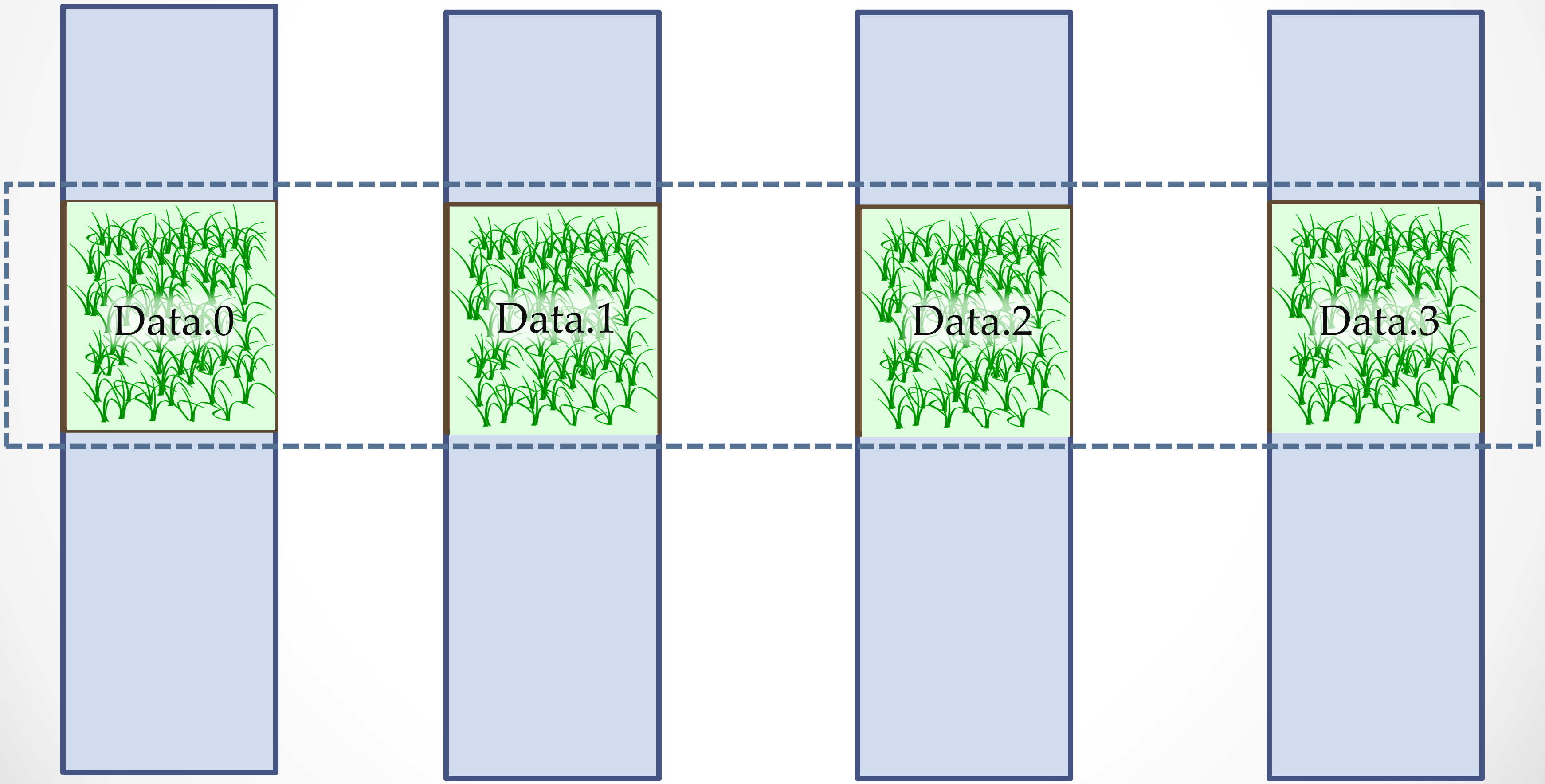
Do we need a crystal ball for task migration?

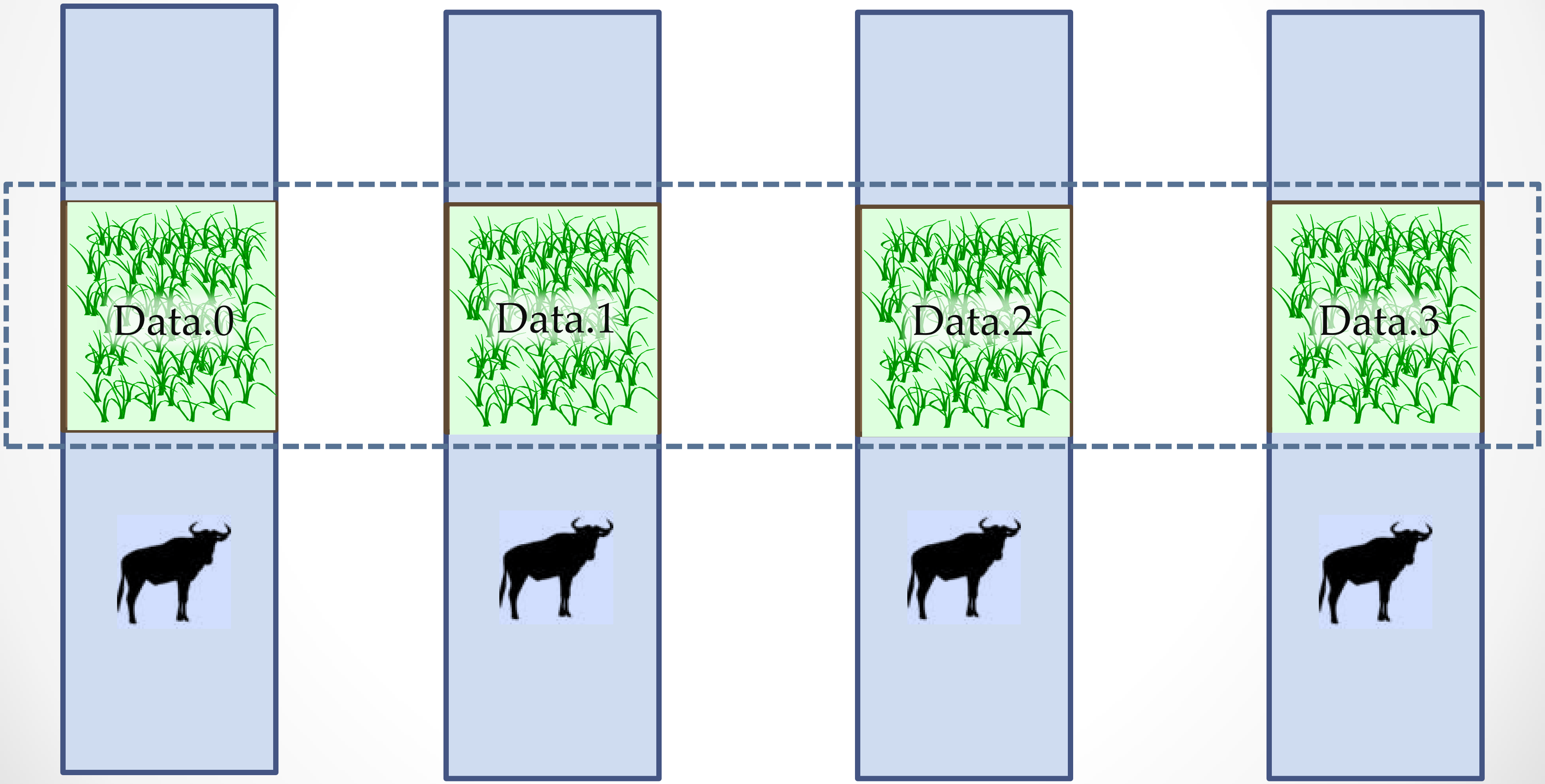
Brandon {Myers,Holt}
University of Washington
bdmyers@cs.washington.edu

