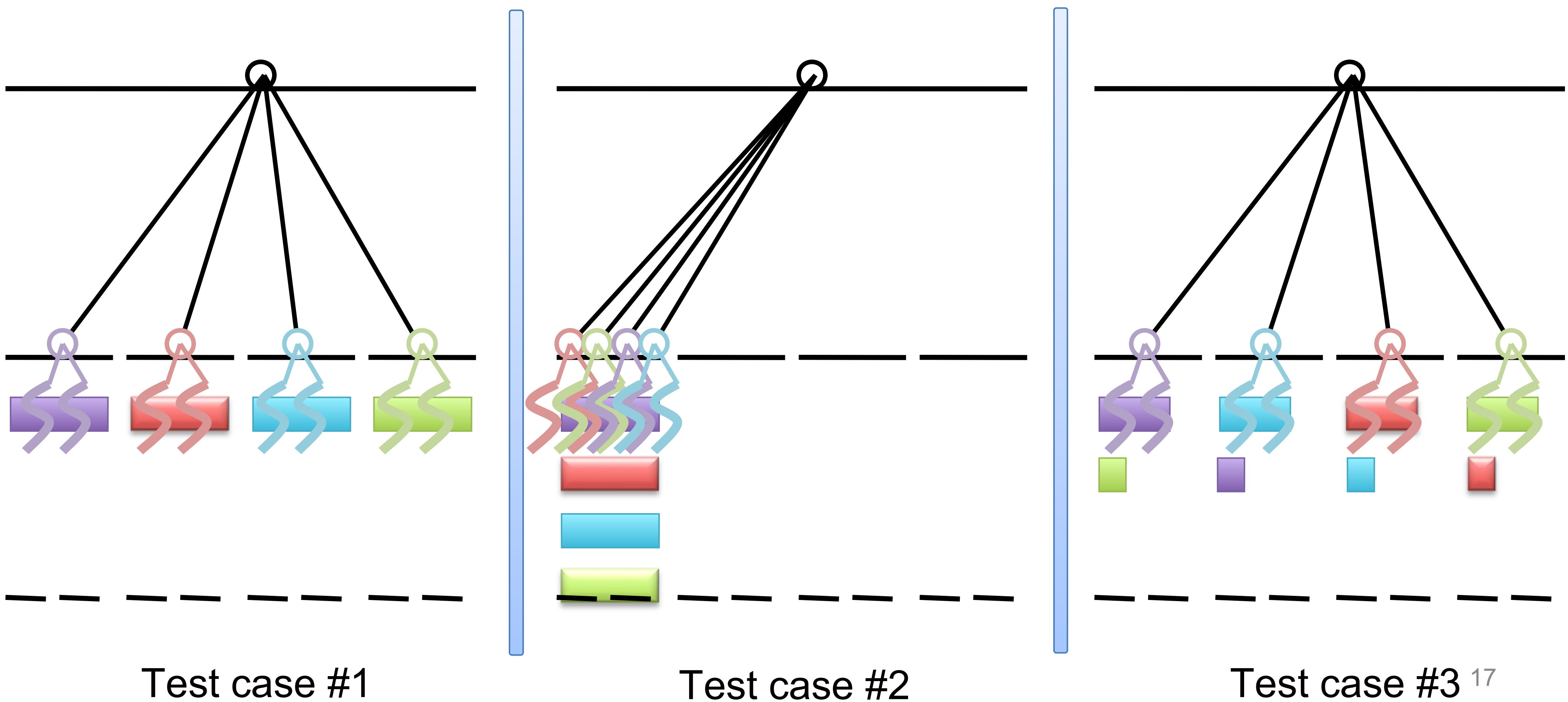
Memory: an affinity-aware bubble scheduler

First phase: Draw the threads to the node holding the bigger amount of their data



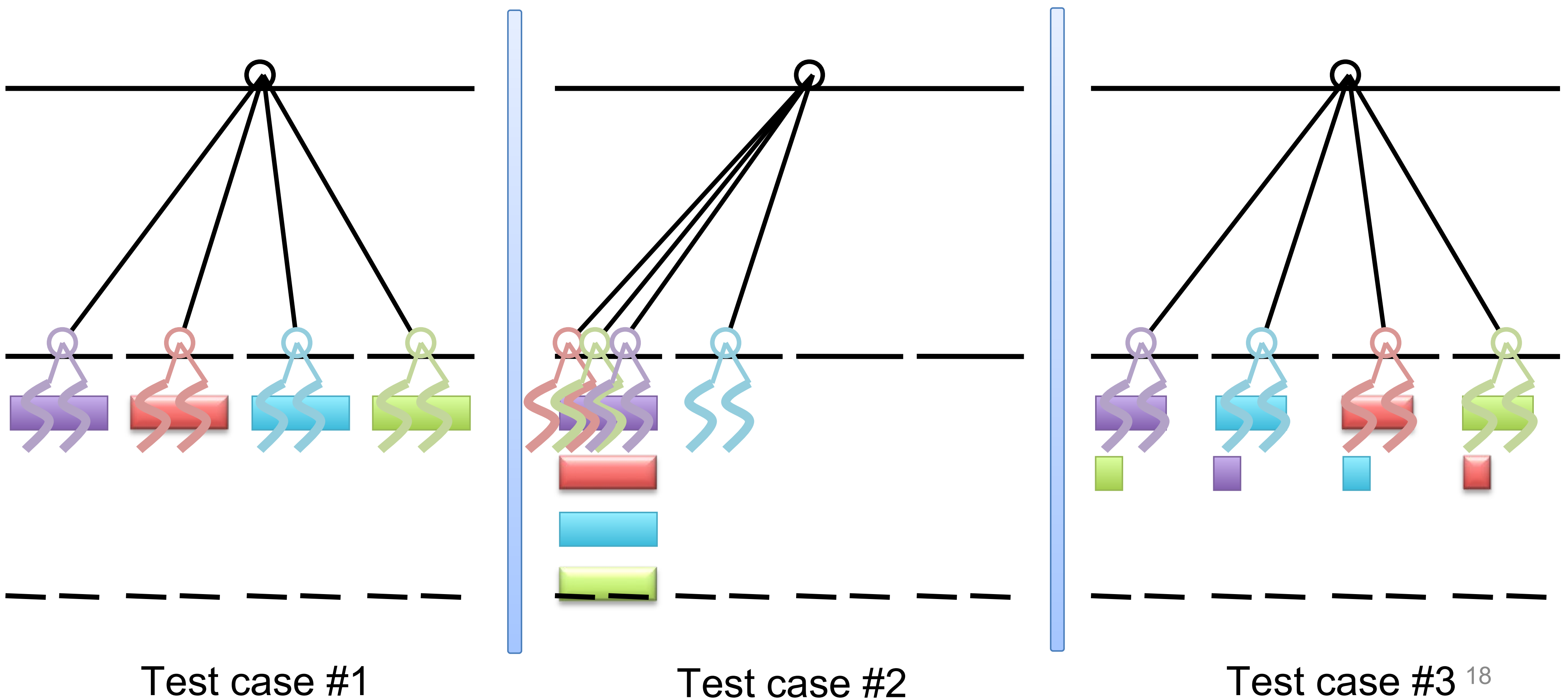
Memory: an affinity-aware bubble scheduler

First phase: Draw the threads to the node holding the bigger amount of their data



