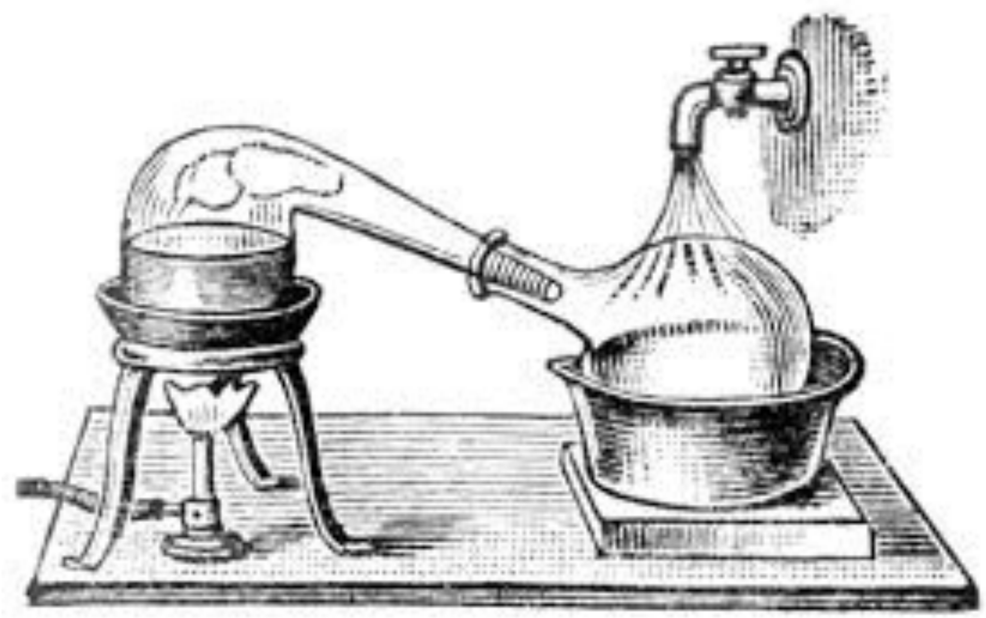
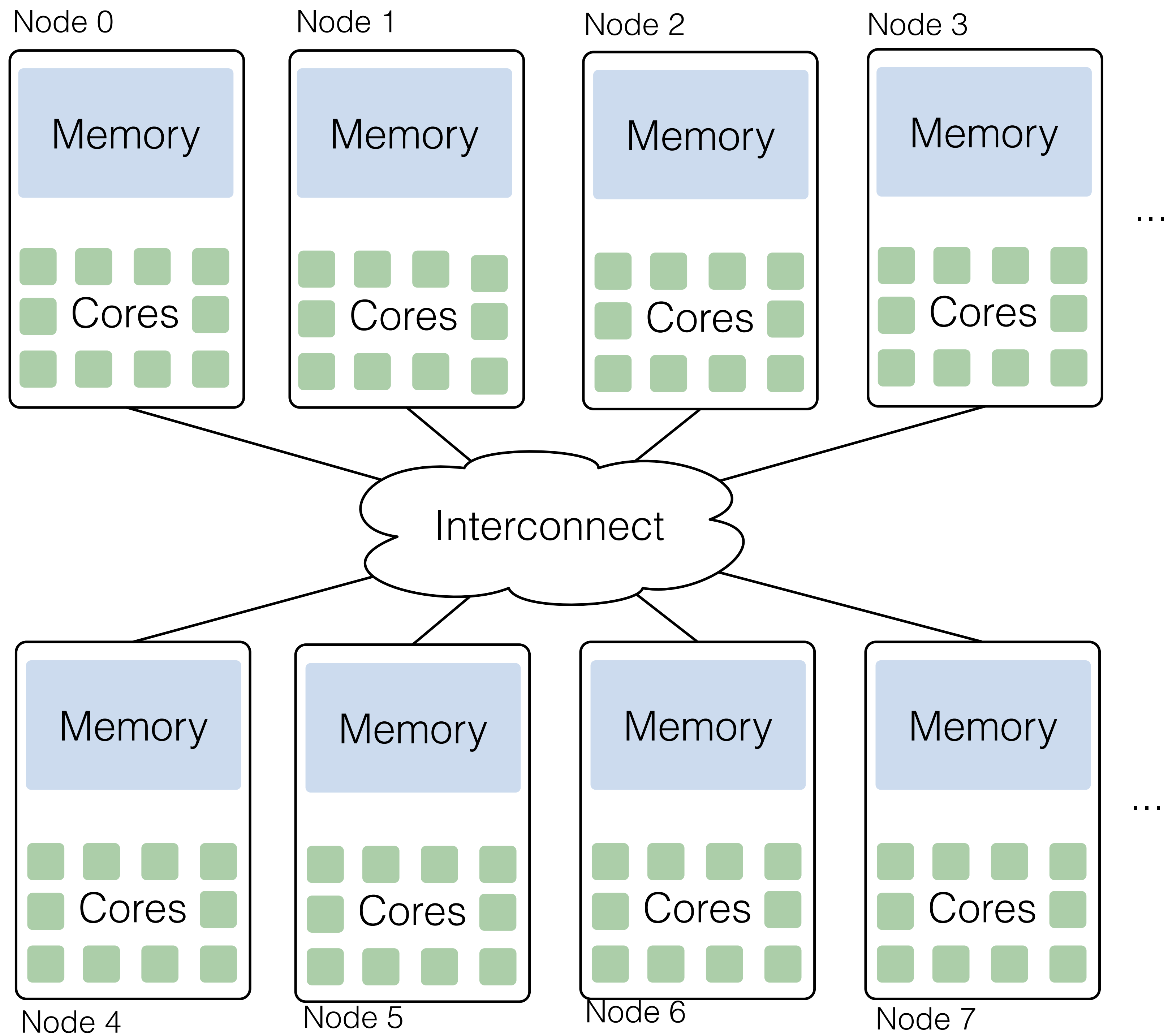
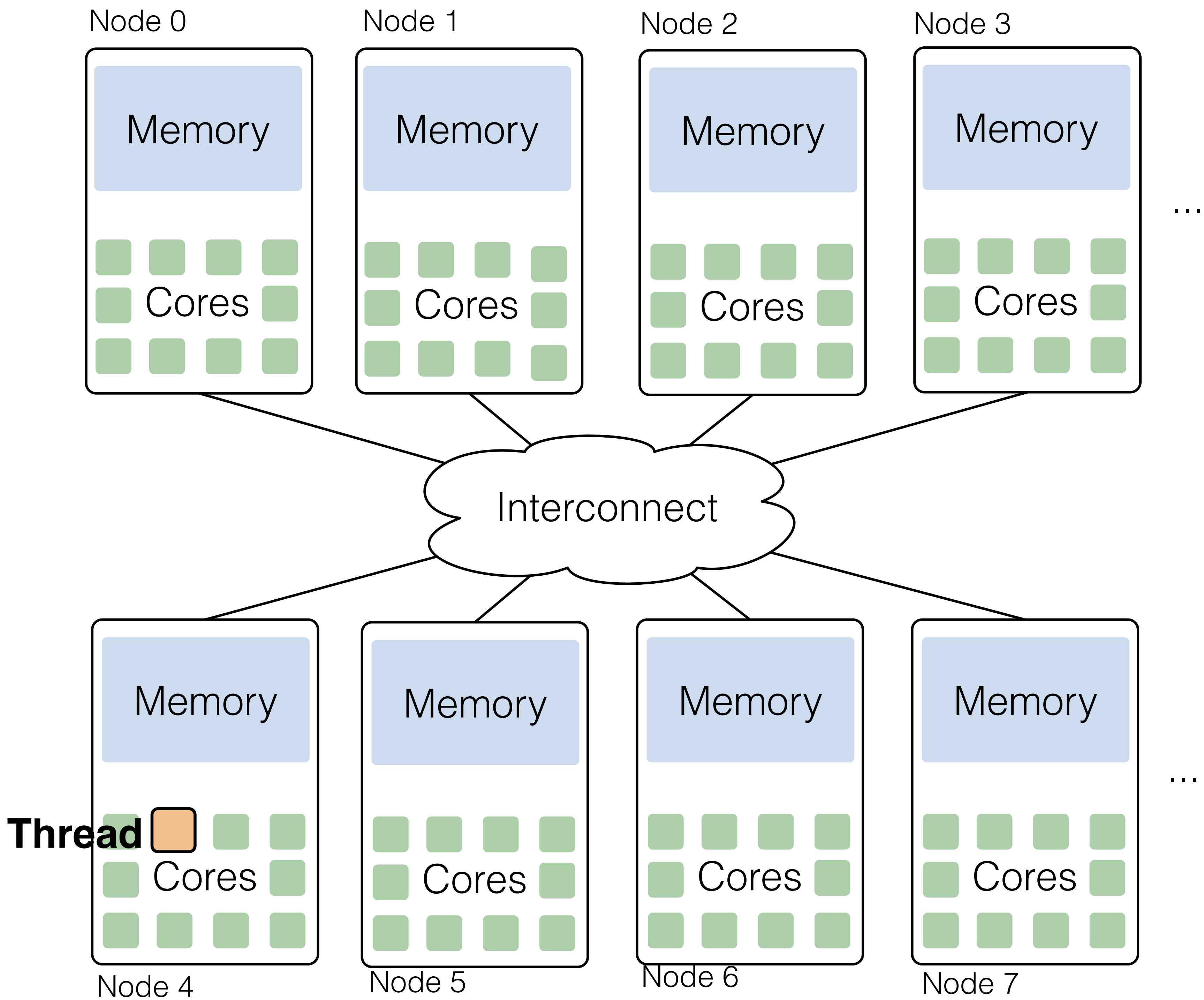