Large data sets

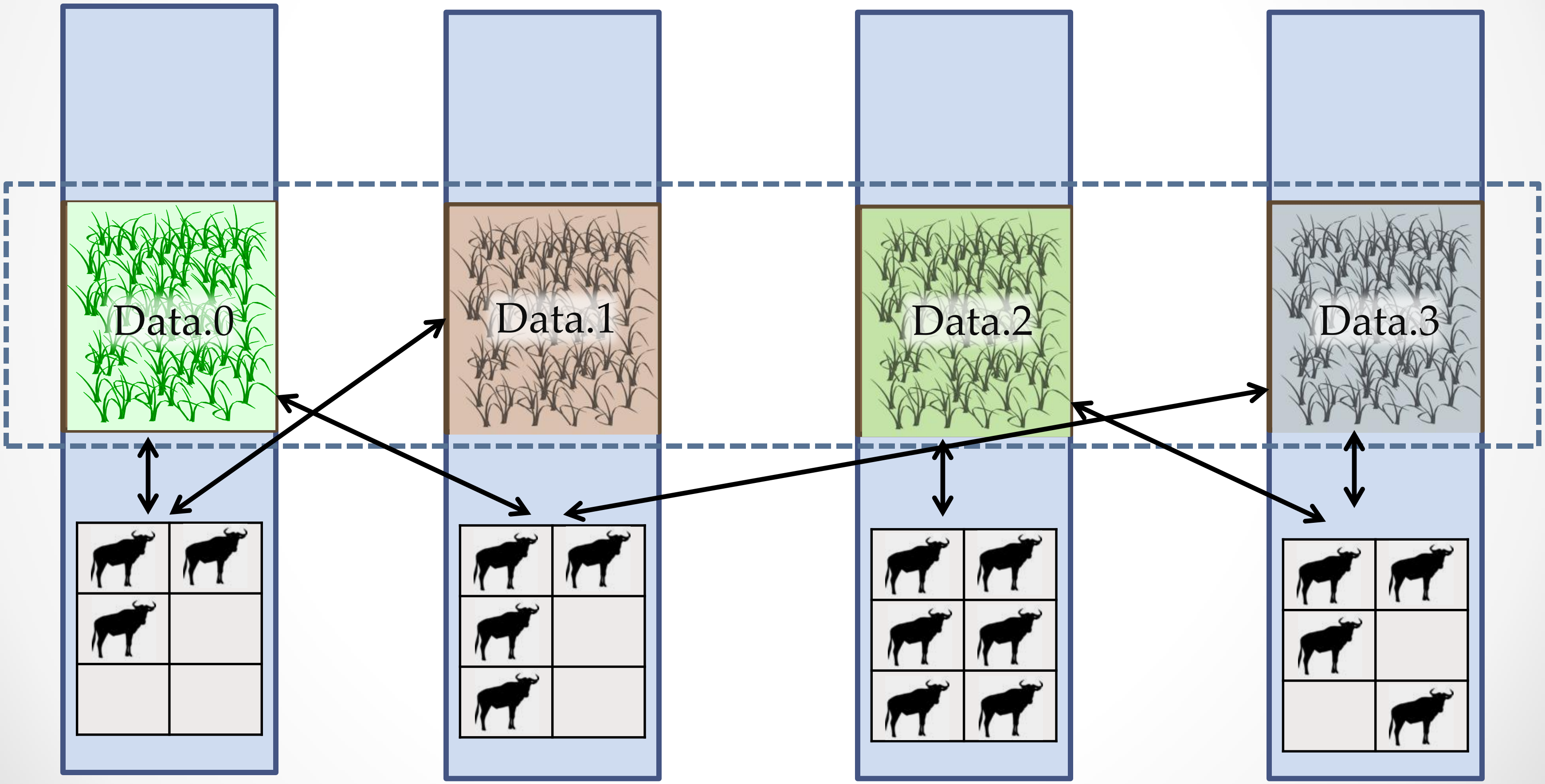
Spread data



Spread data



Resources: compute, bandwidth



Task migration

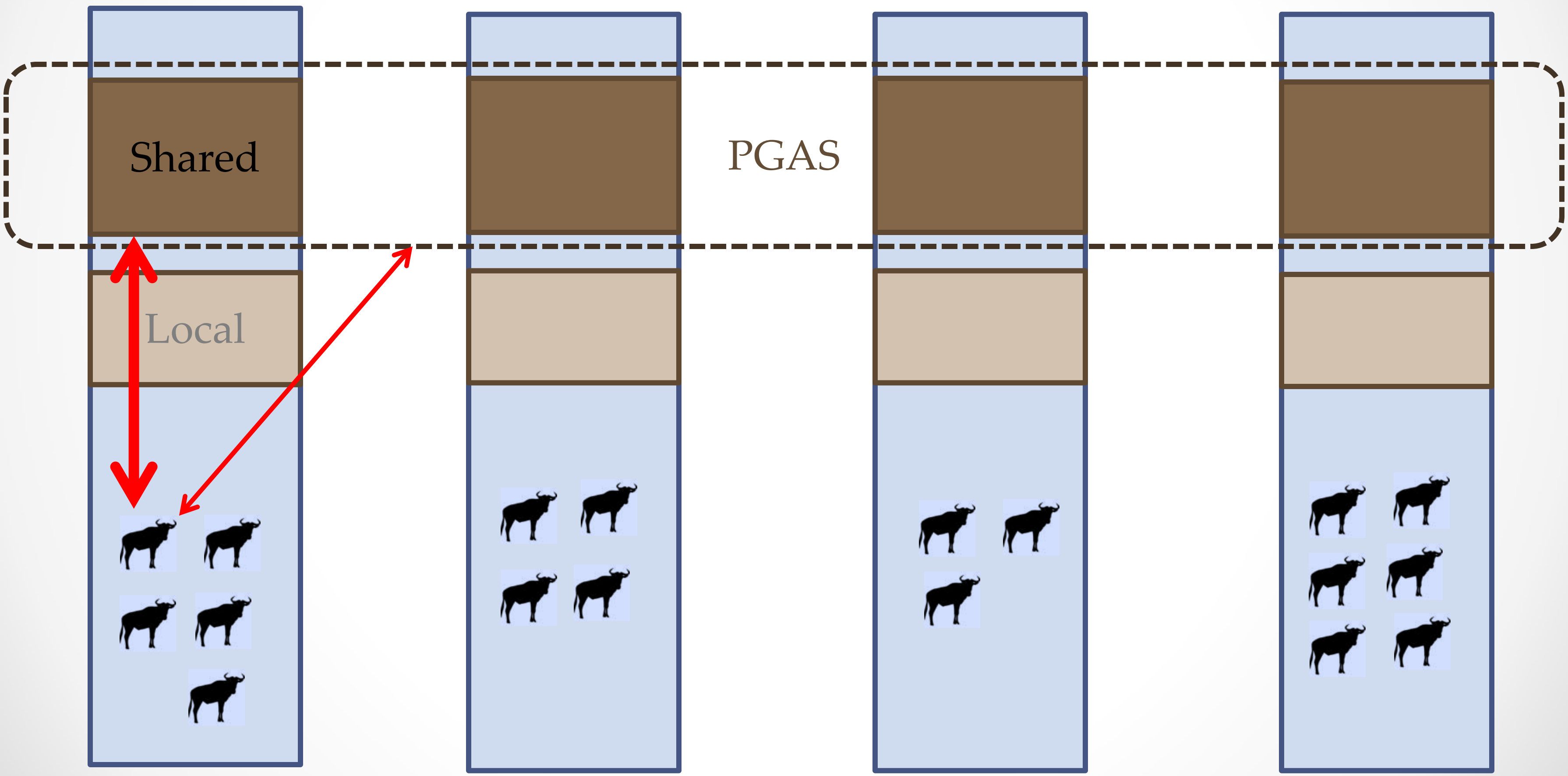
- move a running task to another node
- purpose:
  - increase utilization or manage resources
  - move task near tasks that share data
  - move task closer to data it will access
- costs:
  - moving local data required for the task to proceed
  - cpu time to stop and resume a task