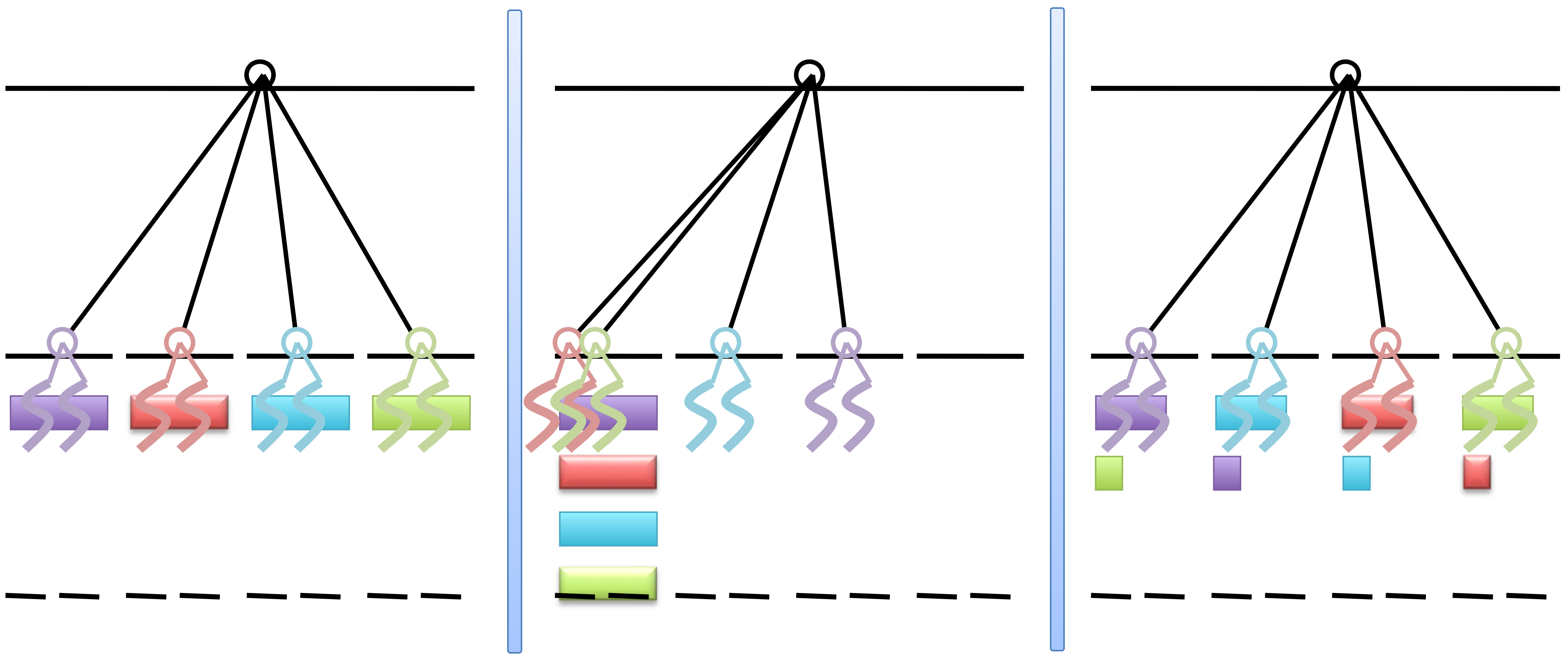
Memory: an affinity-aware bubble scheduler

Second phase: Balance the load to occupy every processor



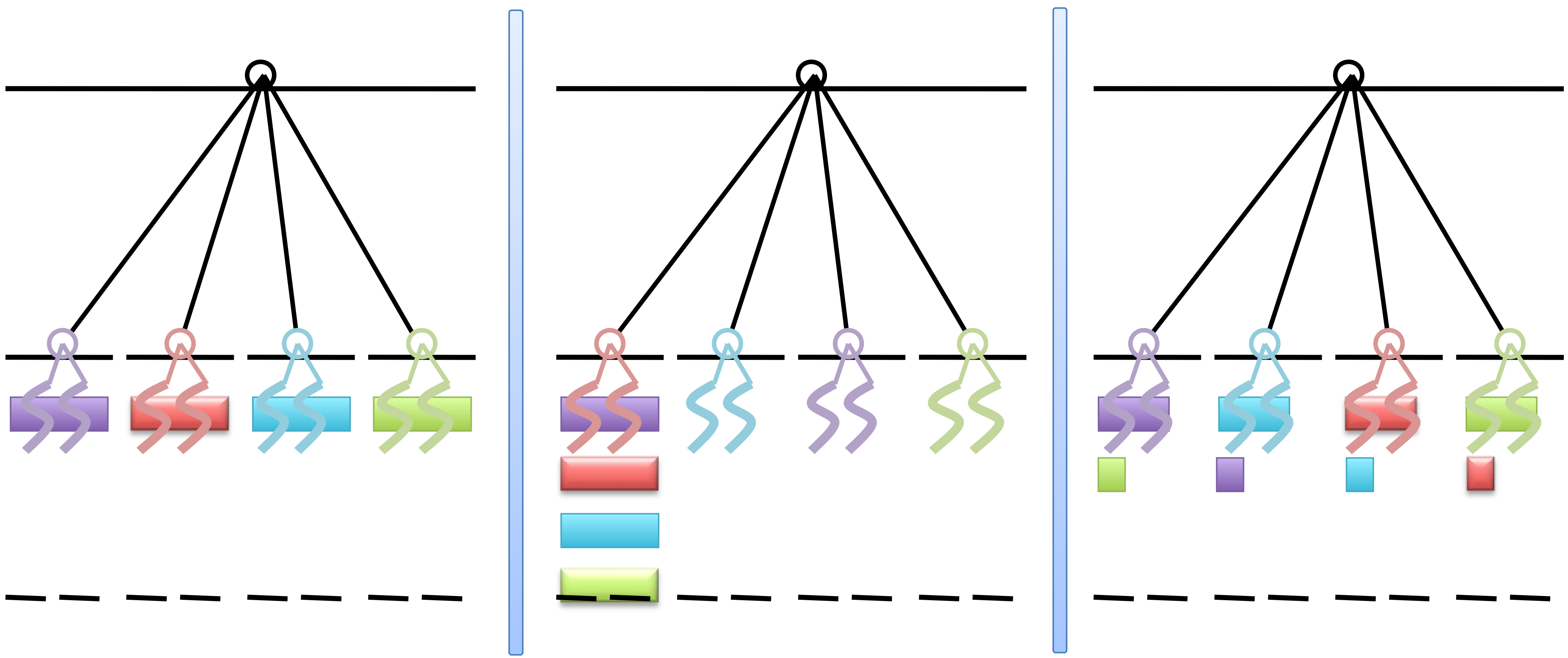
Memory: an affinity-aware bubble scheduler