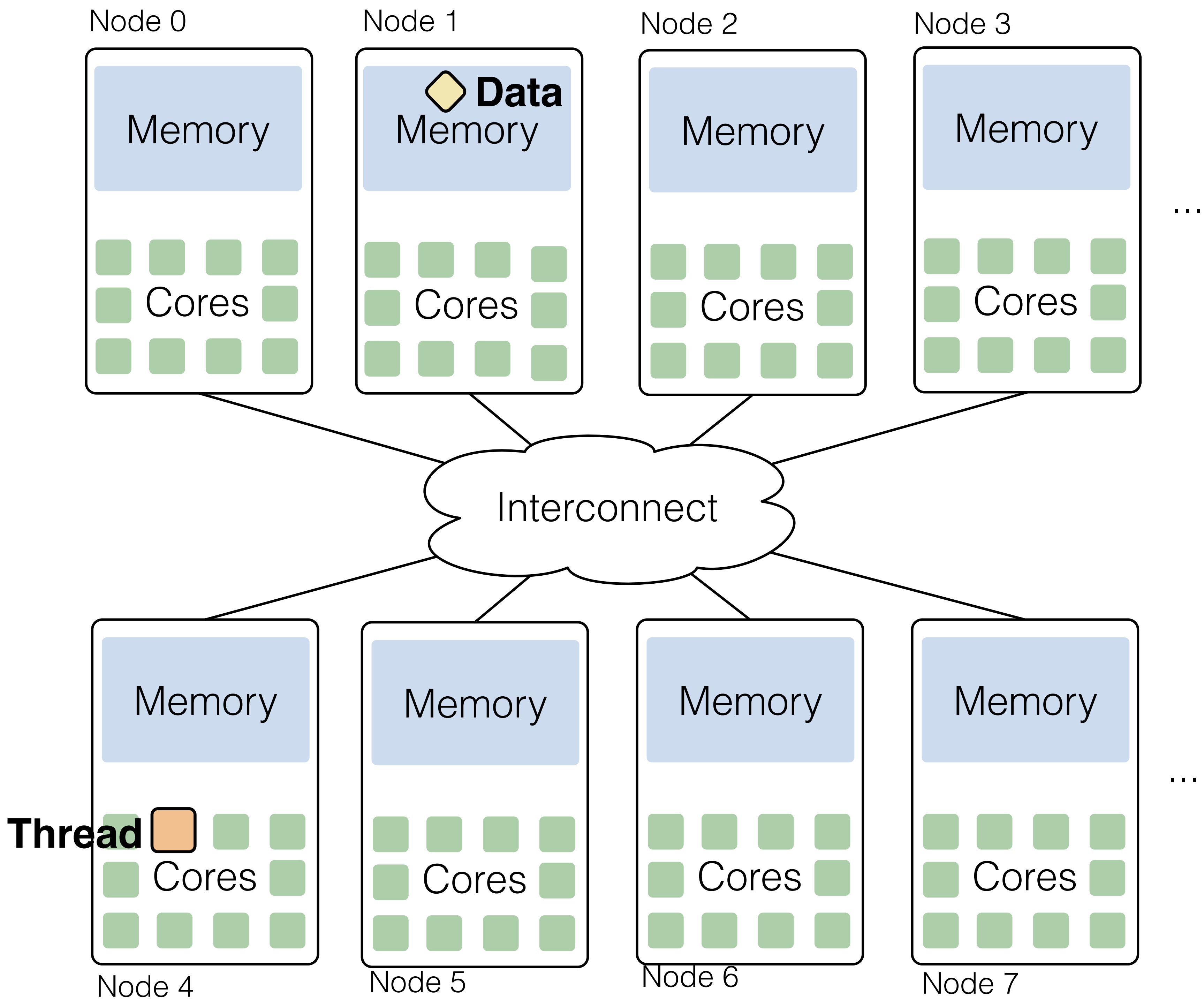
Alembic

Automatic Locality Extraction via Migration

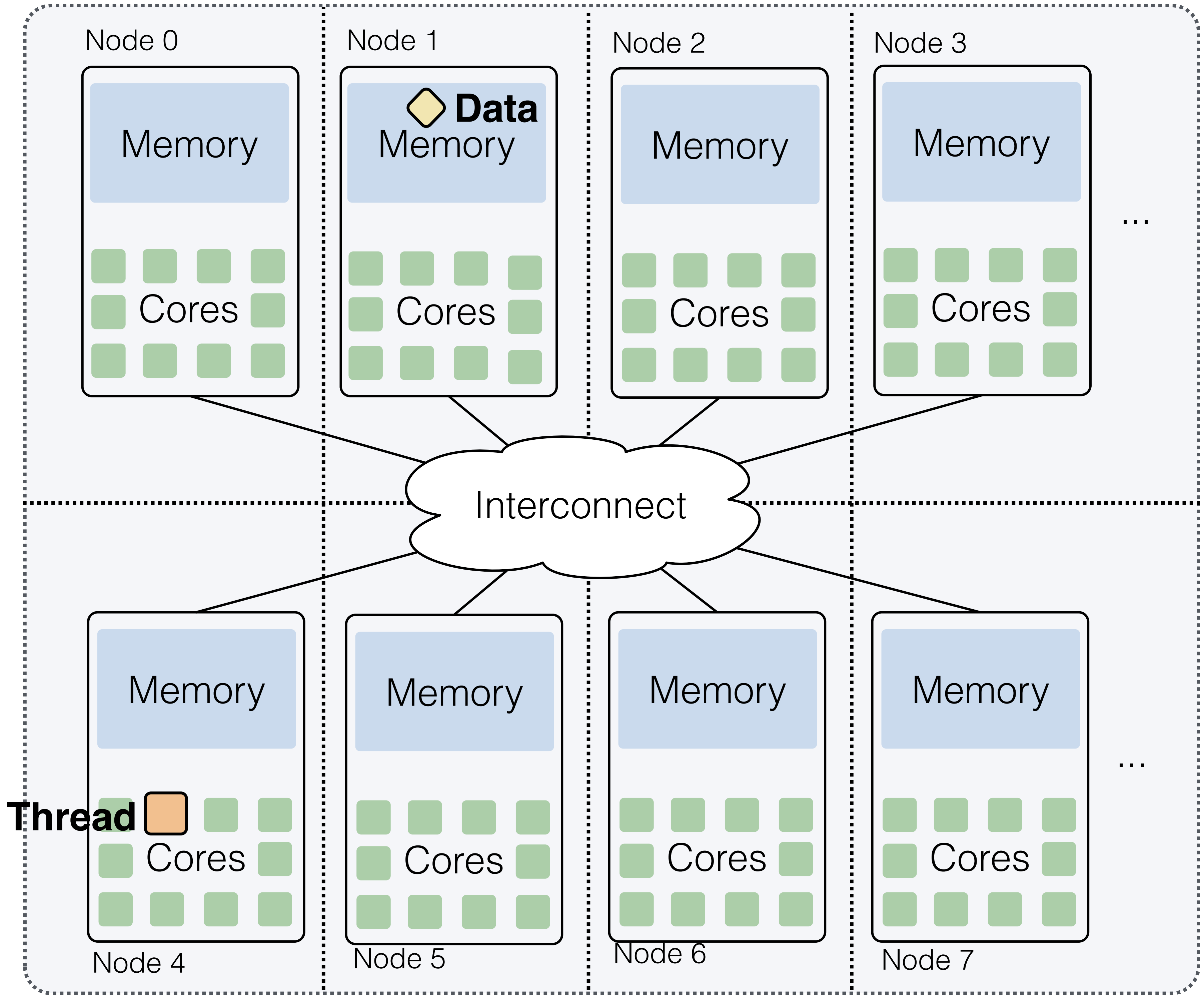
Brandon Holt, Preston Briggs, Luis Ceze, Mark Oskin


Node 0
Node 1
Node 2
Node 3
Memory
Cores
...
Interconnect
Node 4
Node 5
Node 6
Node 7

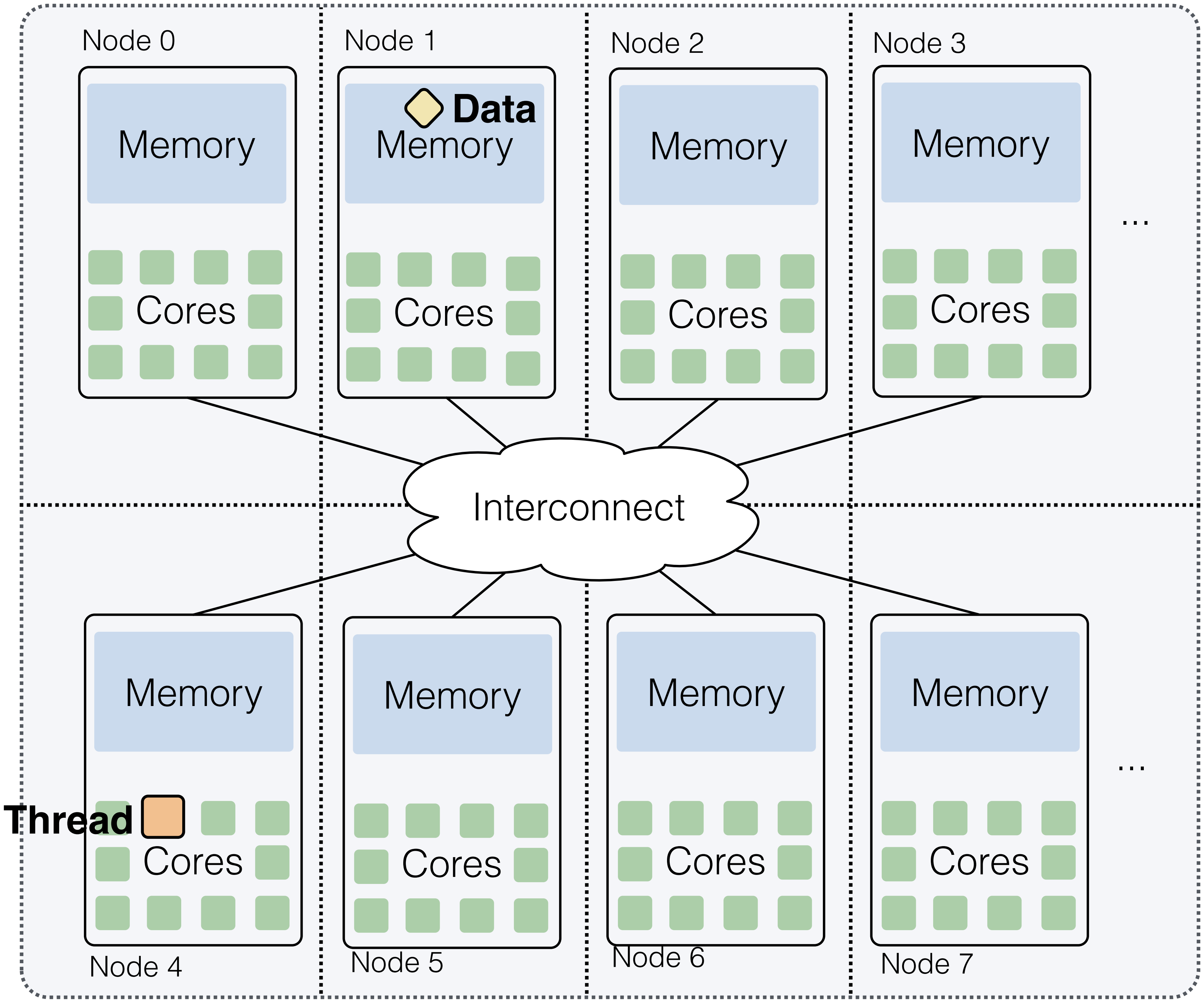

Node 0
Node 1
Node 2
Node 3
Memory
Cores
...
Interconnect
Thread
Node 4
Node 5
Node 6
Node 7


Node 0
Node 1
Node 2
Node 3
Data
Memory
Cores
...
Interconnect
Thread
Node 4
Node 5
Node 6
Node 7

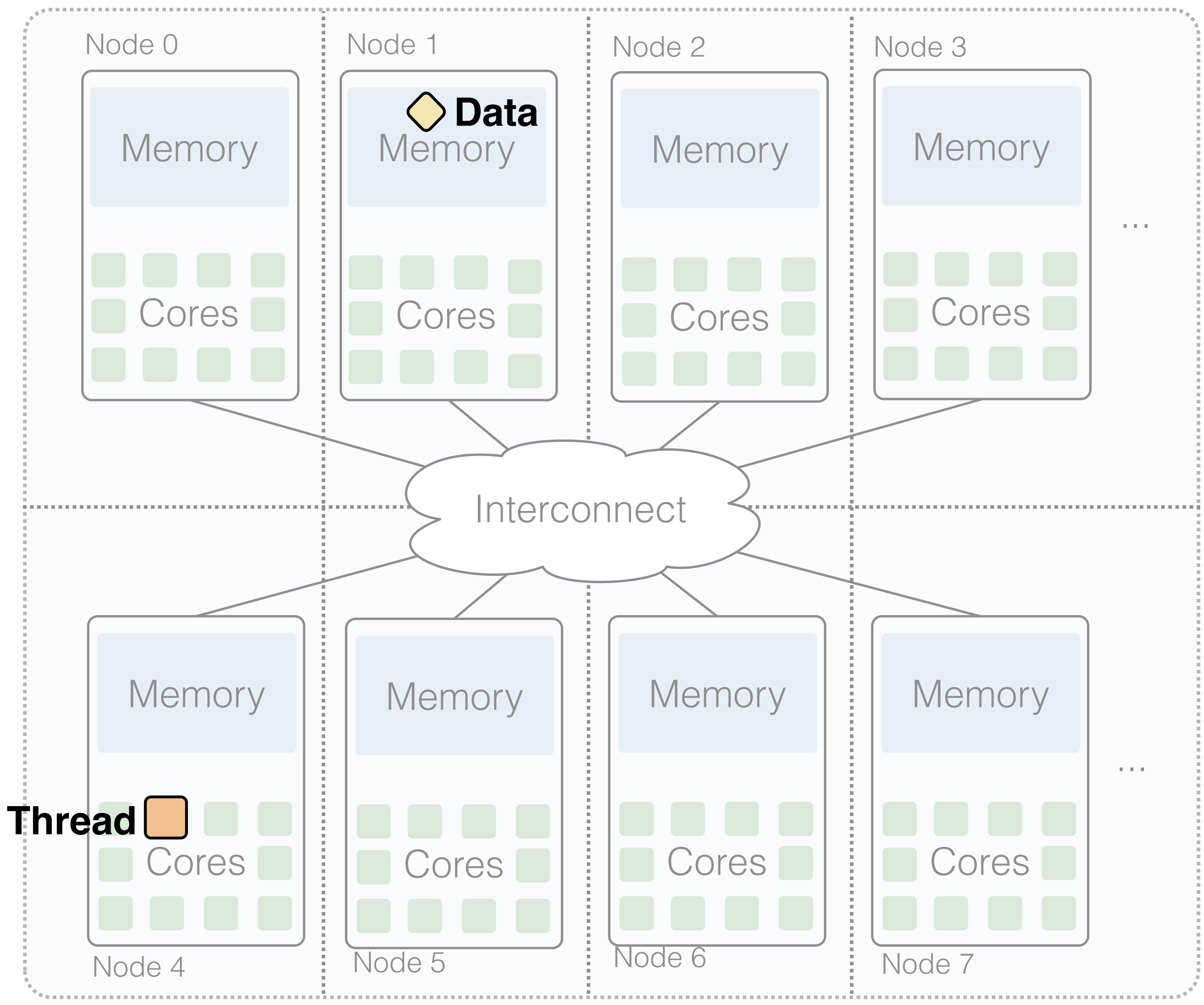
Partitioned Global Address Space (PGAS)



Partitioned Global Address Space (PGAS)