Prior work

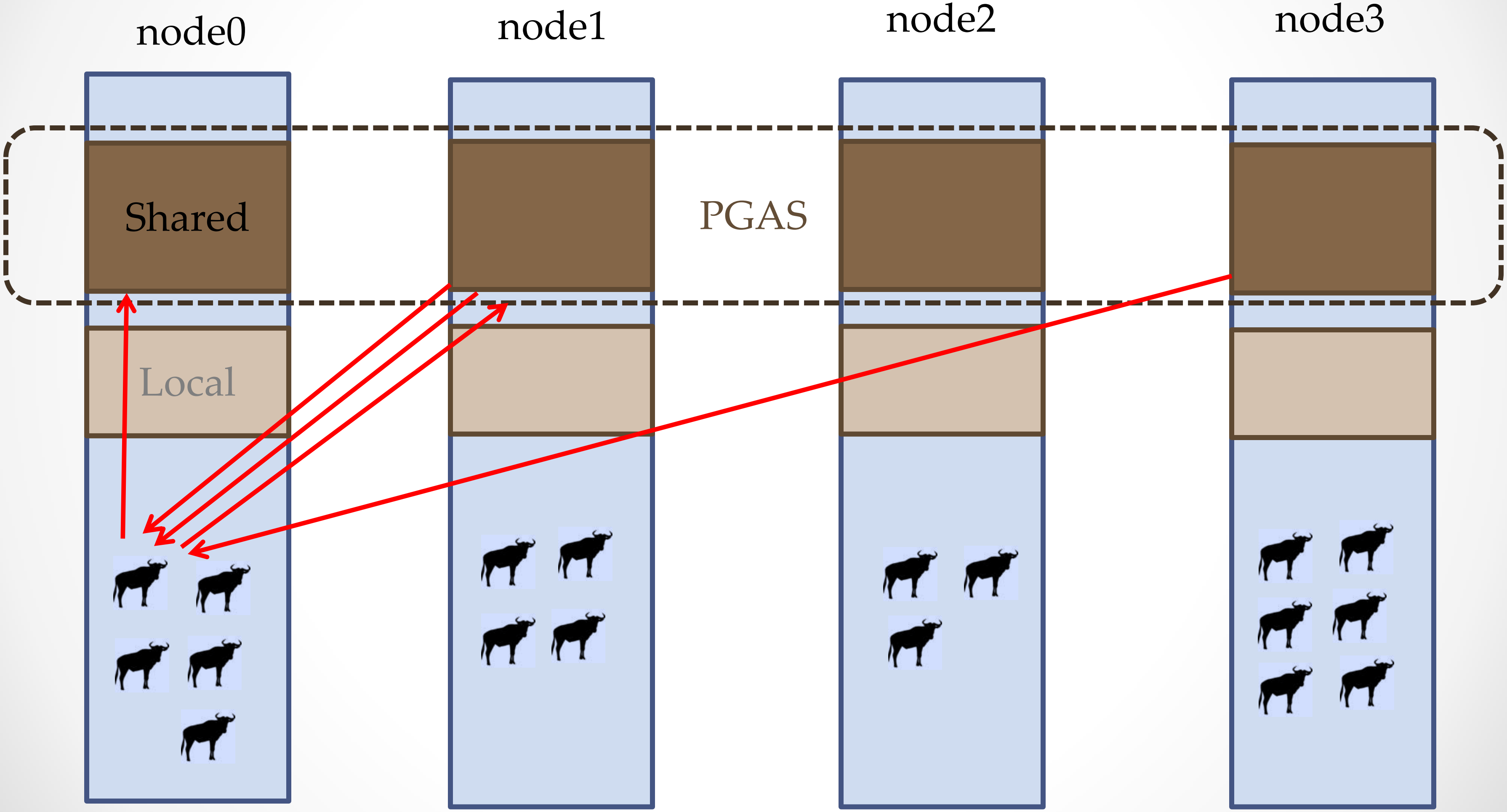
- task migration for:
  - efficient use of resources
  - load balancing
- thread placement on cache coherent systems using sharing information[1]
- prediction for migration on NoC[2]

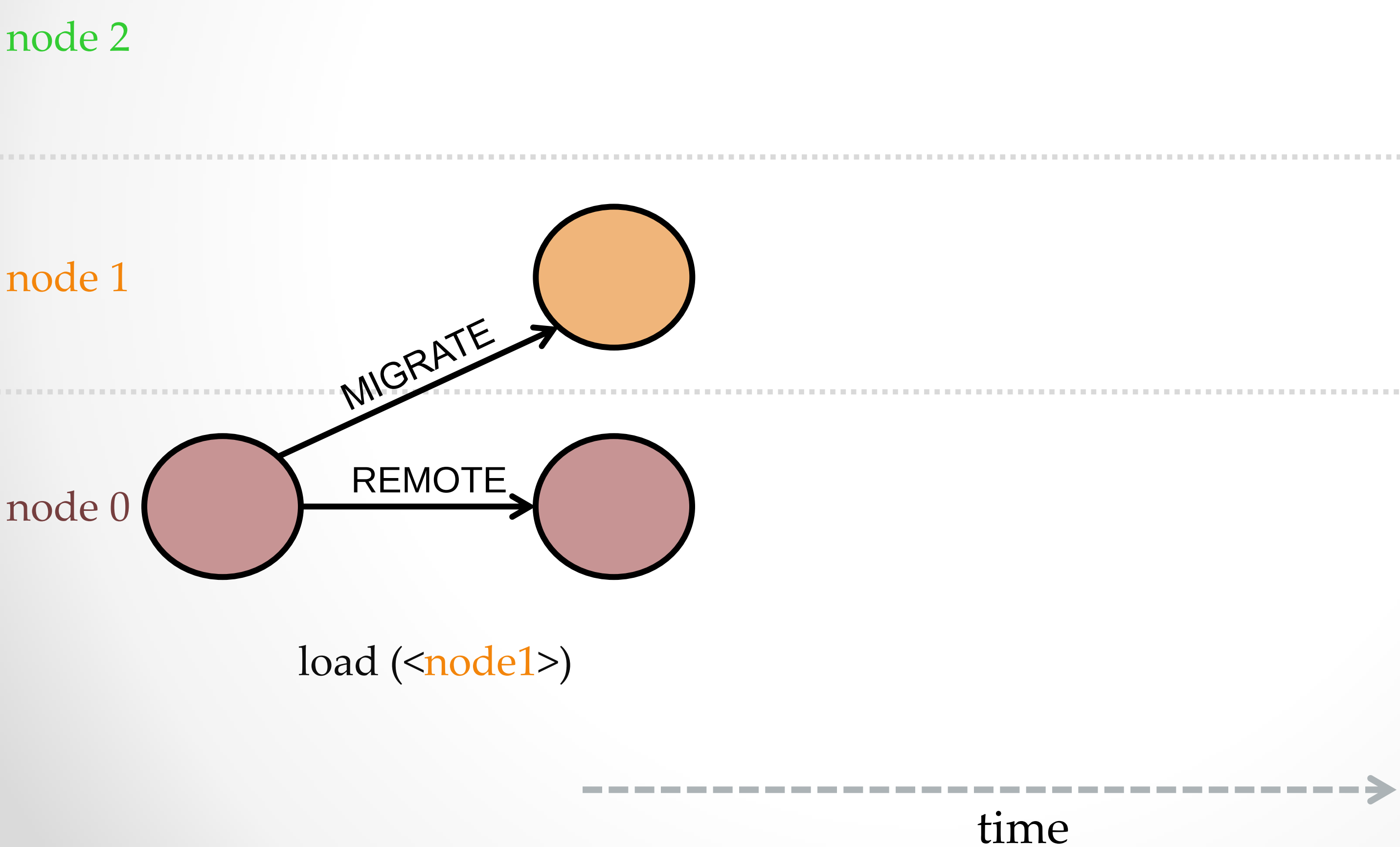
1. F. Song et al. *Analytical modeling and optimization for affinity based thread scheduling on multicore systems*. CLUSTER '09.
2. Chao Wang et al. *Packet Triggered Prediction Based Task Migration for Network-on-Chip*. 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing, Feb ‘12