Second phase: Balance the load to occupy every processor



Memory: an affinity-aware bubble scheduler

Second phase: Balance the load to occupy every processor



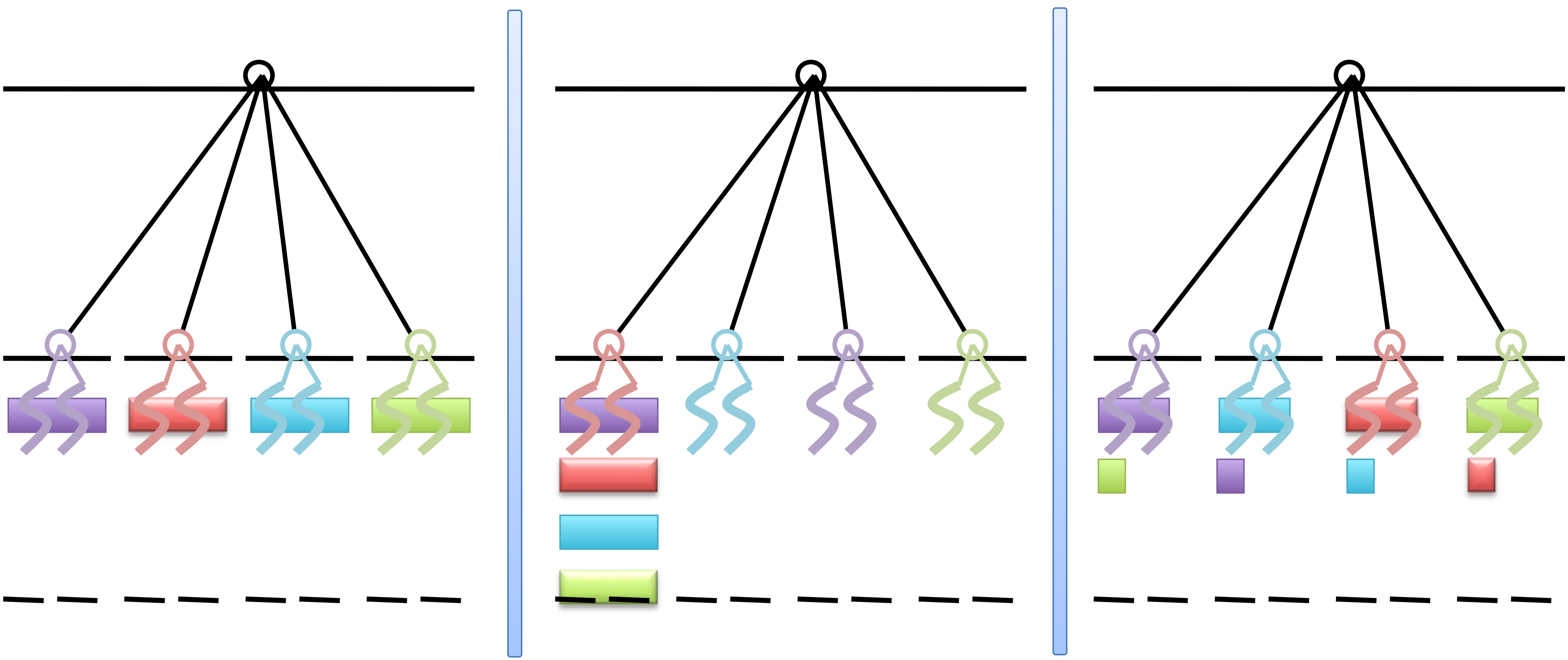
Test case #1

Test case #2

Test case #3

Memory: an affinity-aware bubble scheduler

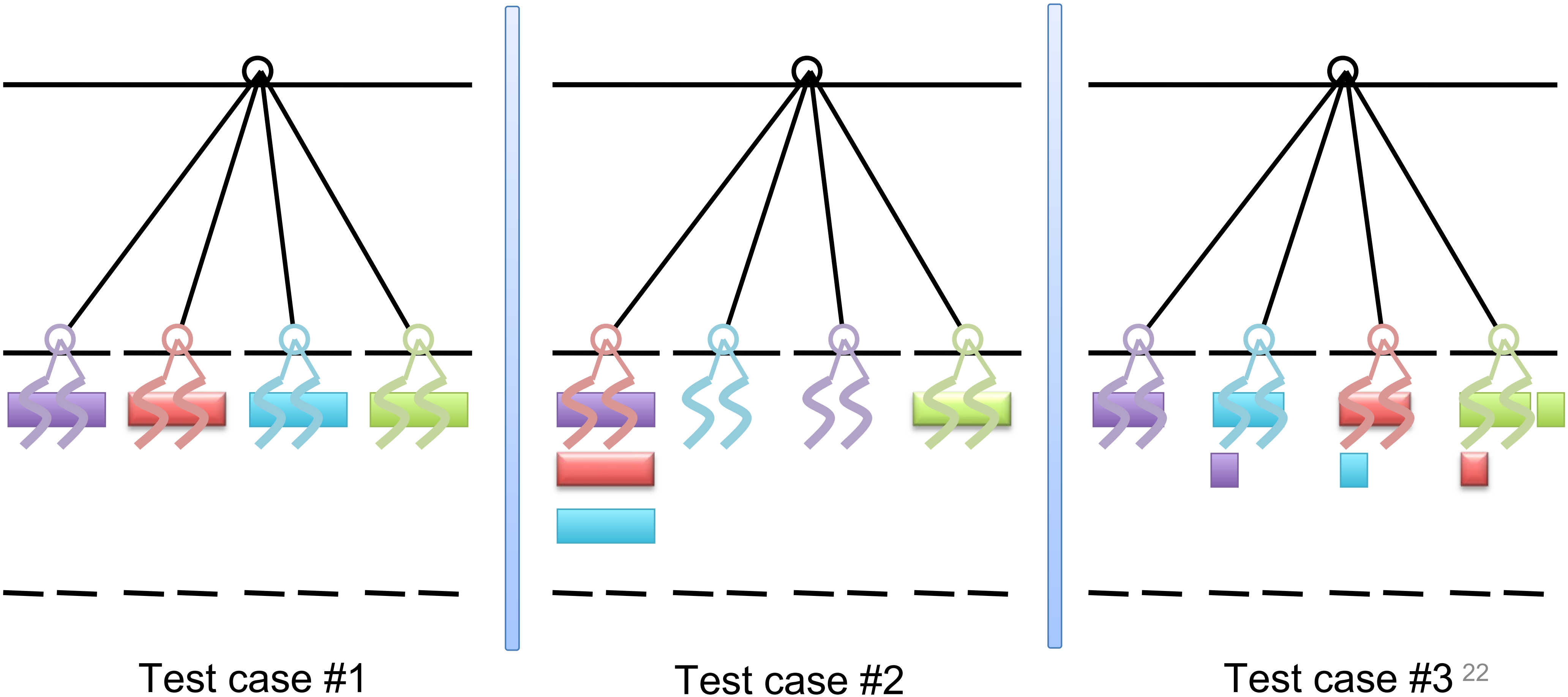
Third phase: Migrate the remaining distant data