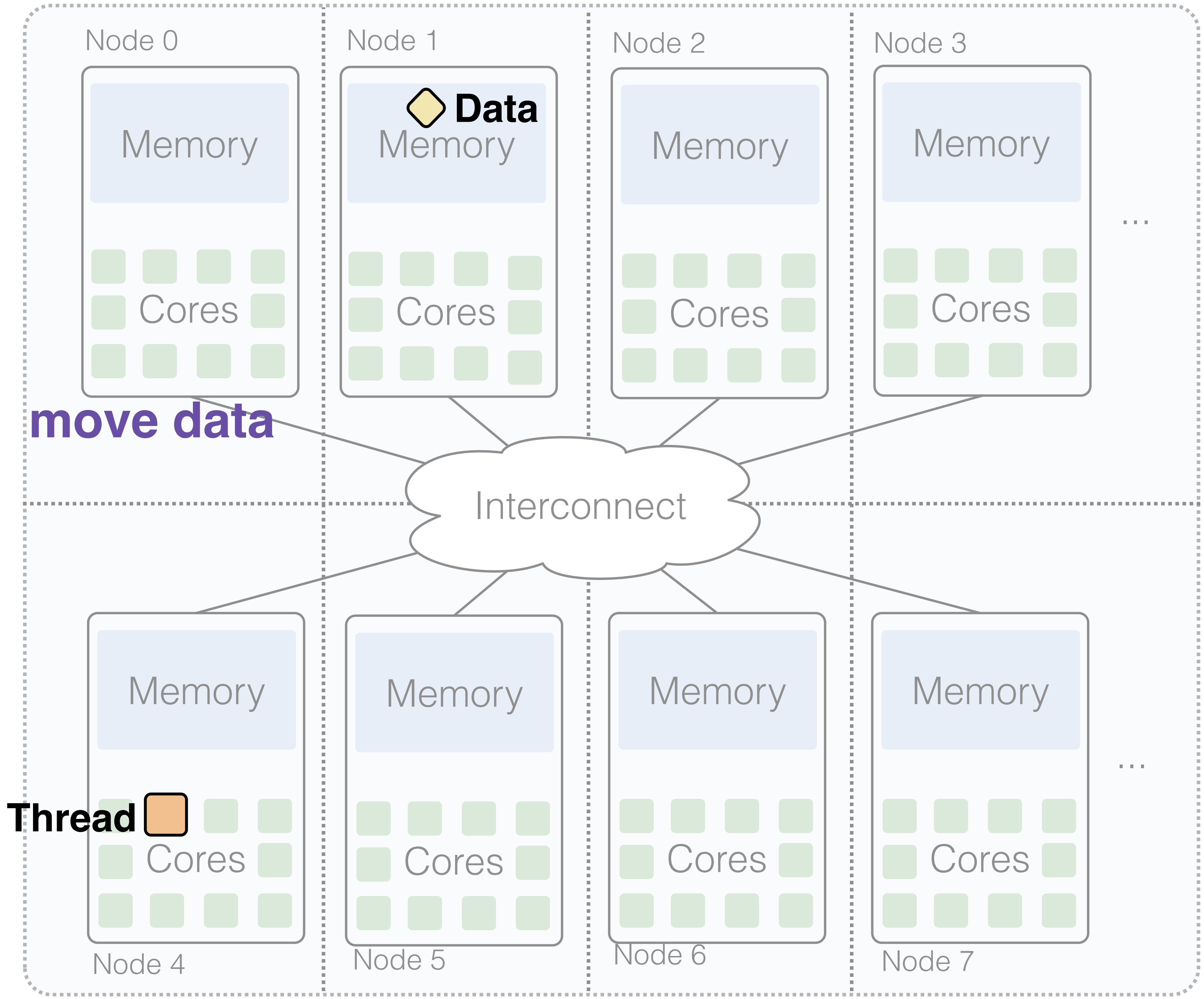
Move data or computation?



* M. C. Carlisle and A. Rogers. Software caching and computation migration in Olden. In *PPOPP '95*, ACM.

† L. V. Kale and S. Krishnan. CHARM++: A portable concurrent object oriented system based on C++. *OOPSLA '93*. ACM.

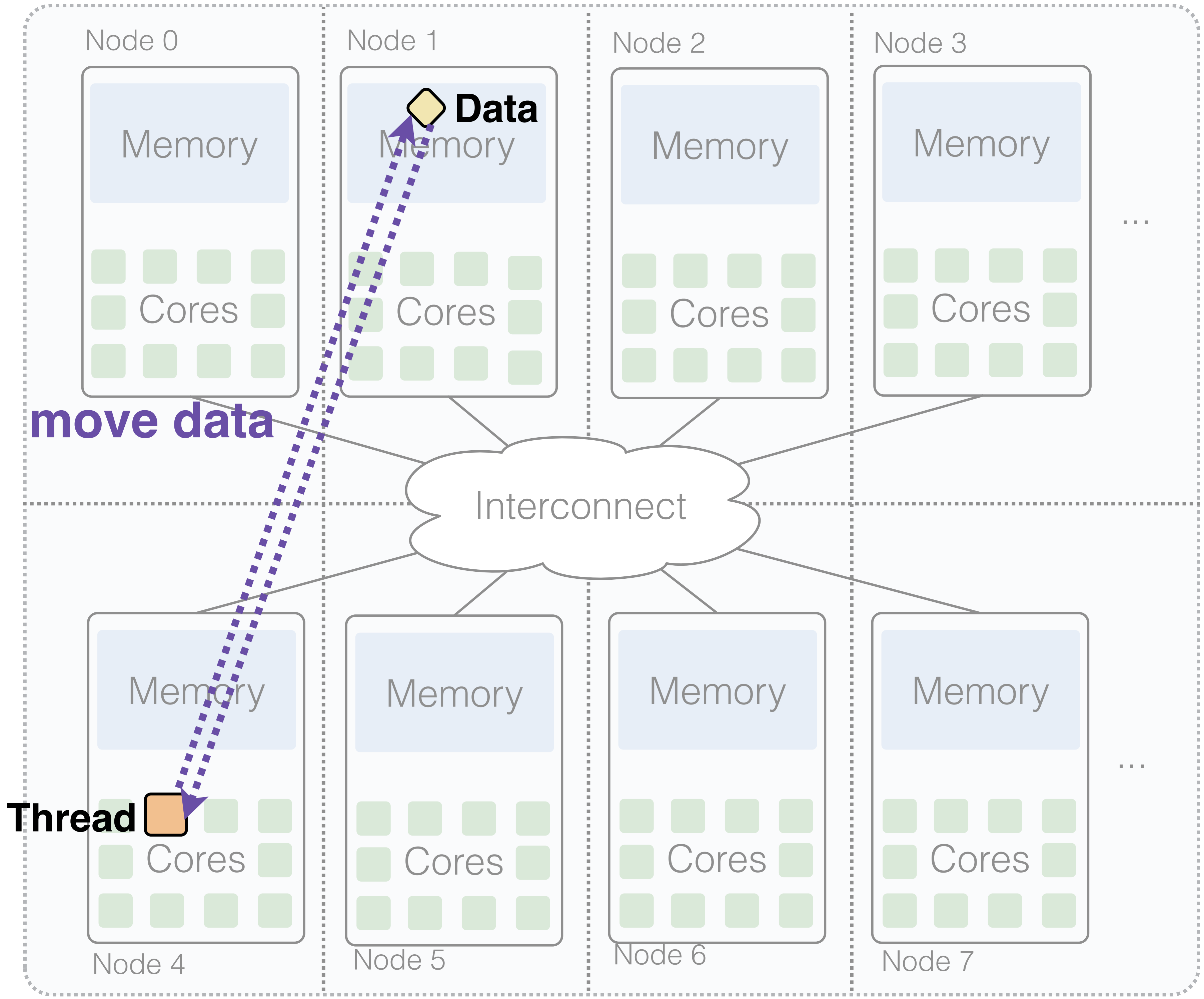
Move data or computation?



* M. C. Carlisle and A. Rogers. Software caching and computation migration in Olden. In *PPOPP '95*, ACM.

† L. V. Kale and S. Krishnan. CHARM++: A portable concurrent object oriented system based on C++. *OOPSLA '93*. ACM.

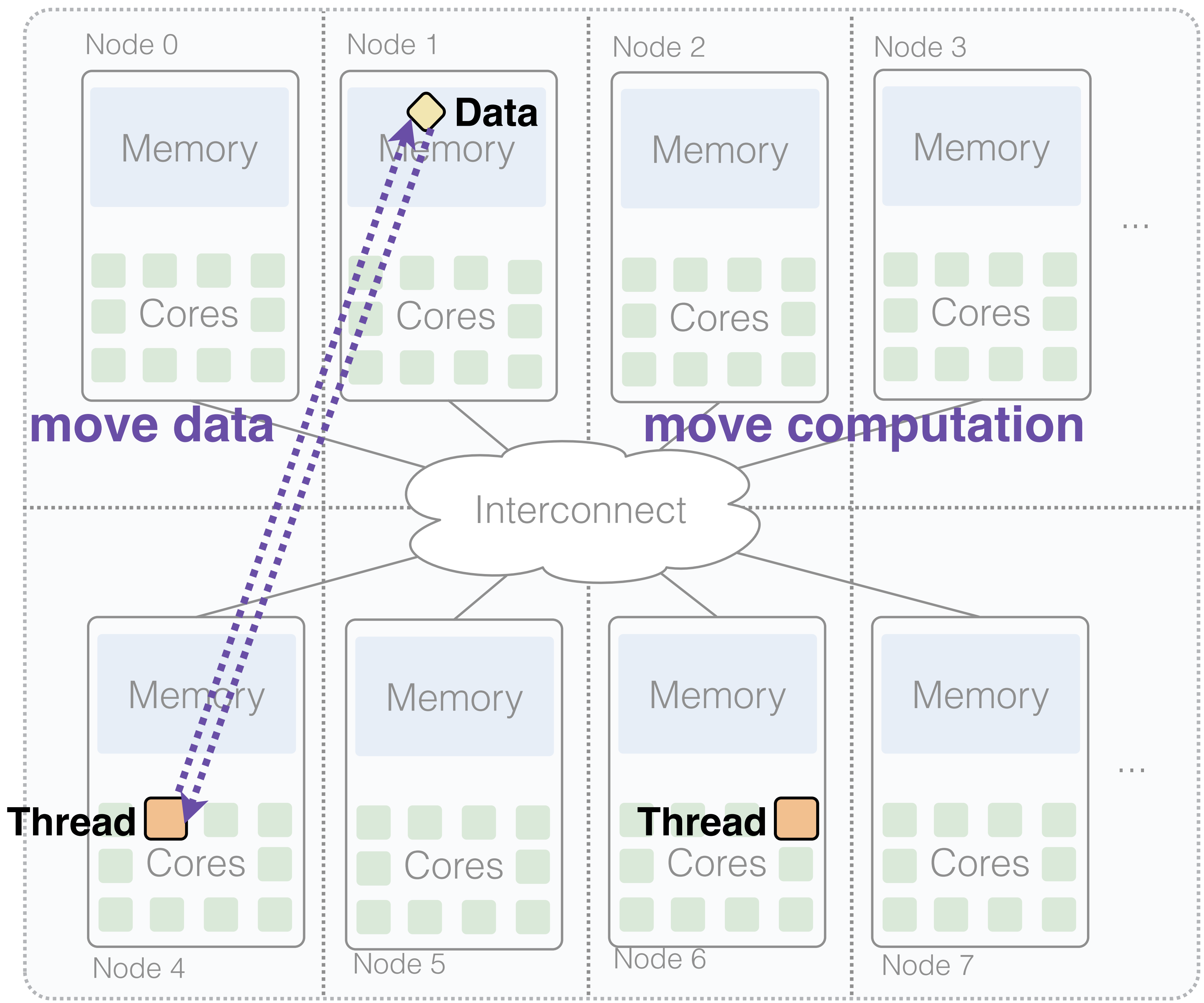
Move data or computation?



* M. C. Carlisle and A. Rogers. Software caching and computation migration in Olden. In *PPOPP '95*, ACM.

† L. V. Kale and S. Krishnan. CHARM++: A portable concurrent object oriented system based on C++. *OOPSLA '93*. ACM.