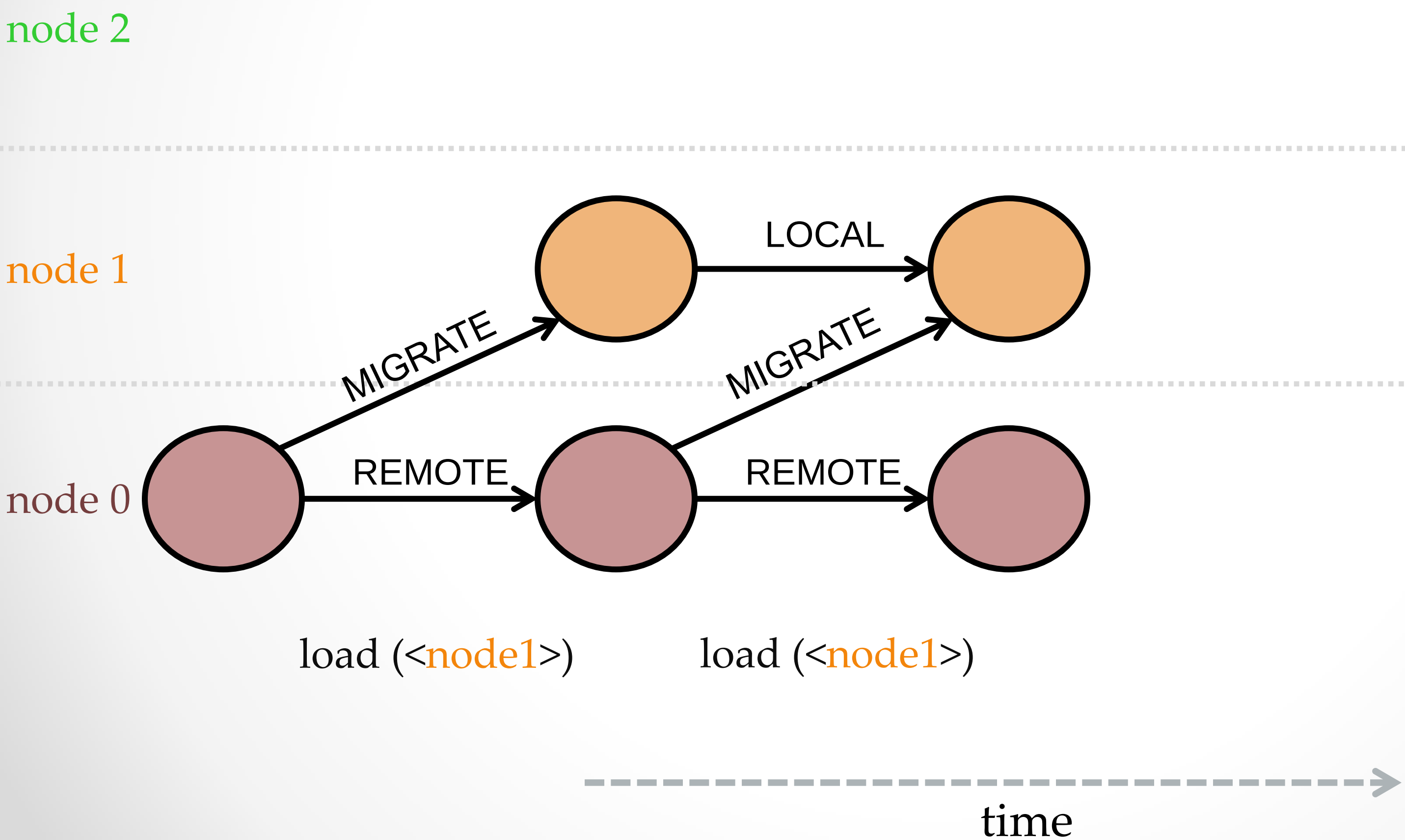
Non-uniform cost to access shared data

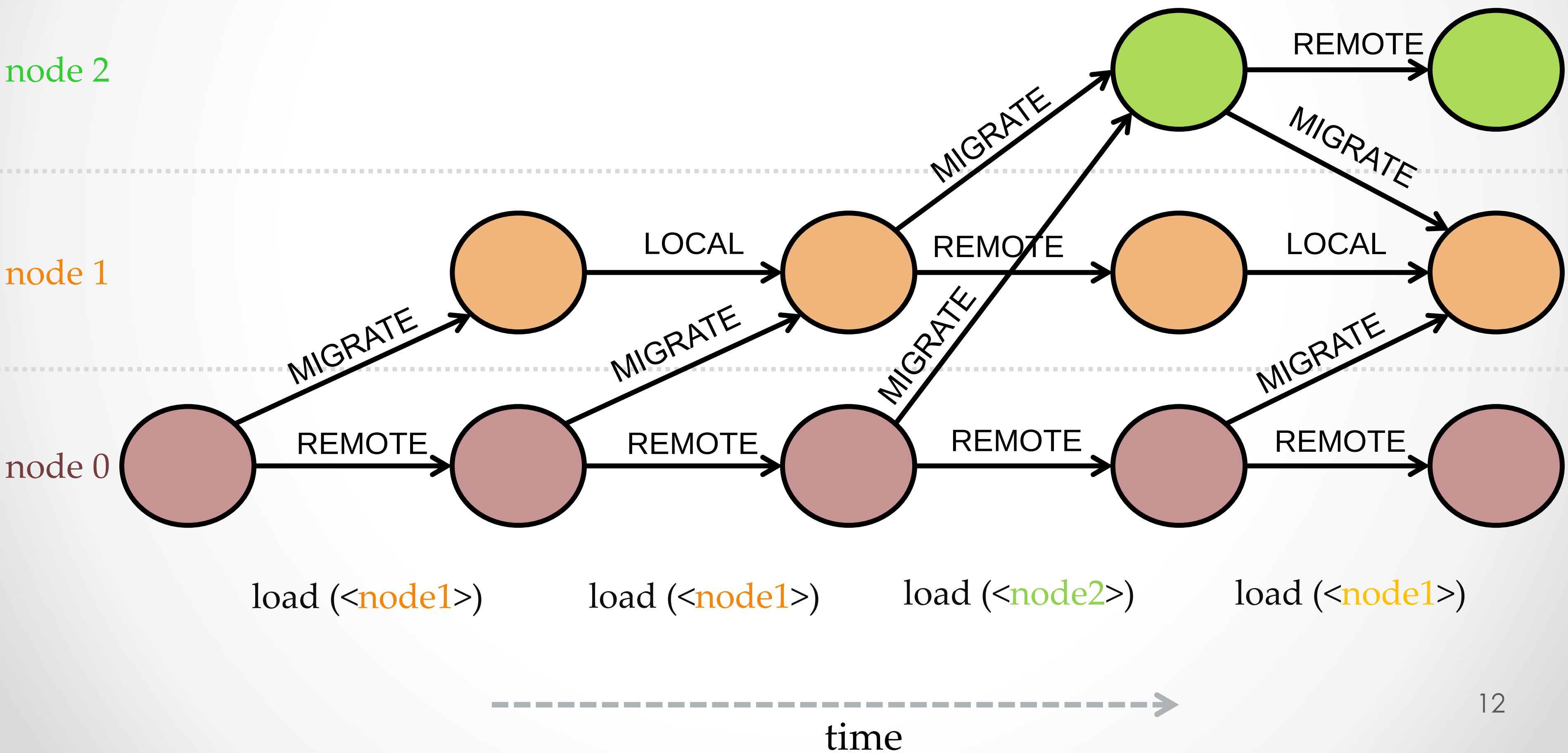


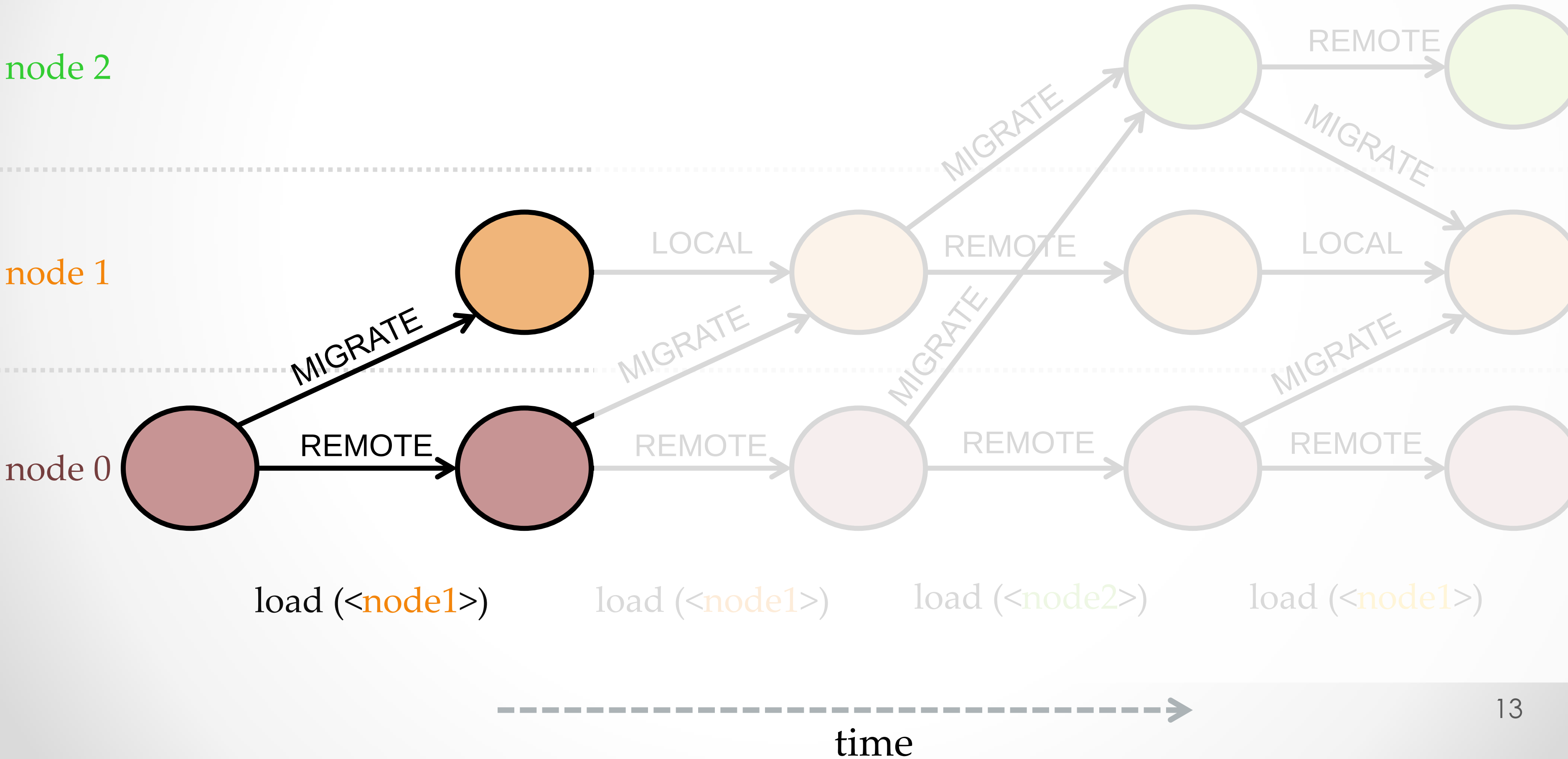
Exploit locality




node 2
node 1
MIGRATE
REMOTE
node 0
load (<node1>)
time


node 2
node 1
node 0
MIGRATE
LOCAL
MIGRATE
REMOTE
REMOTE
load (<node1>)
load (<node1>)
time