Test case #1 Test case #2 Test case #3

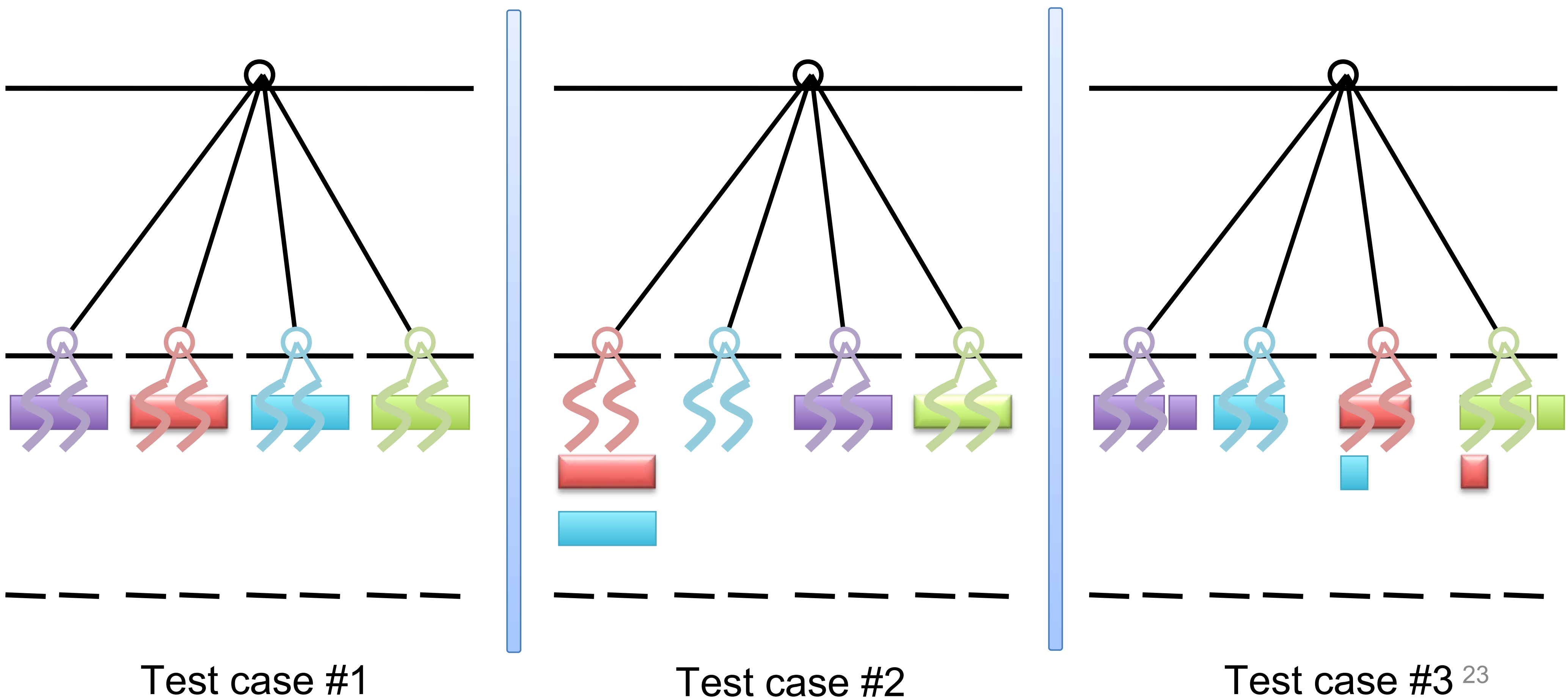
Memory: an affinity-aware bubble scheduler

Third phase: Migrate the remaining distant data



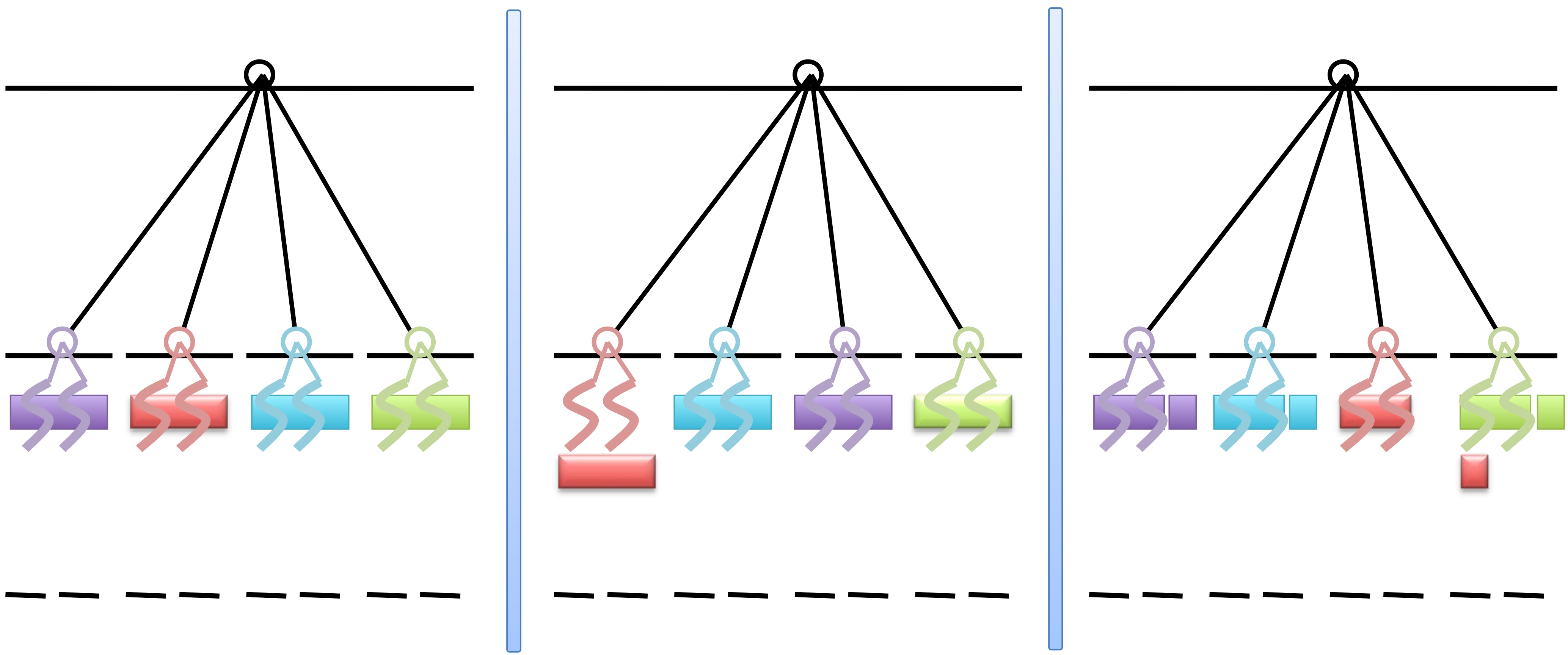
Memory: an affinity-aware bubble scheduler

Third phase: Migrate the remaining distant data



Memory: an affinity-aware bubble scheduler

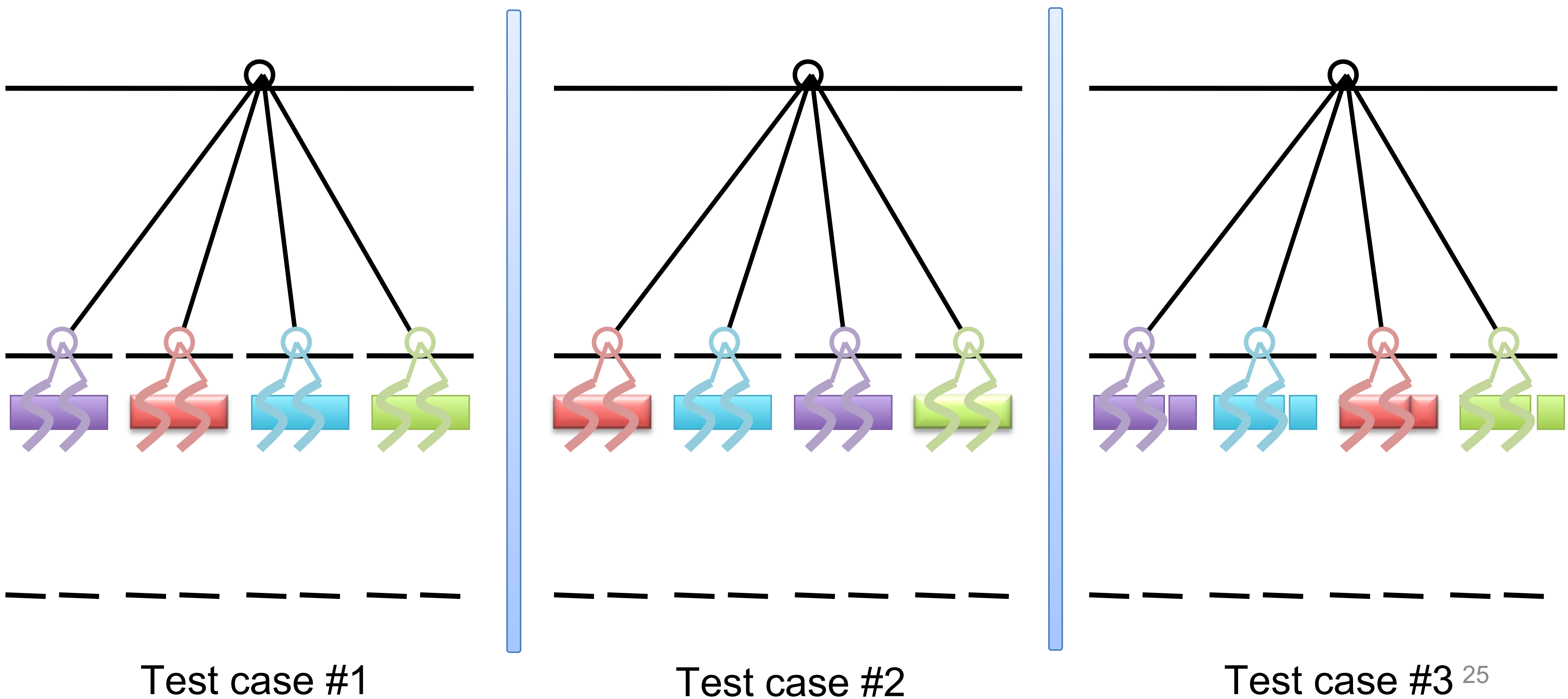
Third phase: Migrate the remaining distant data