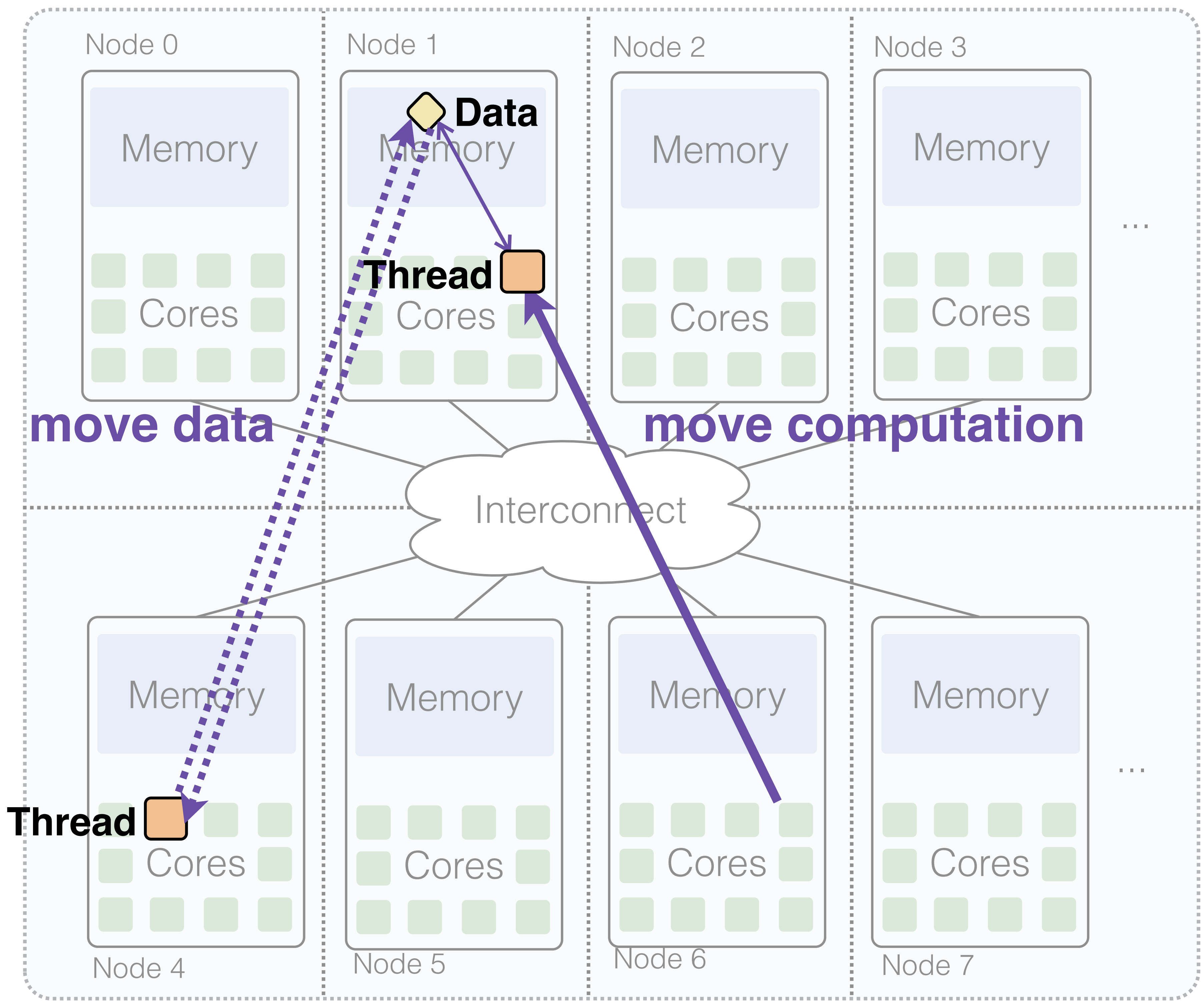
Move data or computation?



* M. C. Carlisle and A. Rogers. Software caching and computation migration in Olden. In *PPOPP '95*, ACM.

† L. V. Kale and S. Krishnan. CHARM++: A portable concurrent object oriented system based on C++. *OOPSLA '93*. ACM.

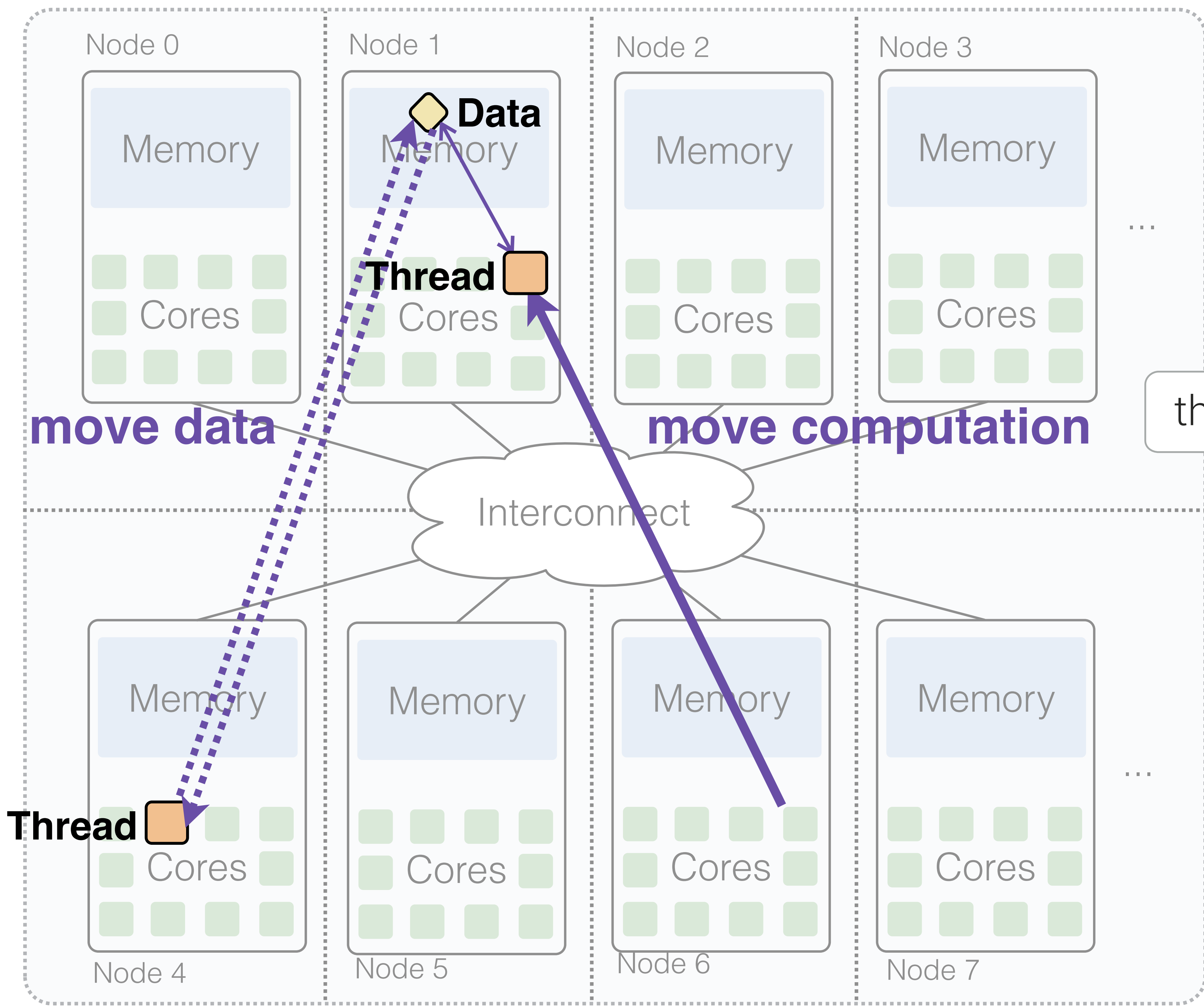
Move data or computation?



* M. C. Carlisle and A. Rogers. Software caching and computation migration in Olden. In *PPOPP '95*, ACM.

† L. V. Kale and S. Krishnan. CHARM++: A portable concurrent object oriented system based on C++. *OOPSLA '93*. ACM.

Move data or computation?

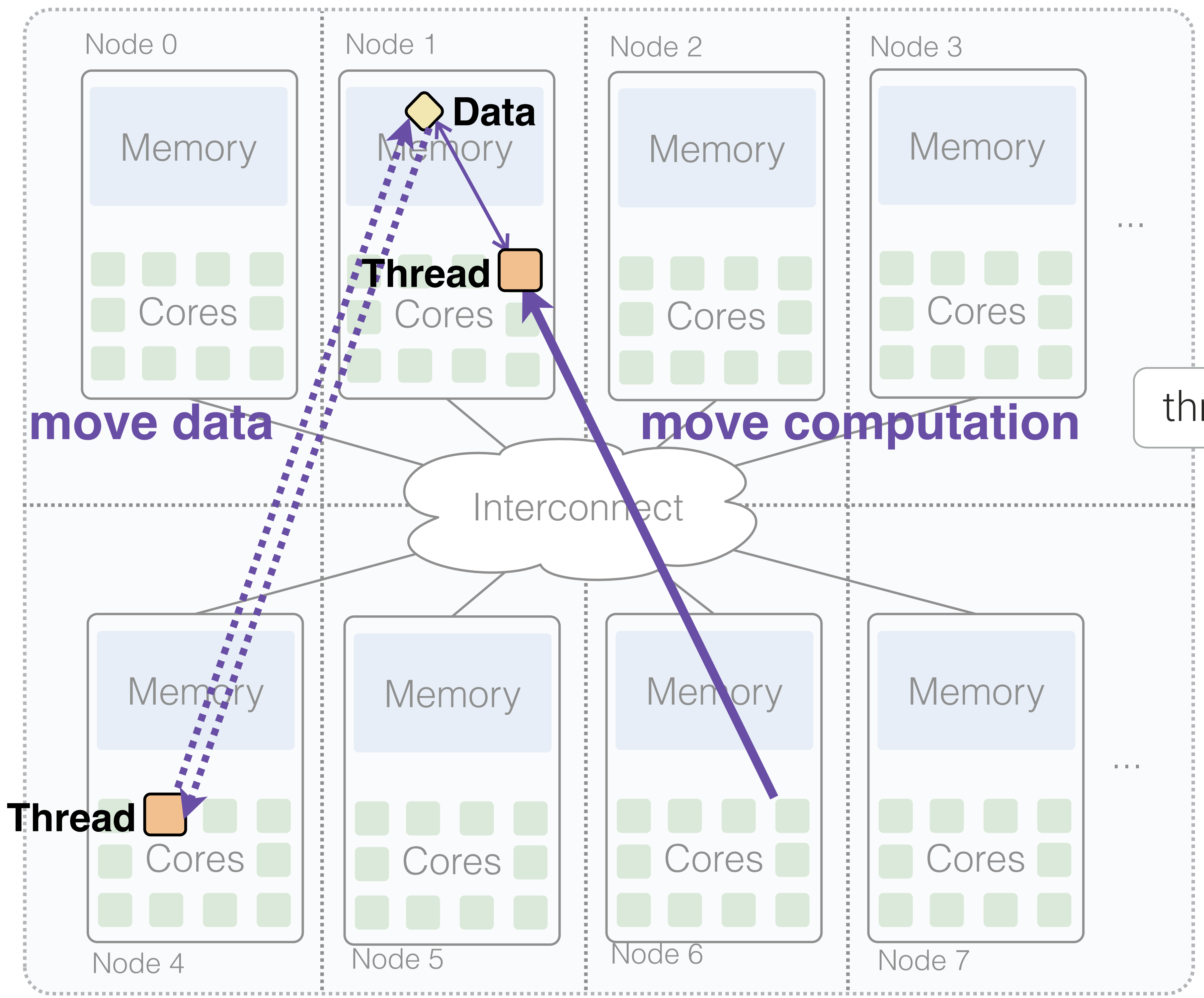


thread migration*

* M. C. Carlisle and A. Rogers. Software caching and computation migration in Olden. In *PPOPP '95*, ACM.

† L. V. Kale and S. Krishnan. CHARM++: A portable concurrent object oriented system based on C++. *OOPSLA '93*. ACM.

Move data or computation?



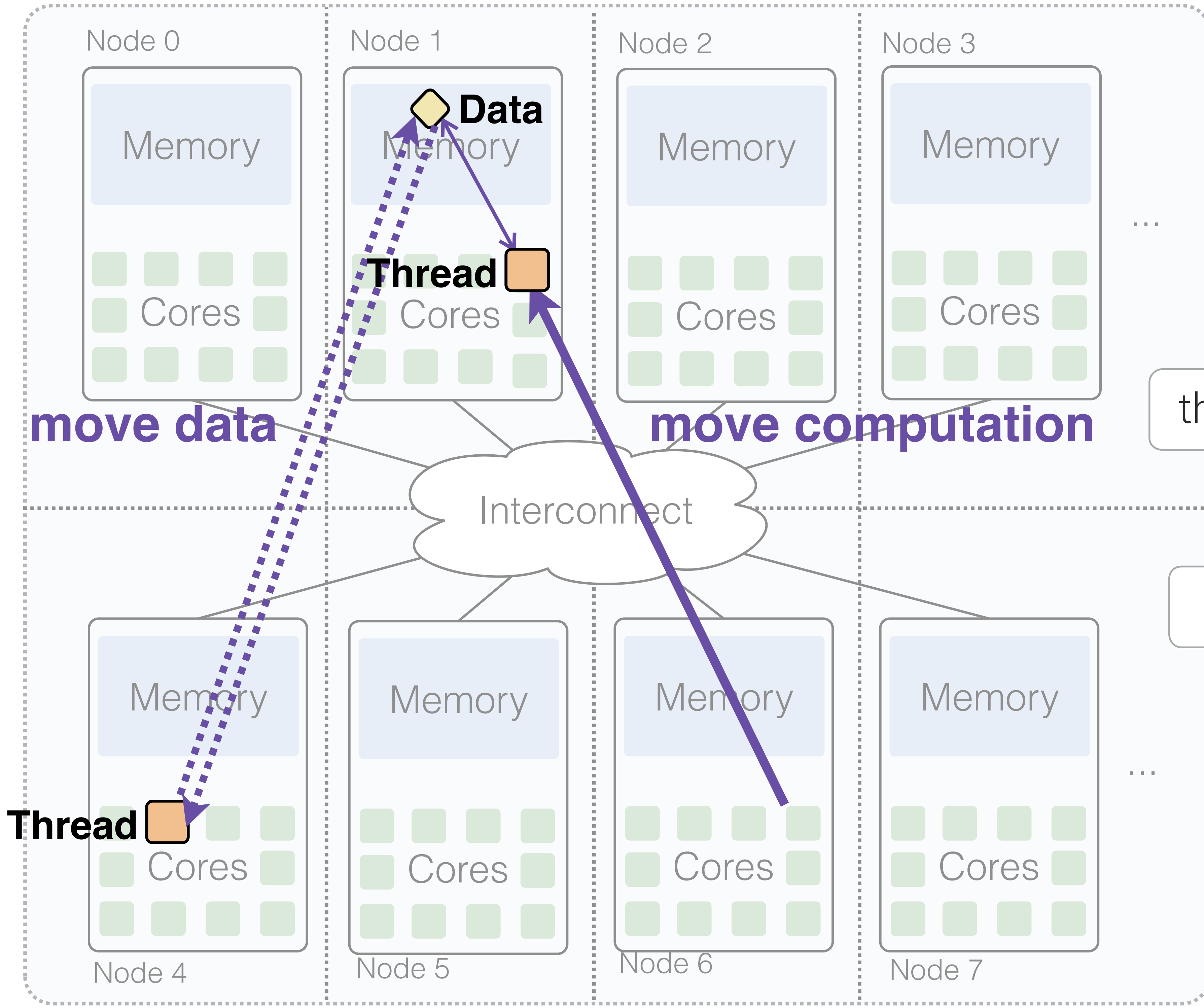
thread migration*

RPC†

* M. C. Carlisle and A. Rogers. Software caching and computation migration in Olden. In *PPOPP '95*, ACM.