node 2
node 1
node 0
REMOTE
MIGRATE
LOCAL
REMOTE
MIGRATE
MIGRATE
LOCAL
REMOTE
MIGRATE
MIGRATE
REMOTE
REMOTE
REMOTE
MIGRATE
REMOTE
load (<node1>)
load (<node1>)
load (<node2>)
load (<node1>)
time


node 2
node 1
node 0
MIGRATE
REMOTE
LOCAL
MIGRATE
REMOTE
MIGRATE
REMOTE
MIGRATE
REMOTE
REMOTE
MIGRATE
LOCAL
MIGRATE
REMOTE
load (<node1>)
load (<node1>)
load (<node2>)
load (<node1>)
time

Question

- consider task migration as a prediction problem

- can we predict when it will be more efficient
  to move the data to the task,
  or move the task to the data?

Outline

- Motivation
- System model and cost metric
- Online migration predictors
- Evaluation

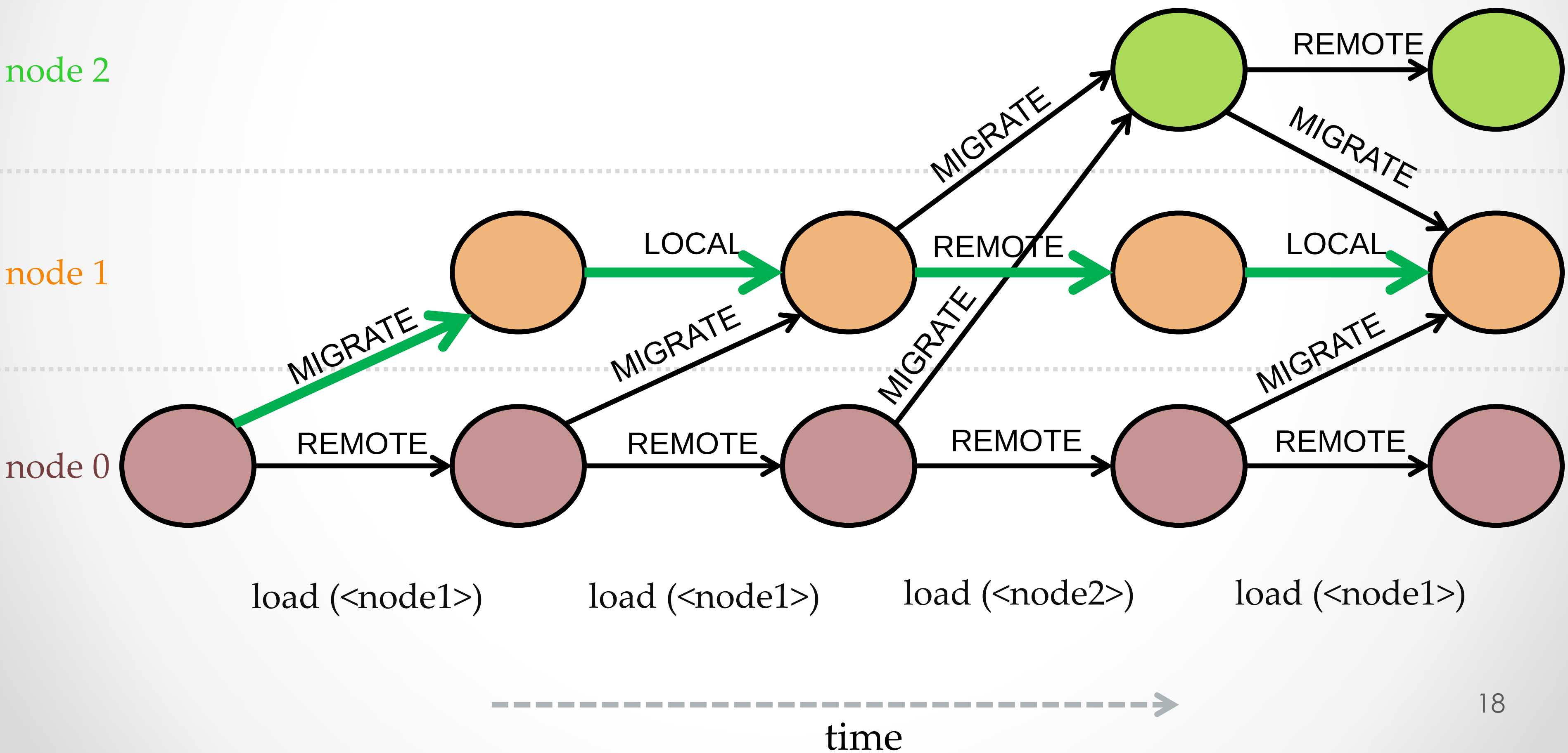
System model

- assumption: network is limiting resource
- simplification: flat network topology
  - only distinguish between local and remote shared memory
- cost metric: bytes transferred over the network
  - others are possible; this is enough to capture network usage
  - no timing required

Optimal task migration

- What is the best possible cost for a given execution?
- Find the schedule of migrations that minimizes bytes transferred
- Model excludes timing => schedules can be calculated independently for each task

Optimal schedule



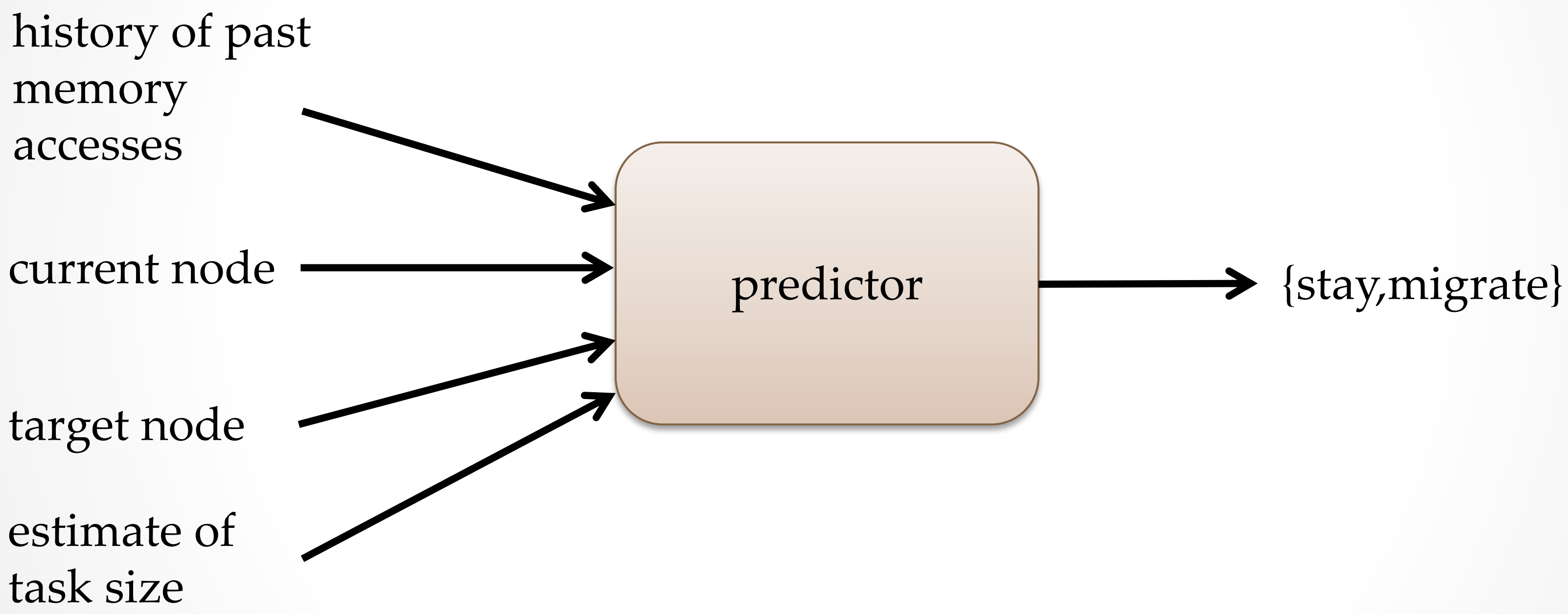
Outline

- Motivation
- System model and cost metric
- Online migration policies
- Evaluation