Test case #1 Test case #2 Test case #3

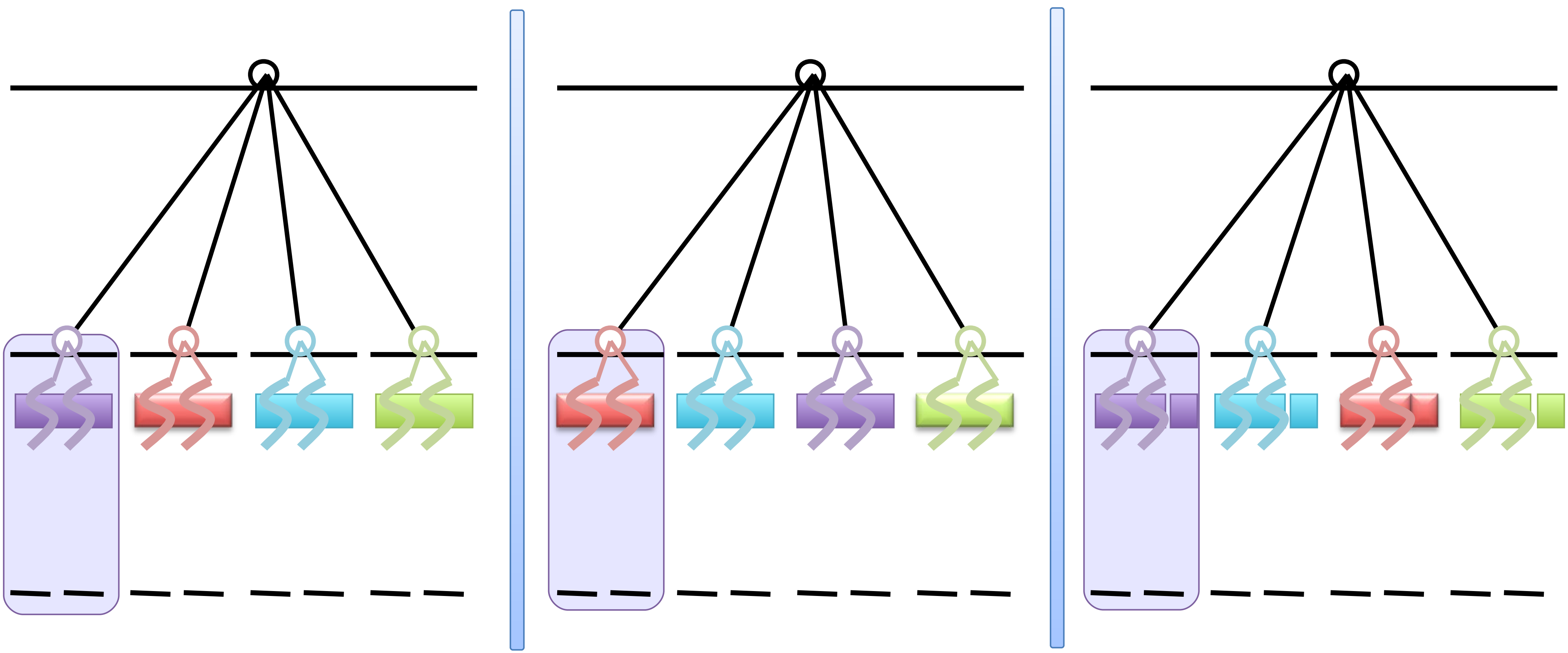
Memory: an affinity-aware bubble scheduler

Third phase: Migrate the remaining distant data



Memory: an affinity-aware bubble scheduler

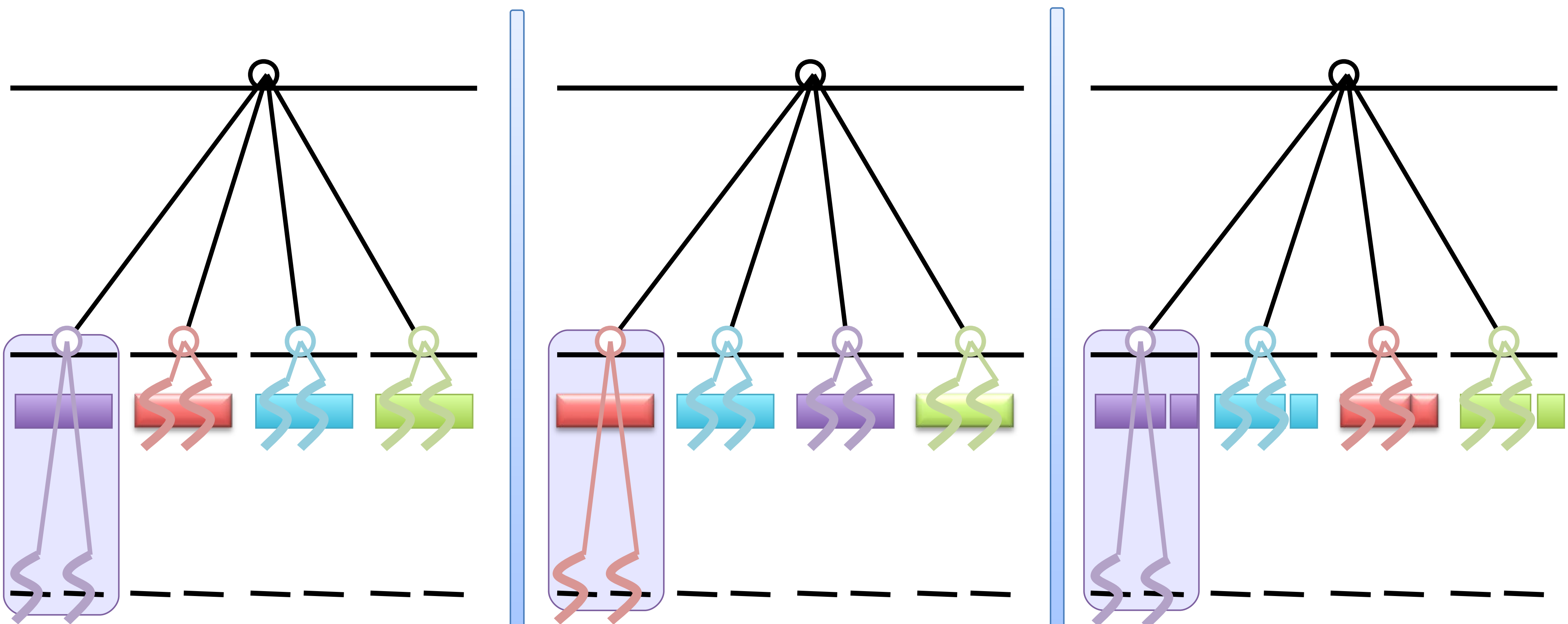
Eventually, call the Cache Bubble Scheduler inside each NUMA node