† L. V. Kale and S. Krishnan. CHARM++: A portable concurrent object oriented system based on C++. *OOPSLA '93*. ACM.

Move data or computation?



thread migration*

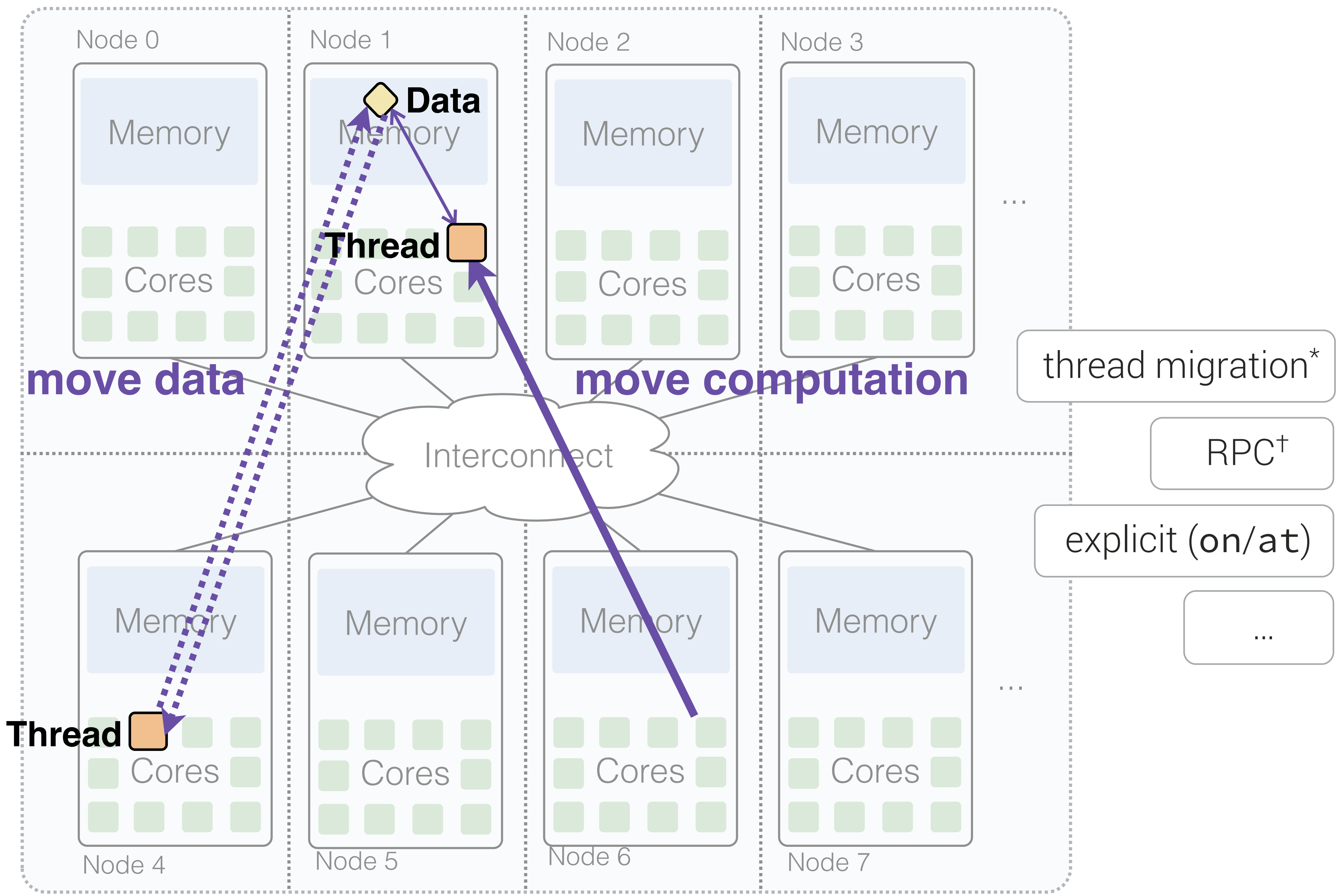
RPC†

explicit (`on`/`at`)

* M. C. Carlisle and A. Rogers. Software caching and computation migration in Olden. In *PPOPP '95*, ACM.

† L. V. Kale and S. Krishnan. CHARM++: A portable concurrent object oriented system based on C++. *OOPSLA '93*. ACM.

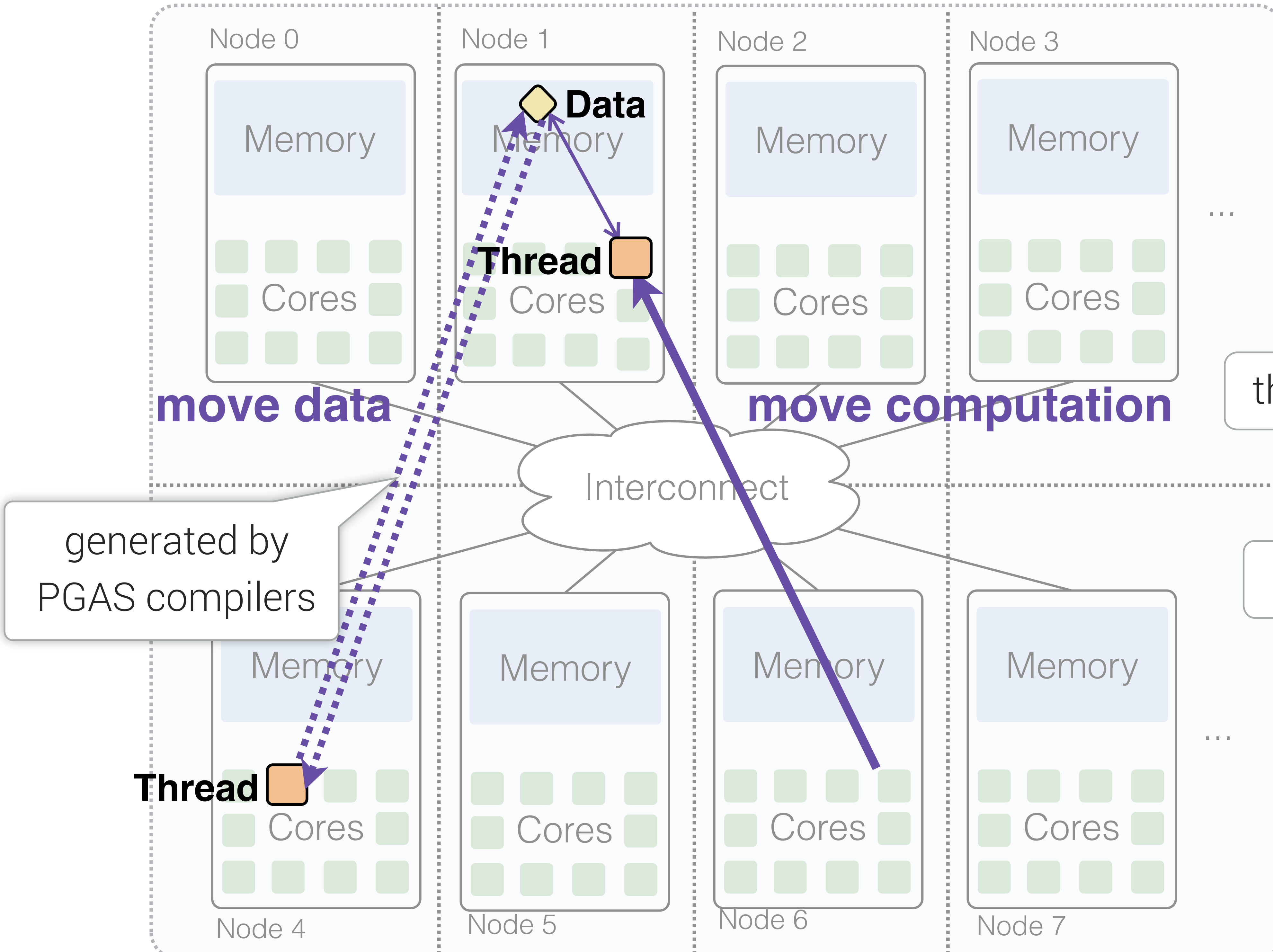
Move data or computation?



* M. C. Carlisle and A. Rogers. Software caching and computation migration in Olden. In *PPOPP '95*, ACM.

† L. V. Kale and S. Krishnan. CHARM++: A portable concurrent object oriented system based on C++. *OOPSLA '93*. ACM.

Move data or computation?



thread migration*

RPC†

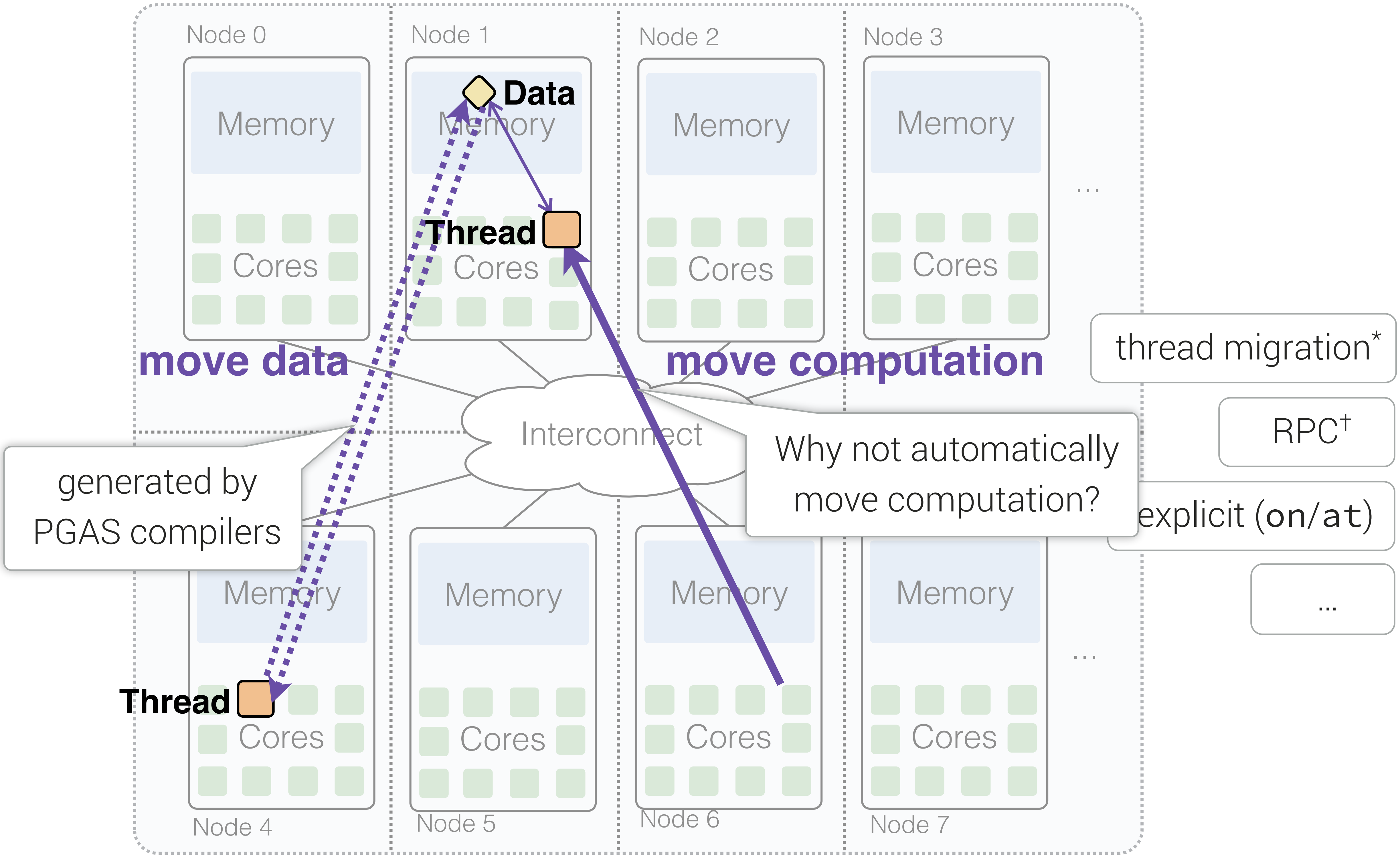
explicit (`on`/`at`)

...