Online policies

- predict whether a migration will give benefit
- look at past access patterns
- similar to prefetch prediction in computer architecture

Migration predictors



Stream Predictor policy

- influenced by *stream buffer** prefetch prediction
- migrate task when it has seen 'enough' references to the same node in the immediate past

PC

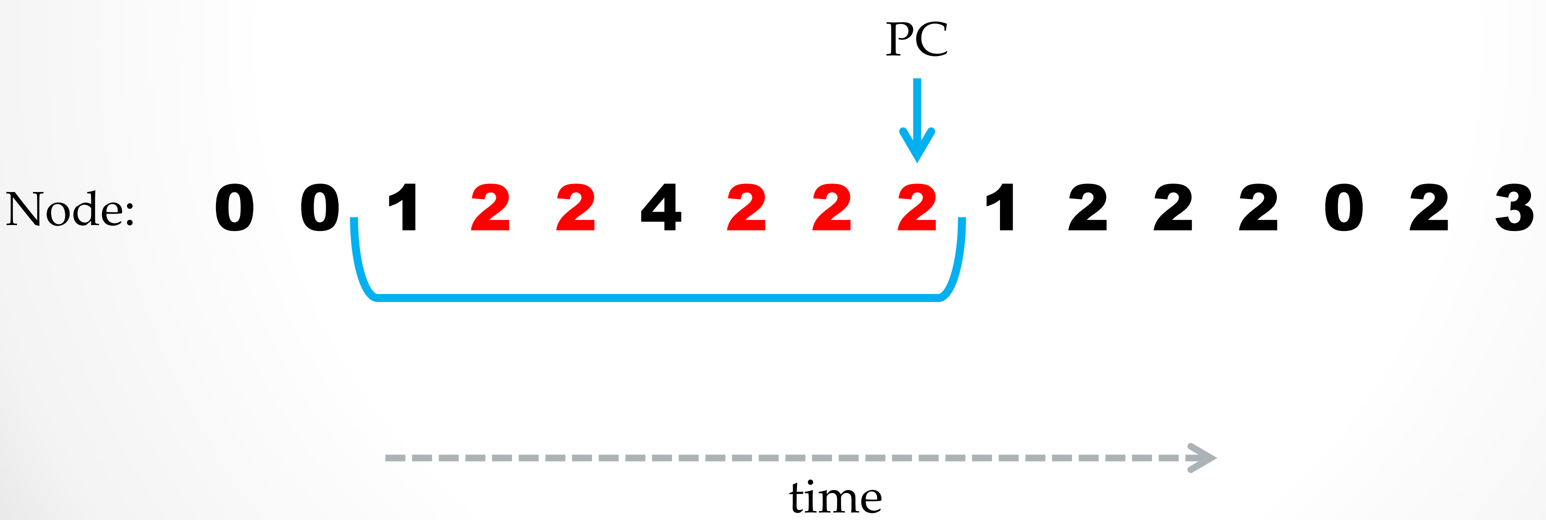
Node: 0 0 1 2 2 4 2 2 2 1 2 2 2 0 2 3

time

*N. P. Jouppi. *Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers*. 17th ISCA '90, pages 364{373, New York, NY, USA, 1990. ACM.

Stream Predictor policy

- disadvantages of Stream:
  - do extra remote accesses before recognizing pattern
  - must do this every time



Hindsight Migrate policy

- insight:
  - same code may always have the same access pattern
- solution:
  - remember PCs that would have been good to migrate at

Hindsight: motivation

```
1.      shared arrays[][];
2.      for particleArray in arrays:
3.          totalWeight = 0
4.          for p in particleArray:
5.              totalWeight += p.weight
6.          histogram[totalWeight]++
```

PC	2	4	5	…	5	6.a	6.b	2	4	5	…	5	6.a	6.b
Node	0	3	3	3…	3	1	1	0	7	7	7…	7	4	4

Hindsight Migrate policy

PC

PC	0	1	2	3	4	5	6	7	8	9	4	5	6	7	8	9	4	5
Node	0	2	0	0	1	1	1	3	1	1	2	2	2	8	2	2	3	3

Migration set

Hindsight Migrate policy

PC

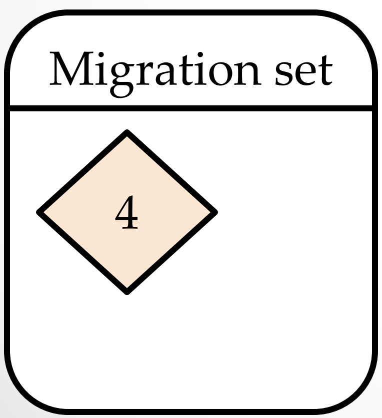
PC	0	1	2	3	4	5	6	7	8	9	4	5	6	7	8	9	4	5
Node	0	2	0	0	**1**	**1**	**1**	3	**1**	**1**	2	2	2	8	2	2	3	3

Migration set

Hindsight Migrate policy

PC

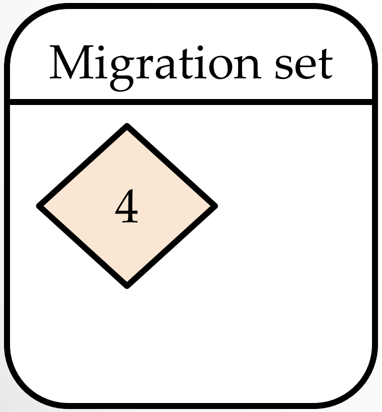
PC	0	1	2	3	4	5	6	7	8	9	4	5	6	7	8	9	4	5
Node	0	2	0	0	**1**	**1**	**1**	3	**1**	**1**	2	2	2	8	2	2	3	3



Hindsight Migrate policy

PC

PC	0	1	2	3	4	5	6	7	8	9	4	5	6	7	8	9	4	5
Node	0	2	0	0	**1**	**1**	**1**	3	**1**	**1**	2	2	2	8	2	2	3	3



Hindsight Migrate policy

PC

PC	0	1	2	3	4	5	6	7	8	9	4	5	6	7	8	9	4	5
Node	0	2	0	0	1	1	1	3	1	1	2	2	2	8	2	2	3	3

Migration set

4

migrate? yes

Outline

- Motivation
- Simplified system model and cost metric
- Online migration policies
- Evaluation

Evaluation

- potential for task migration over no migration?
- how much of this can predictors achieve?

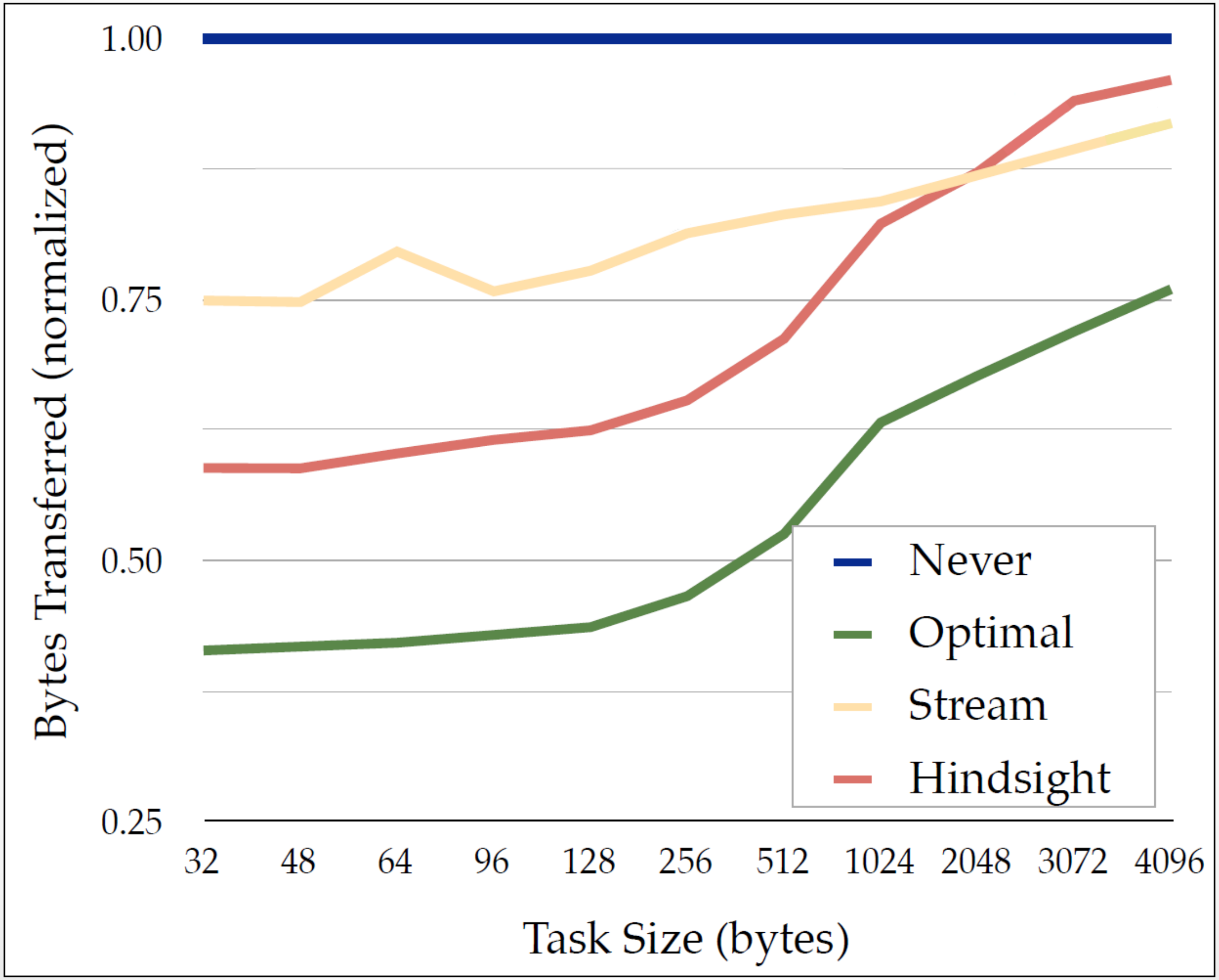
- procedure:
  - collect shared memory trace from program execution
  - simulate it in our model and measure total cost
  - run simulations with fixed task sizes