Test case #1 Test case #2 Test case #3

Memory: an affinity-aware bubble scheduler

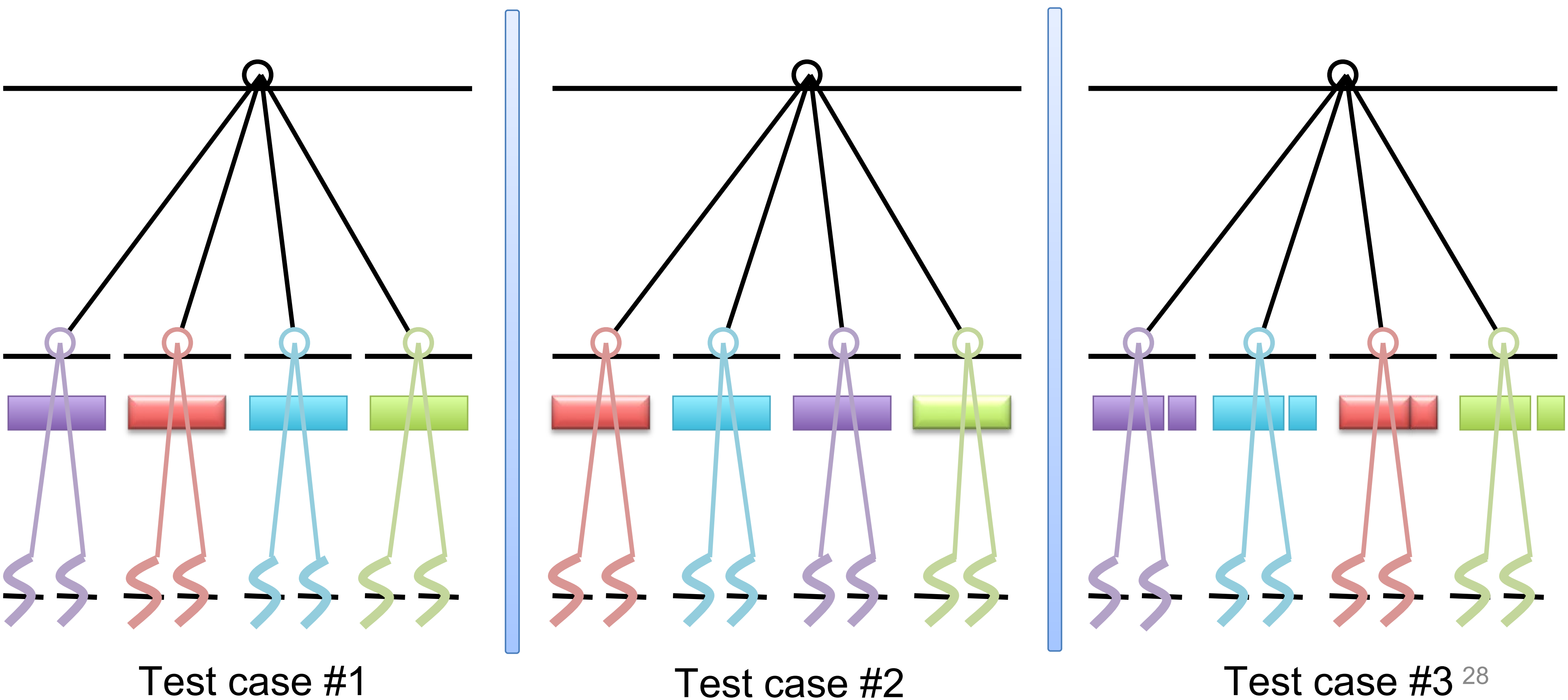
Eventually, call the Cache Bubble Scheduler inside each NUMA node



Test case #1 Test case #2 Test case #3

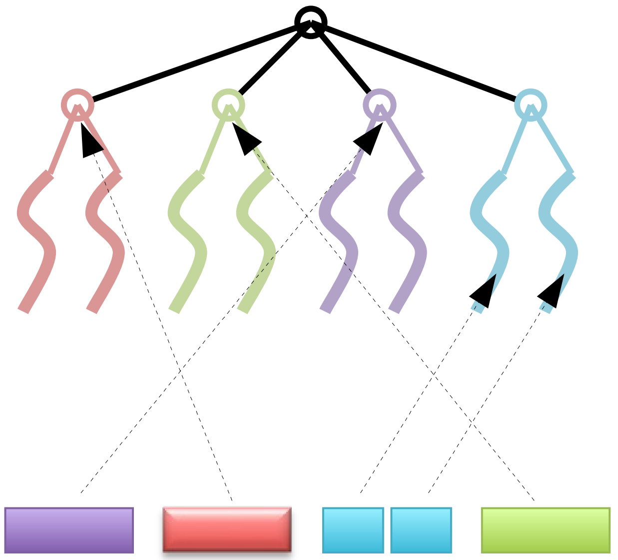
Memory: an affinity-aware bubble scheduler

Eventually, call the Cache Bubble Scheduler inside each NUMA node



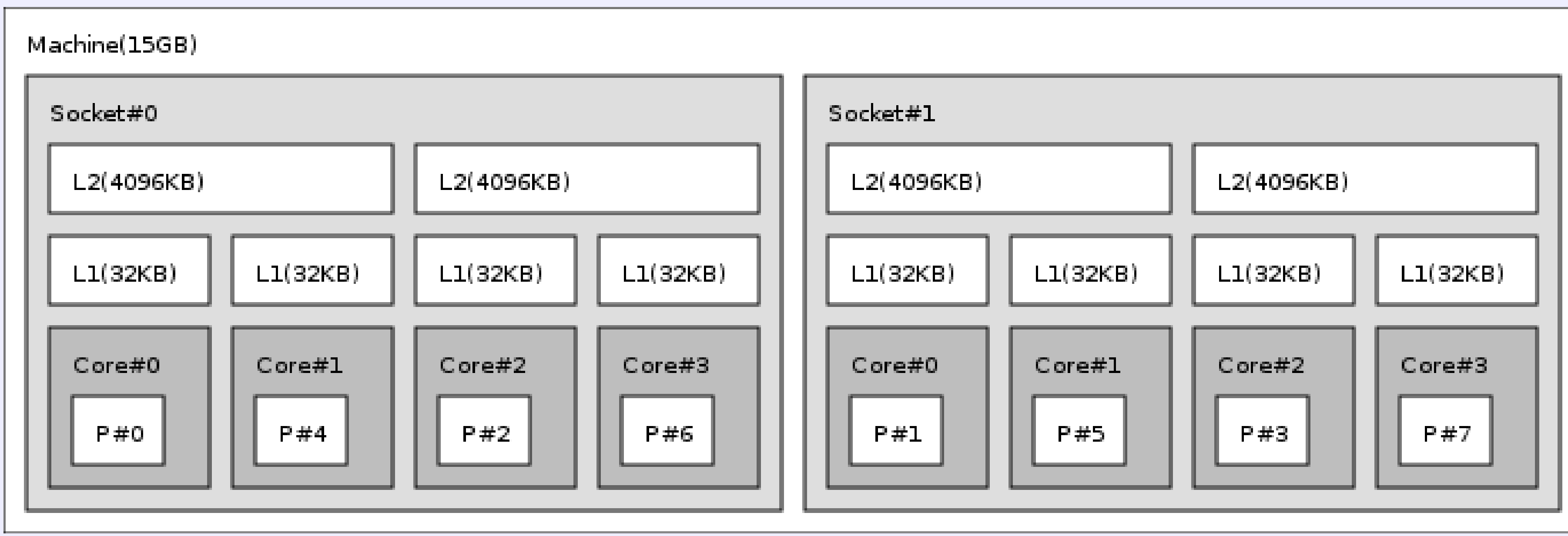
A programming interface to express memory affinities