* M. C. Carlisle and A. Rogers. Software caching and computation migration in Olden. In *PPOPP '95*, ACM.

† L. V. Kale and S. Krishnan. CHARM++: A portable concurrent object oriented system based on C++. *OOPSLA '93*. ACM.

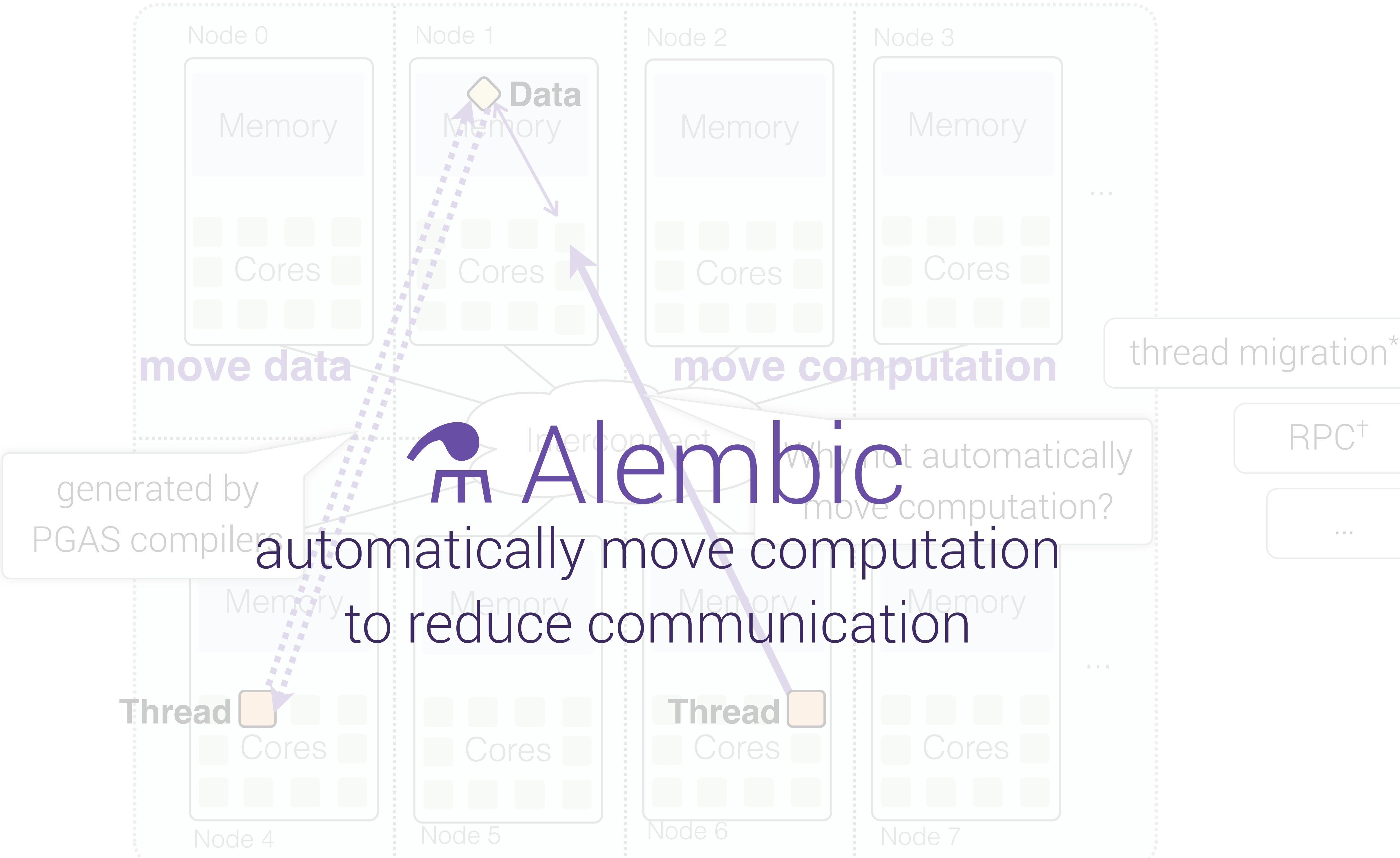
Move data or computation?



* M. C. Carlisle and A. Rogers. Software caching and computation migration in Olden. In *PPOPP '95*, ACM.

† L. V. Kale and S. Krishnan. CHARM++: A portable concurrent object oriented system based on C++. *OOPSLA '93*. ACM.

Move data or computation?



Alembic

automatically move computation
to reduce communication

* M. C. Carlisle and A. Rogers. Software caching and computation migration in Olden. In *PPOPP '95*, ACM.

† L. V. Kale and S. Krishnan. CHARM++: A portable concurrent object oriented system based on C++. *OOPSLA '93*. ACM.

Alembic

Static optimizing migration algorithm

- Constrained by anchor points
- Greedy heuristic to reduce communication

Implementation for C++ in LLVM

Evaluation

- ***6x better*** than naive compiler-generated communication
- 82% of hand-tuned performance

Alembic

Static optimizing migration algorithm

- Constrained by anchor points
- Greedy heuristic to reduce communication

Implementation for C++ in LLVM

Evaluation

- ***6x better*** than naive compiler-generated communication
- 82% of hand-tuned performance

algorithm

algorithm

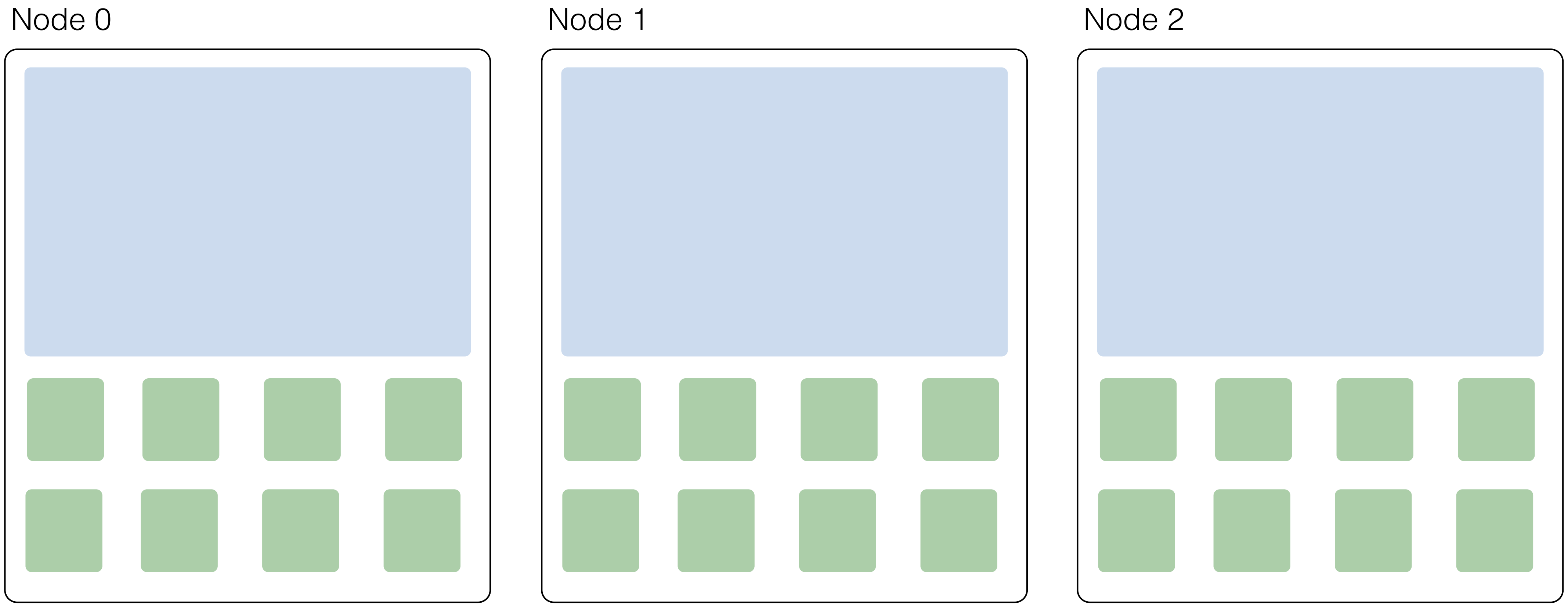
Locality analysis

- Identify *anchor points*
- Partition anchors into *locality sets*

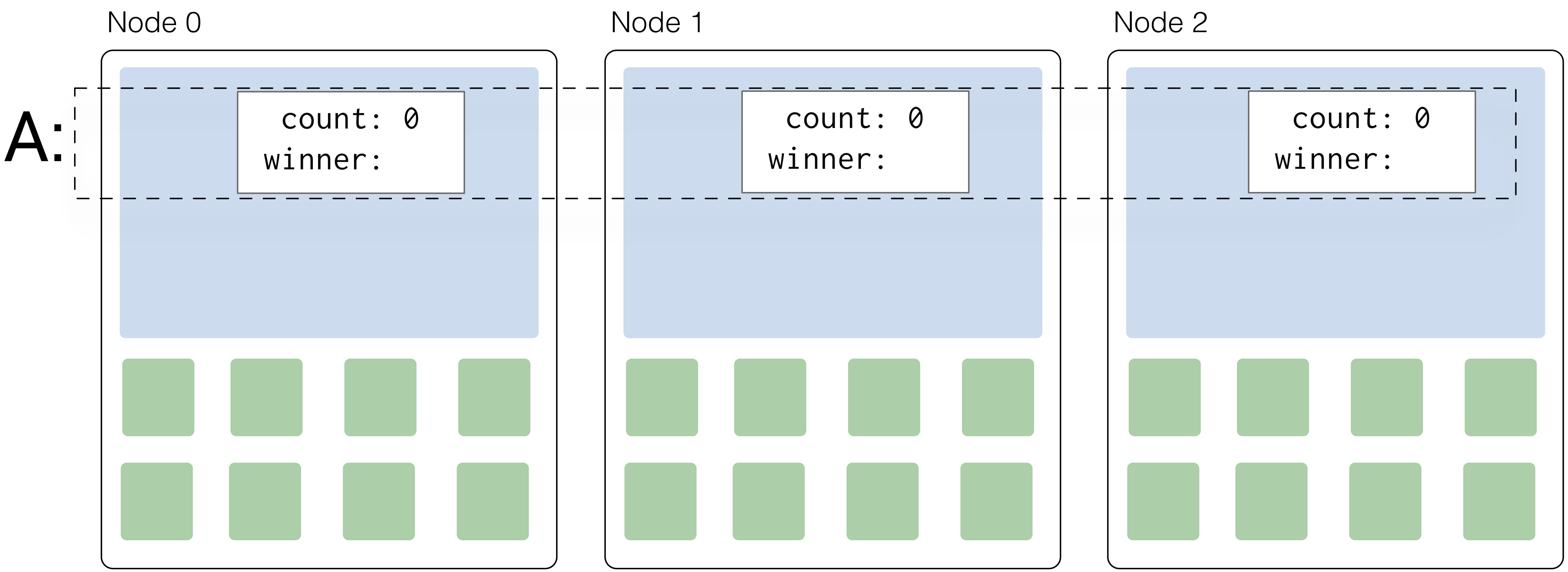
Heuristic region selection

- Divide into regions that *minimize communication*
- Transform task to *migrate* at region boundaries