- benchmarks
  - NPB IntSort
  - PARSEC FluidAnimate
  - SSCA#2 Betweenness centrality

Simulation

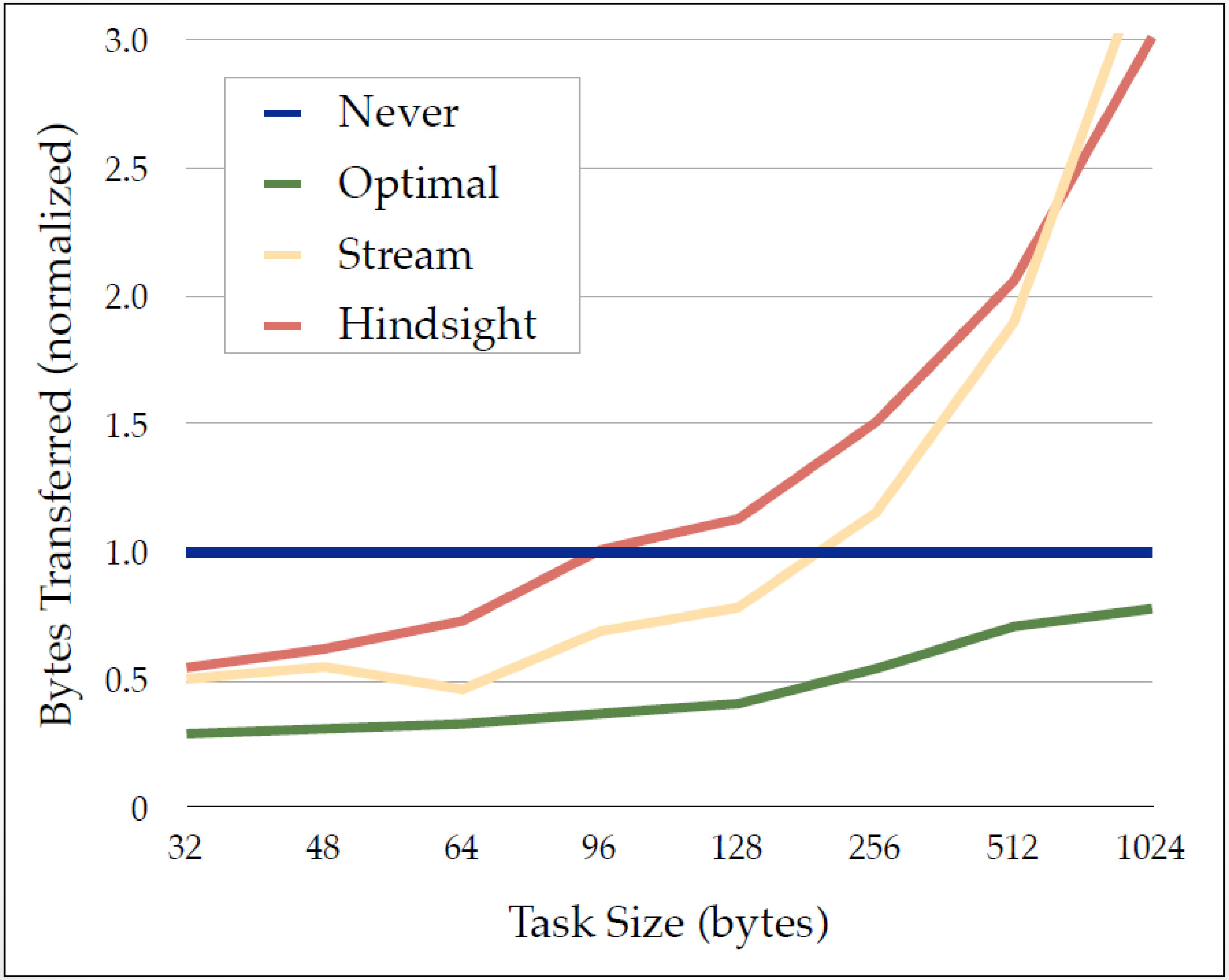
1. annotate application code to choose a distribution for each shared memory allocation
2. collect shared memory trace for an execution
3. simulate:
   1. at each memory access, ask the policy whether the task should migrate
   2. add the cost of the chosen action

IntSort

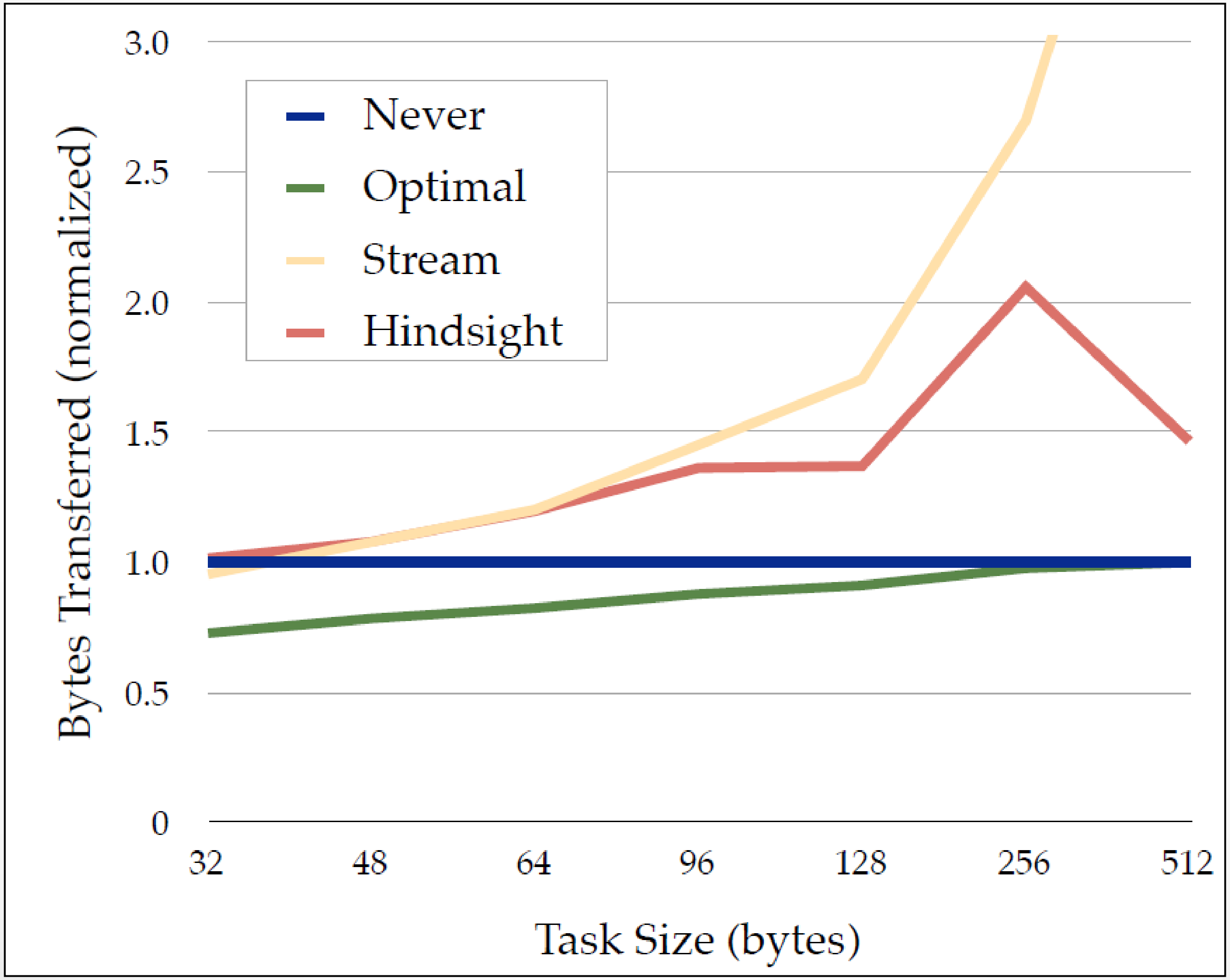


1.00
0.75
0.50
0.25
Bytes Transferred (normalized)
32
48
64
96
128
256
512
1024
2048
3072
4096
Task Size (bytes)
Never
Optimal
Stream
Hindsight