- MaMI, a NUMA-aware Allocation Library
  - Implements first-touch, next-touch and explicit migration
  - Able to « attach » memory to Marcel threads and bubbles
- A ForestGOMP/MaMI Programming Interface for Memory Affinity Relations
  - The programmer can attach memory to OpenMP teams
    - Before a parallel region
    - Inside a parallel region
  - Synthesize affinity relations on bubbles



A portable library for modeling complex architectures

- Libtopology: a portable abstraction for hierarchical topologies
  - Generic expression of any computer architecture
  - http://runtime.bordeaux.inria.fr/libtopology/



A 2-socket quad-core Xeon computer

Performance evaluation: STREAM

- STREAM: An OpenMP Memory Benchmark
  - Sustainable memory bandwith
  - Arithmetic operations on simple vector kernels
- Inside STREAM
  - Data set: 3 vectors of double precision integers (A, B, C)
  - Successive operations:
    - Copy: C = A
    - Scale: B = scalar * C
    - Add: C = A + B
    - Triad: A = B + scalar * C
- ForestGOMP related assets
  - Schedule the threads over the cores

    => first-touch valid during the whole run

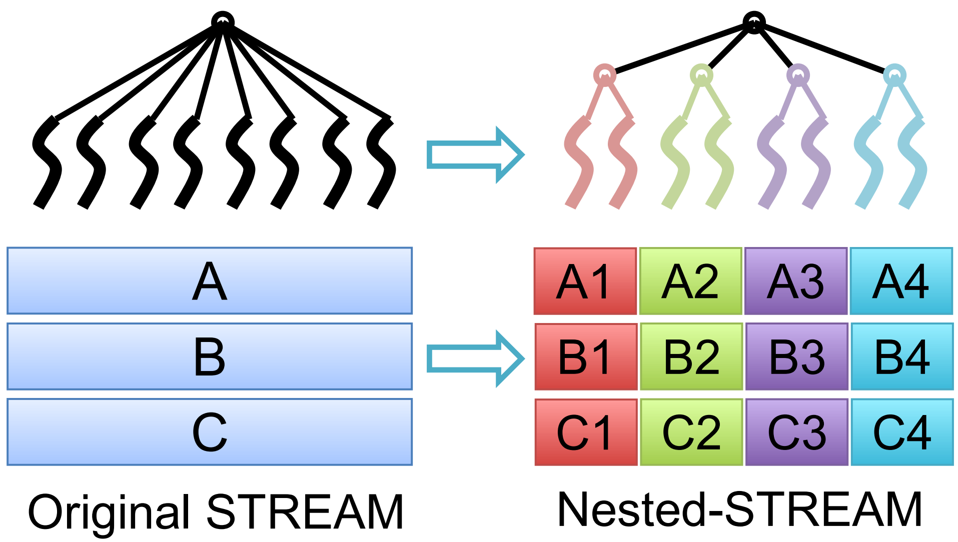
Compiler	Bandwidth (GB/s)
GCC 4.2	7.5 ± 1
Intel ICC 10.1	9.3 ± 1
ForestGOMP	8.9 ± 0.5

STREAM performance on a quad-socket quad-core Opteron computer

Performance evaluation: Nested-STREAM

- Parallel STREAM instances
  - Nested OpenMP parallel regions
  - As many OpenMP teams as NUMA nodes
  - A set of STREAM vectors per team
  - first-touch + parallel initialization
- ForestGOMP related assets
  - Create threads next to their master
  - Schedule one team per NUMA node

  => first-touch valid during the whole run



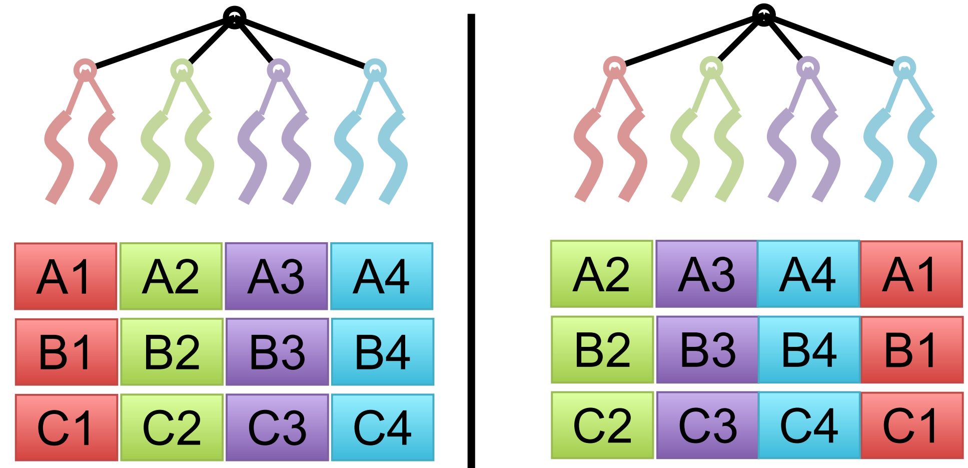
Compiler	**Bandwidth (GB/s)**
GCC 4.2	7.5 ± 1
Intel ICC 10.1	8 ± 1
ForestGOMP	8.9 ± 0.5

Nested-STREAM performance on a quad-socket quad-core Opteron computer

Twisted-STREAM

- More complex memory access pattern
  - Two distinct phases accessing different data
  - First phase = Nested-STREAM
  - Second phase, team “ i ” accesses to vectors “ i + 1 ”

- Two versions
  - Twisted-100: team “ i “ works on vectors A_{i+1}, B_{i+1}, C_{i+1}
  - Twisted-66: team “ i “ works on vectors A_i, B_{i+1}, C_{i+1}