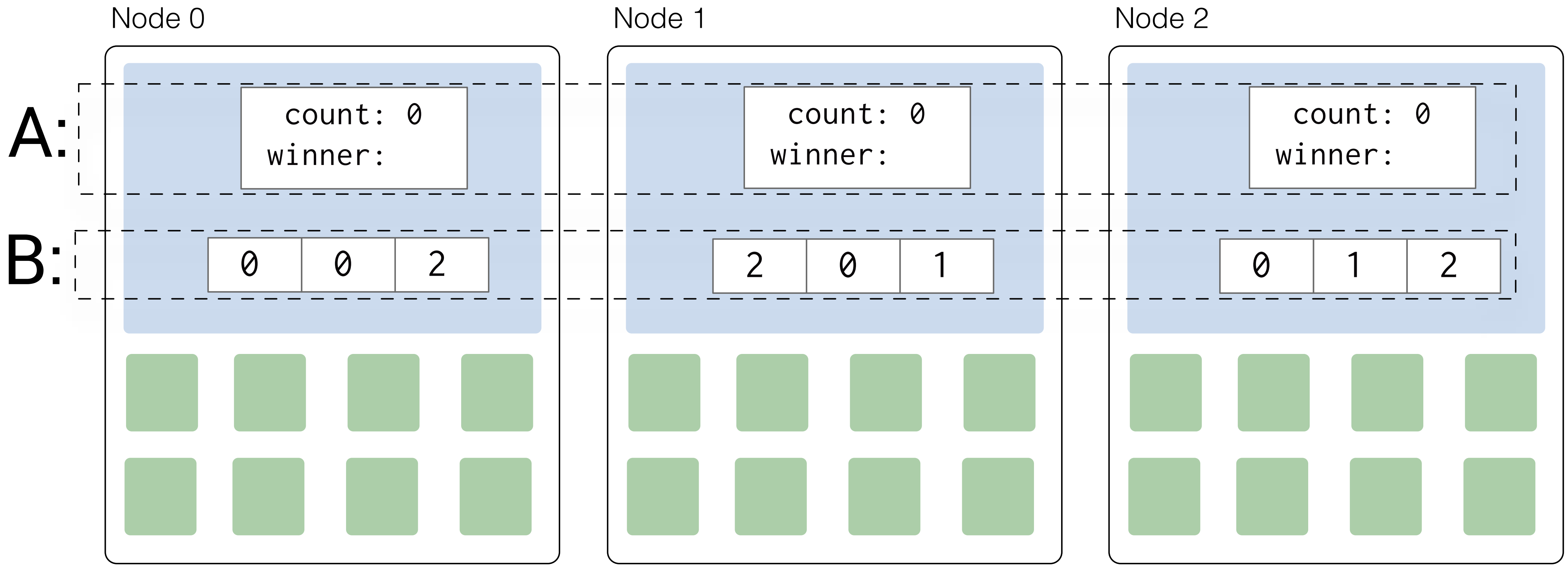
HOPS Benchmark



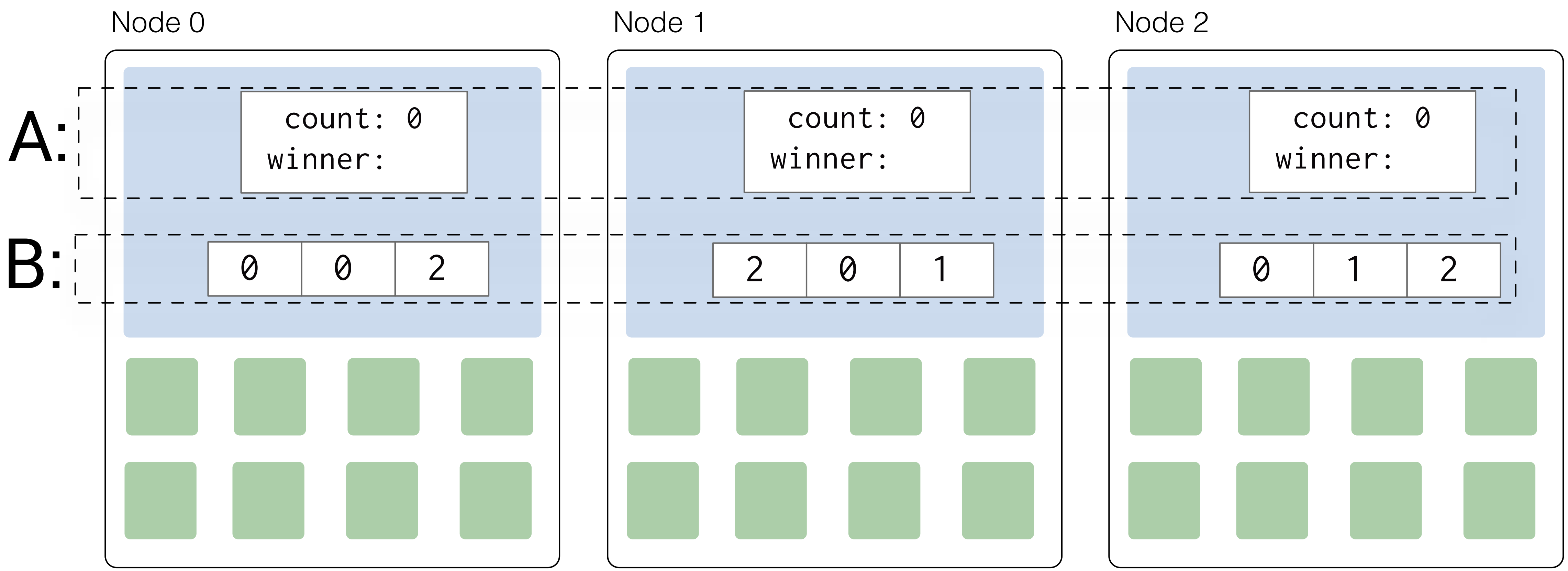
HOPS Benchmark



HOPS Benchmark

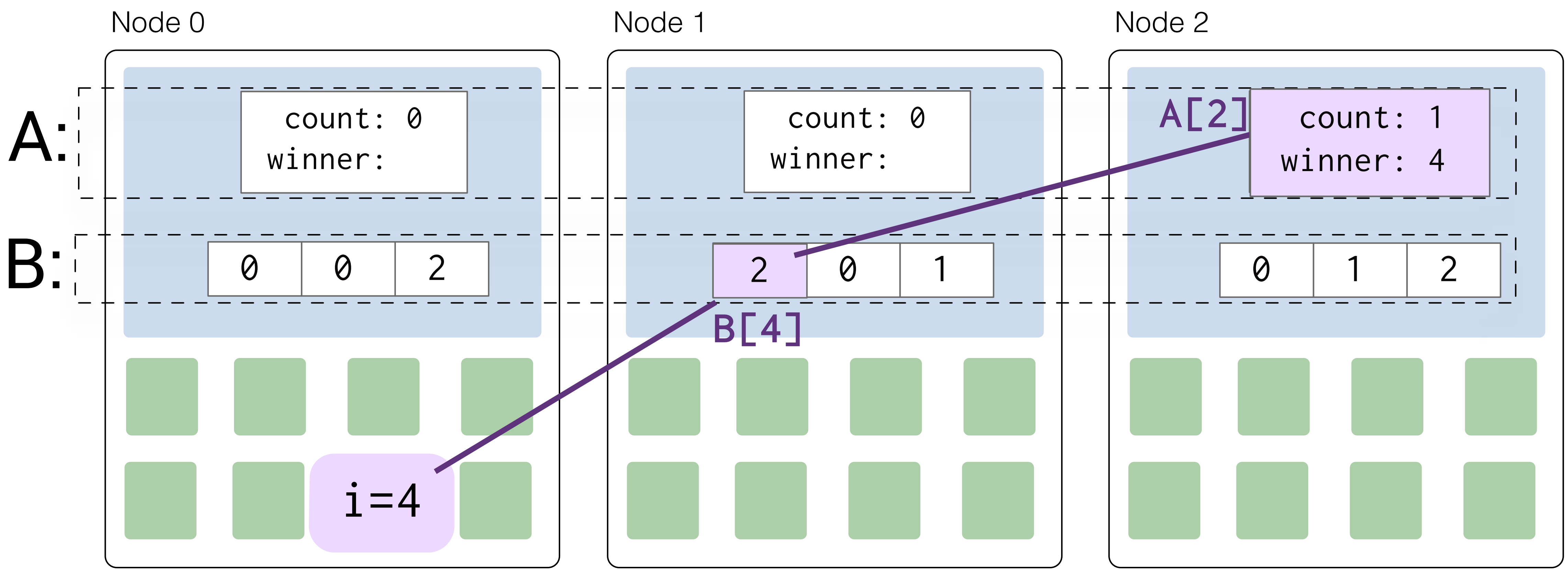


HOPS Benchmark



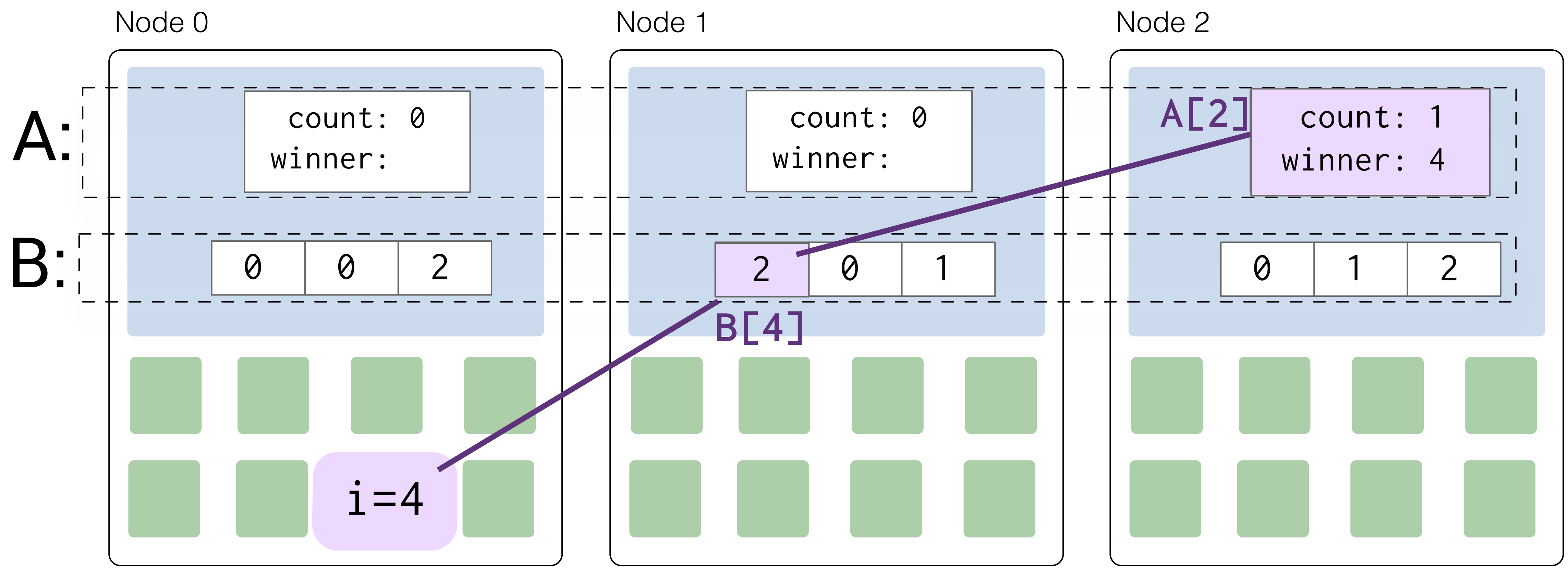

```
forall(0, B.size, [A,B](long i) {
  Counter global* a = A + B[i];
  long prev = fetch_add(&a->count, 1);
  if (prev == 0)    // first to arrive
    a->winner = i; // is winner
});
```

HOPS Benchmark




```
forall(0, B.size, [A,B](long i) {
  Counter global* a = A + B[i];
  long prev = fetch_add(&a->count, 1);
  if (prev == 0)    // first to arrive
    a->winner = i; // is winner
});
```

HOPS Benchmark

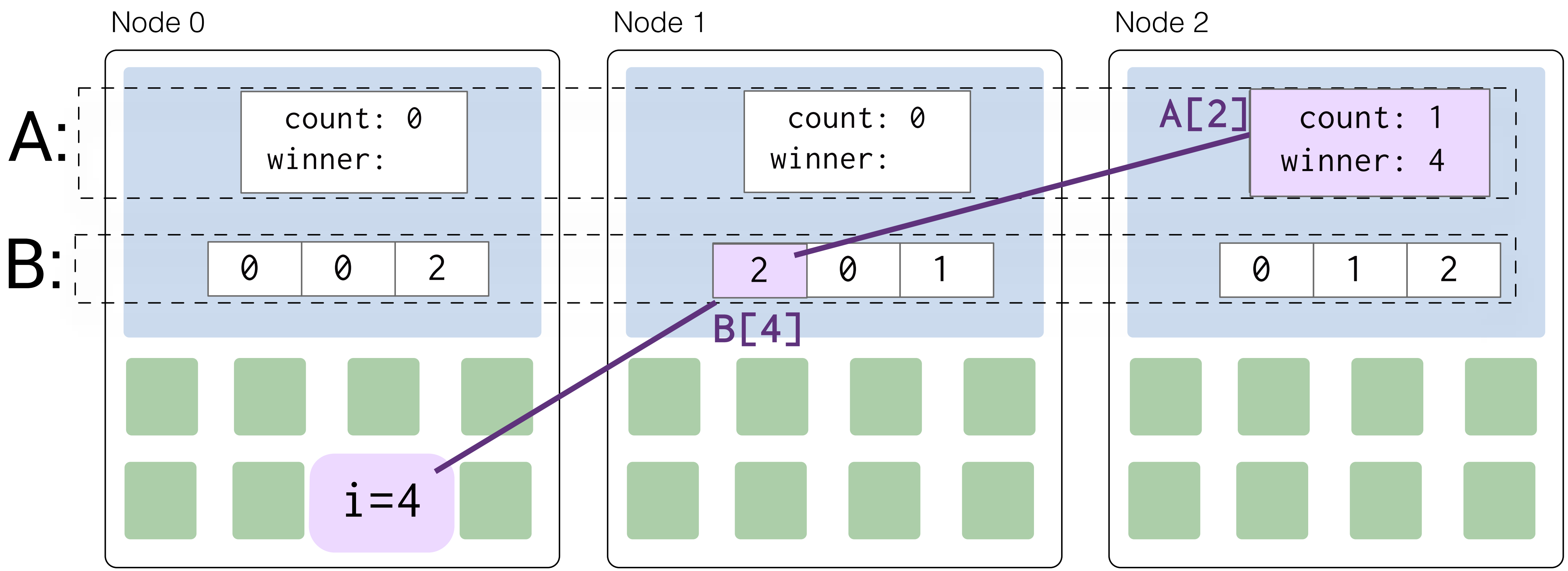



```
forall(0, B.size, [A,B](long i) {
  Counter global* a = A + B[i];
  long prev = fetch_add(&a->count, 1);
  if (prev == 0)    // first to arrive
                                  winner
});
```