FluidAnimate



Betweenness Centrality



Results summary

- simple online predictors achieved up to 60% of optimal reduction in bytes transferred
- higher ratio of random access => lower potential for task migration to reduce network usage

Conclusion

- In this work:
  - task migration for reducing network usage, considered as a prediction problem
- Take-away:
  - migration predictors can make profitable choices based on past memory accesses
  - moving tasks to the data has the potential to improve performance of parallel applications if there is locality to exploit

A better cost metric

- message cost = $\frac{size}{BW(size)}$

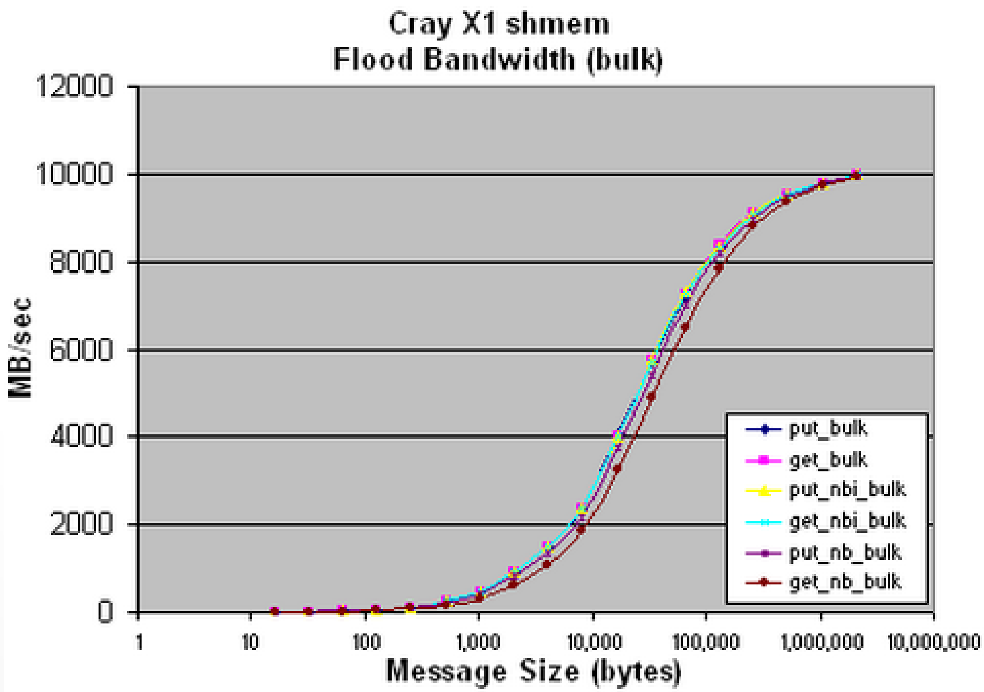
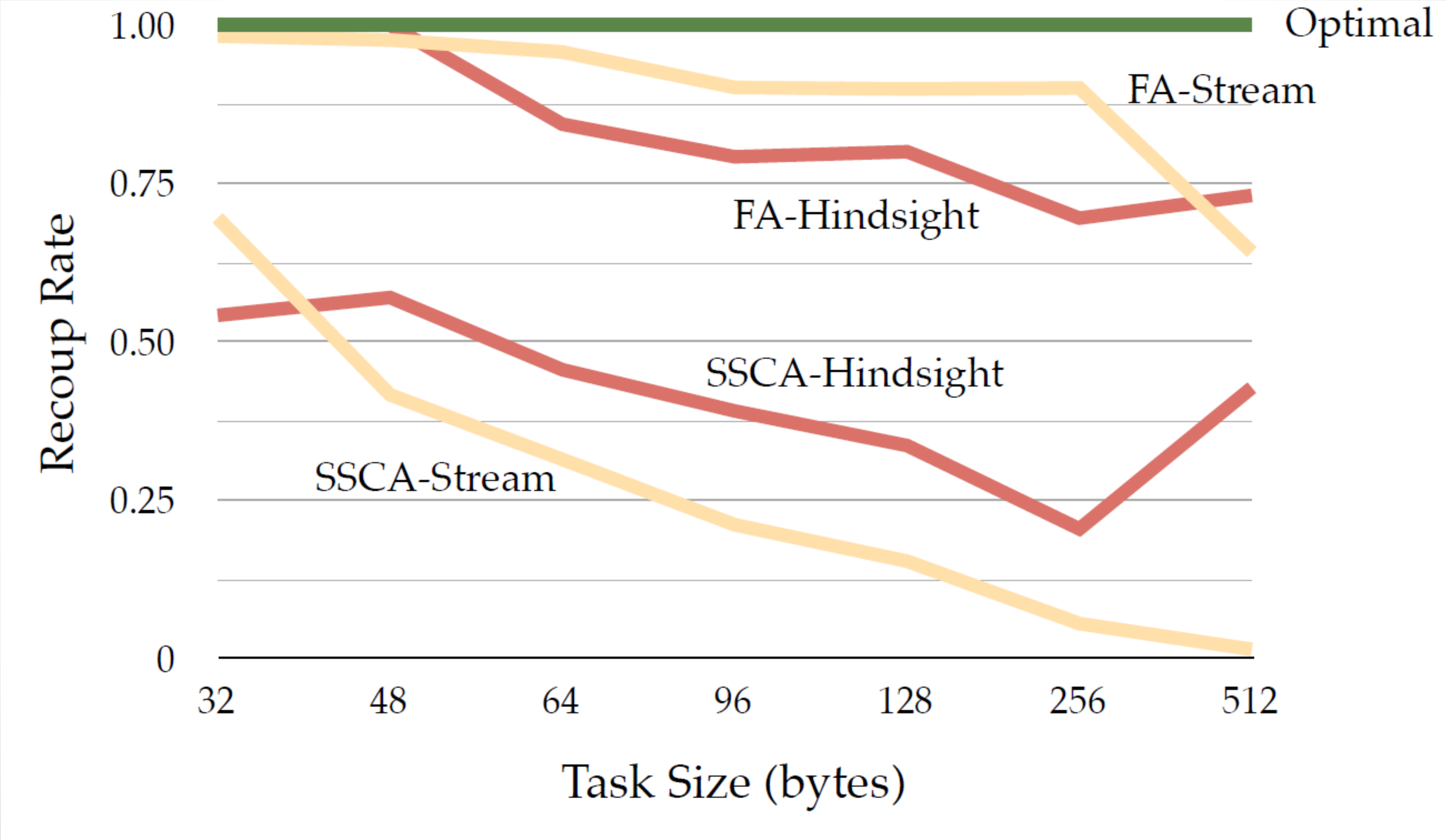


image: http://gasnet.cs.berkeley.edu/performance/

“Recoup rate”



1.00
0.75
0.50
0.25
0
Recoup Rate
Optimal
FA-Stream
FA-Hindsight
SSCA-Hindsight
SSCA-Stream
32
48
64
96
128
256
512
Task Size (bytes)

Annotations

```
edgeData = (graphSDG *) malloc(sizeof(graphSDG));
track_memory(edgeData->startVertex, M, sizeof(VERT_T), BLOCK);
track_memory(edgeData->endVertex, M, sizeof(VERT_T), BLOCK);
track_memory(edgeData->weight, M, sizeof(WEIGHT_T), BLOCK);
BC = (double *) tm_malloc(N , sizeof(double), BLOCK);
elapsed_time = betweennessCentrality(G, BC);
tm_free(BC);
```

Instrumentation

- Use Pin to instrument the tracking functions and memory accesses
- On tracking functions
    - Update mapping of (address range) -> (allocation id)
- On each memory access
    - the callback looks up the access
    - If it is in a tracked region, save information about the access to trace file