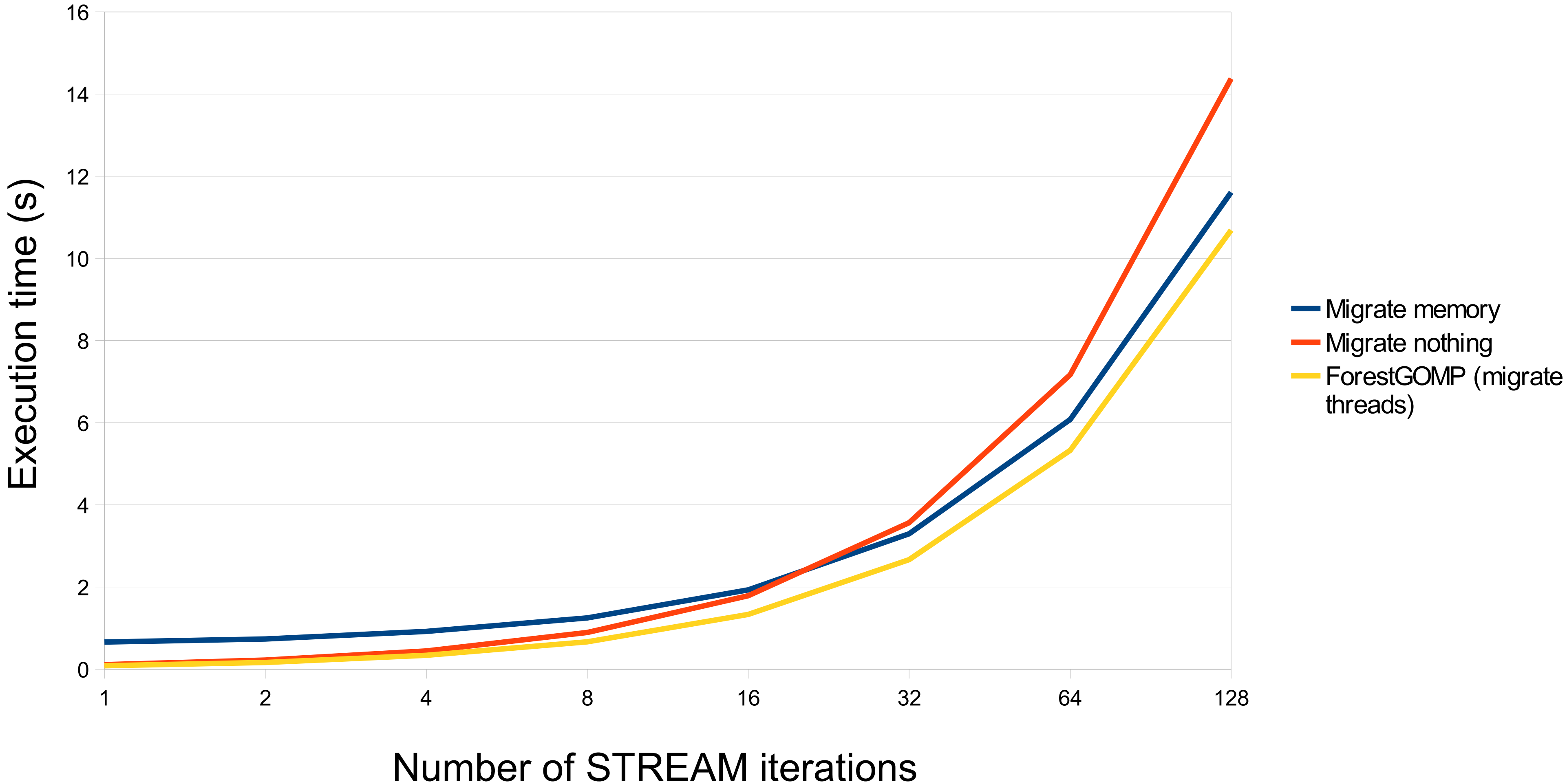


Phase 1 Phase 2 (Twisted-100)

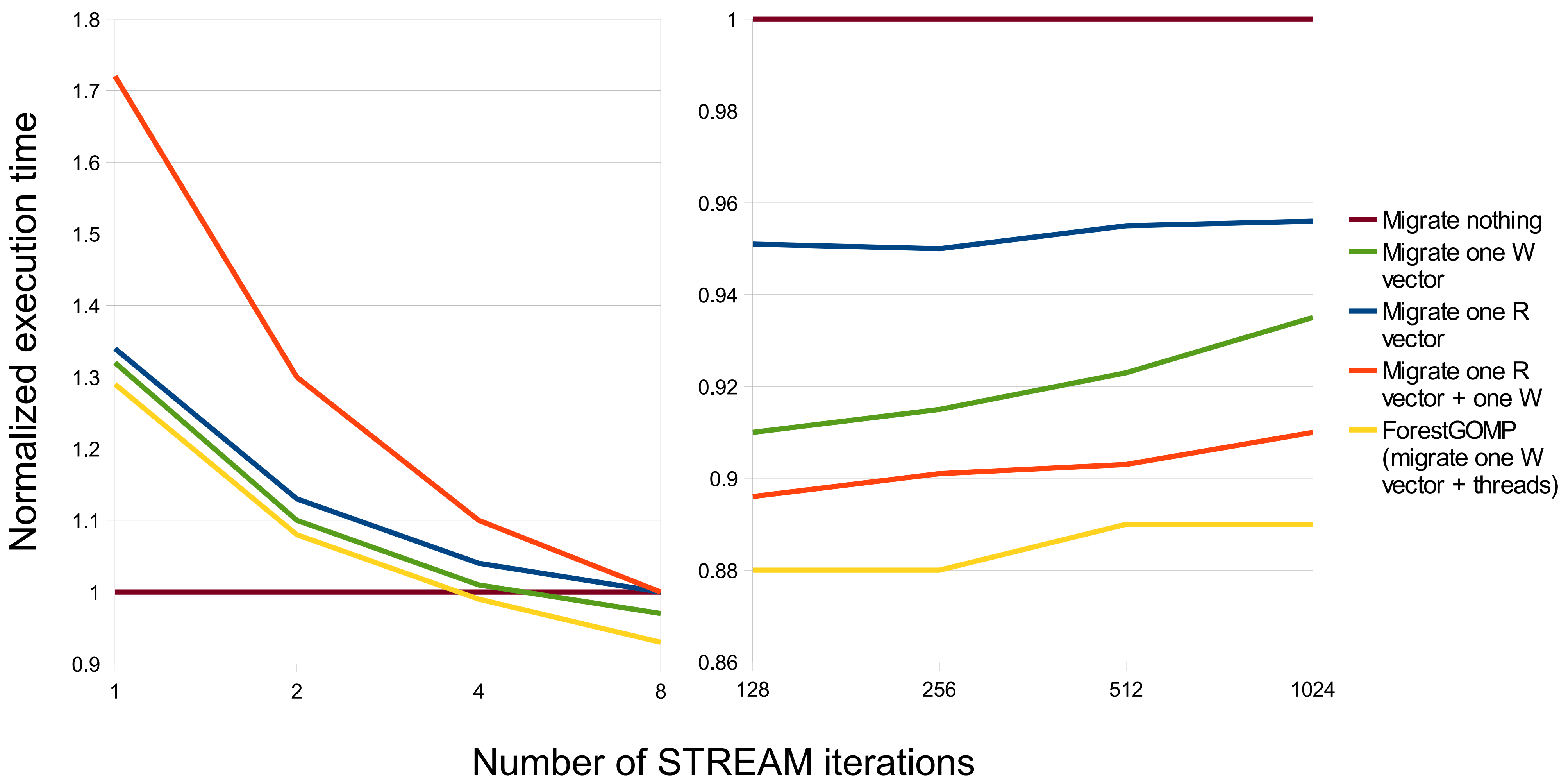
Compiler	Phase 1	Phase 2
GCC 4.2	7.5 ± 1	4.2 ± 1
Intel ICC 10.1	8 ± 1	5.4 ± 1
ForestGOMP	8.9 ± 0.5	5.9 ± 0.5

Twisted-100 results (GB/s) on a quad-socket quad-core Opteron computer

Twisted-100 Results



Twisted-66 Results



Summary

- Memory affinity does matter !
  - Take memory affinity into account = improve performance
    - Less distant accesses
    - Less memory contention
  - Let the programmer provide memory affinities...
  - … but don't ask him to know all about the architecture!
  - Put the runtime in charge of what to/where to migrate
    - Threads? Data? Sometimes both!
  - Can behave better than next-touch based approaches
- Available inside the ForestGOMP platform !
  - Check out the new ForestGOMP release including the Memory Bubble Scheduler and the programming interface to express affinity relations !
  - "man run-forest" for information about how to run a ForestGOMP application
  - http://runtime.futurs.inria.fr/forestgomp/index.php

Future work

- Extraction of information
  - An OpenMP extension to express memory affinity
    - copy_in/copy_out can help on NUMA computers too!
  - Feedback about hardware-related statistics

- Get the world NUMA-aware !
  - Compose schedulers with the Memory Bubble Scheduler
    - Cache Bubble Scheduler... done!
    - OpenMP 3.0 tasks
    - TBB
    - Accelarator-aware scheduler