Anchor points

- memory locations are *owned* by one node
- so memory references are *anchored* to that node
- these are constraints on the thread's execution

HOPS Benchmark




```
forall(0, B.size, [A,B](long i) {
  Counter global* a = A + B[i];
  long prev = fetch_add(&a->count, 1);
  if (prev == 0)    // first to arrive
    a->winner = i;  // is winner
});
```

`A`

`i`

`B`

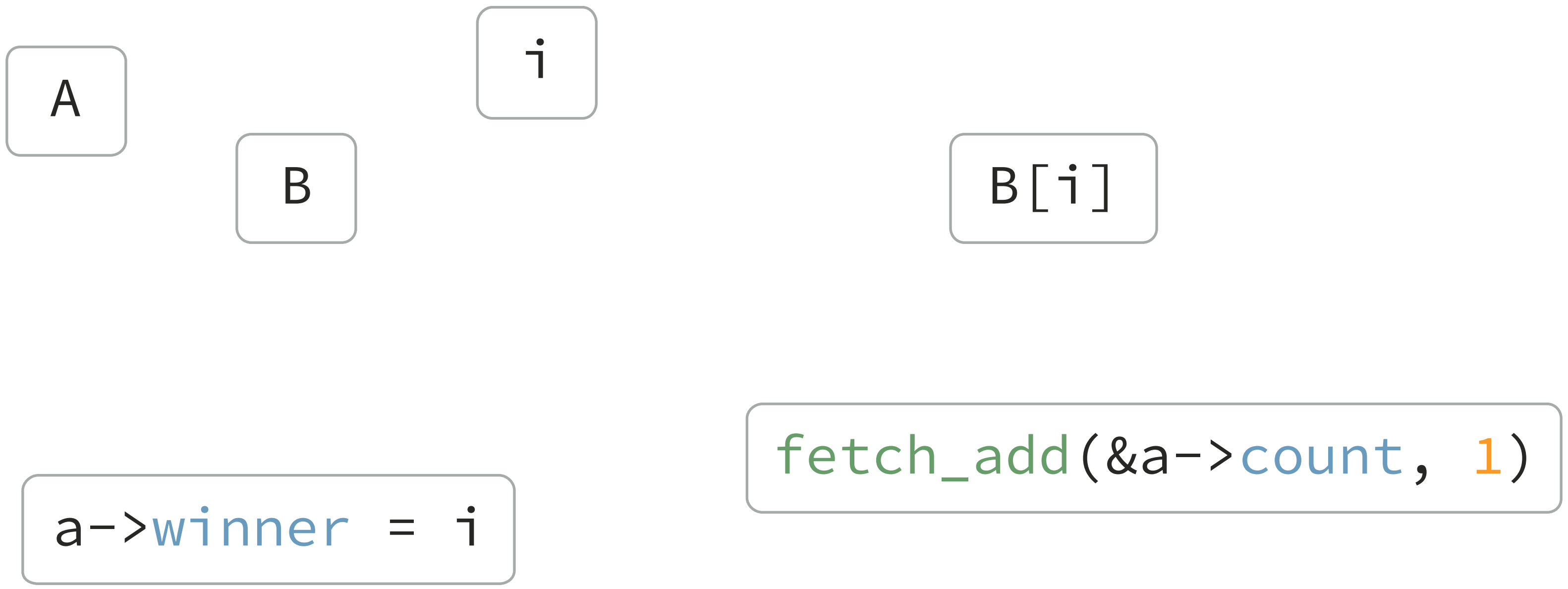
`B[i]`

`fetch_add(&a->count, 1)`

`a->winner = i`

Locality partitioning: *pessimistic value partitioning* (value numbering)*

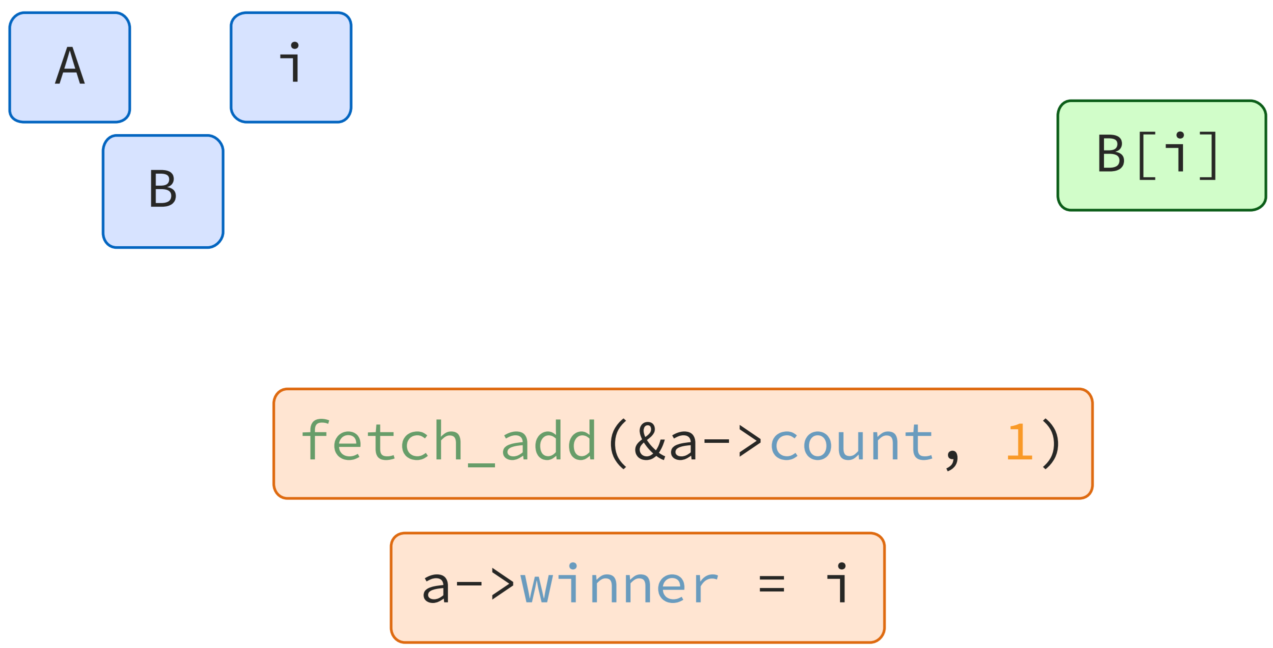
- each anchor starts in its own set
- merge sets if you can prove they are *congruent*
- for locality partitioning: *congruence* means *on the same node*



* B. Alpern, M. N. Wegman, and F. K. Zadeck. Detecting equality of variables in programs. *POPL '88*, pages 1–11. ACM, 1988.

Locality partitioning: *pessimistic value partitioning* (value numbering)*

- each anchor starts in its own set
- merge sets if you can prove they are *congruent*
- for locality partitioning: *congruence* means *on the same node*



* B. Alpern, M. N. Wegman, and F. K. Zadeck. Detecting equality of variables in programs. *POPL '88*, pages 1–11. ACM, 1988.

Locality partitioning: *pessimistic value partitioning* (value numbering)*

- each anchor starts in its own set
- merge sets if you can prove they are *congruent*
- for locality partitioning: *congruence* means *on the same node*

A i

B

B[i]

```
fetch_add(&a->count, 1)
```

```
a->winner = i
```

Plug in your own
locality-congruence rules!

* B. Alpern, M. N. Wegman, and F. K. Zadeck. Detecting equality of variables in programs. *POPL '88*, pages 1–11. ACM, 1988.

Region selection (heuristic optimization)

```
[A,B](long i) {
  Counter global* a = A + B[i];
  long prev = fetch_add(&a->count, 1)
  if (prev == 0)    // first to arrive
    a->winner = i   // is winner
}
```

Region selection (heuristic optimization)

region:
contiguous sequence of instructions (or a DAG of basic blocks) which can all execute on the same node

```
[A,B](long i) {
  Counter global* a = A + B[i];
  long prev = fetch_add(&a->count, 1)
  if (prev == 0)    // first to arrive
    a->winner = i   // is winner
}
```

Region selection (heuristic optimization)

region:

contiguous sequence of instructions (or a DAG of basic blocks) which can all execute on the same node

communication cost heuristic:

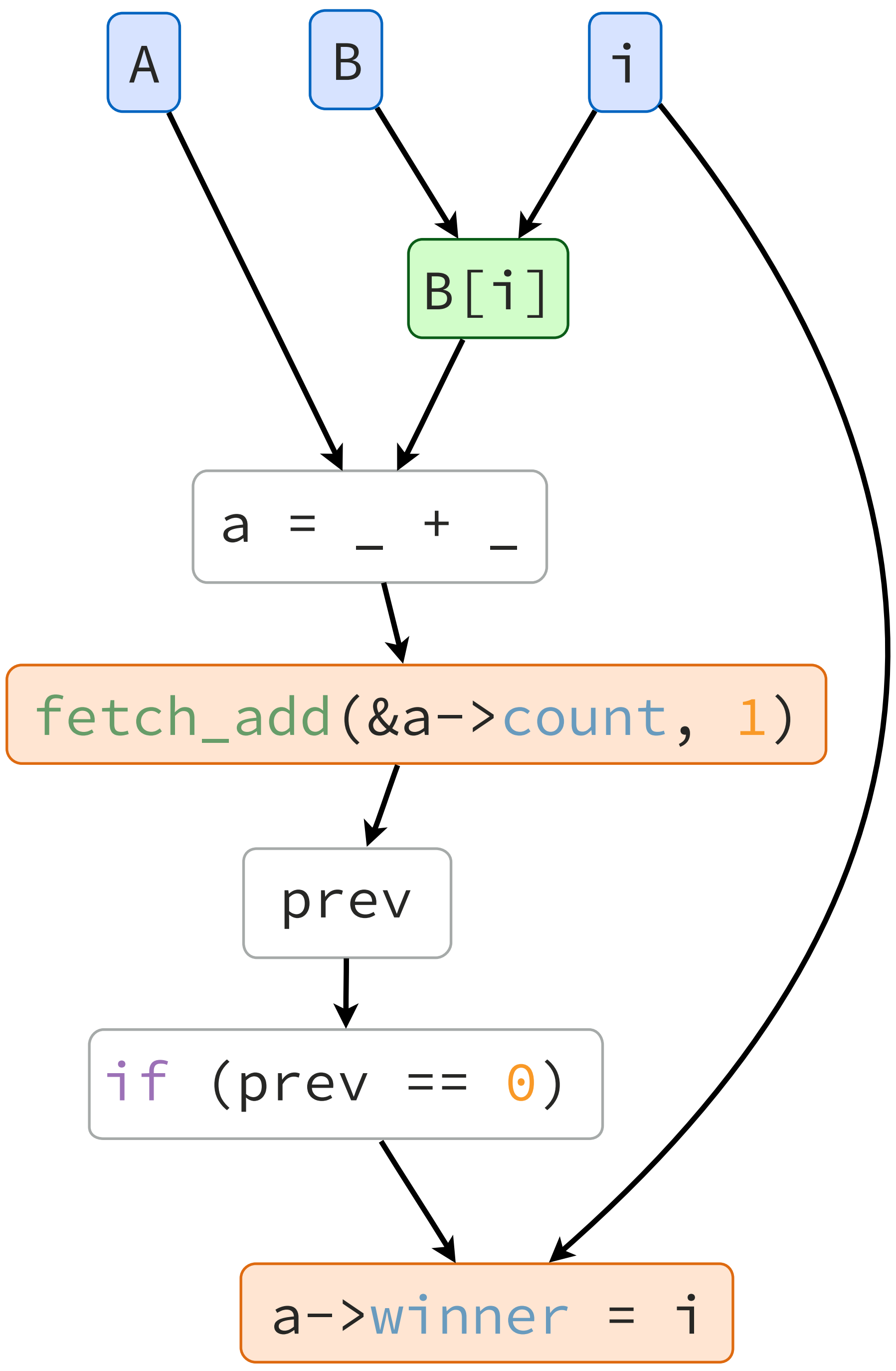
function of *# of messages* and *message size* (continuation size)

```
[A,B](long i) {
  Counter global* a = A + B[i];
  long prev = fetch_add(&a->count, 1)
  if (prev == 0)    // first to arrive
    a->winner = i   // is winner
}
```

Region selection (heuristic optimization)

region:
contiguous sequence of instructions (or a DAG of basic blocks) which can all execute on the same node

communication cost heuristic:
function of *# of messages* and *message size* (continuation size)

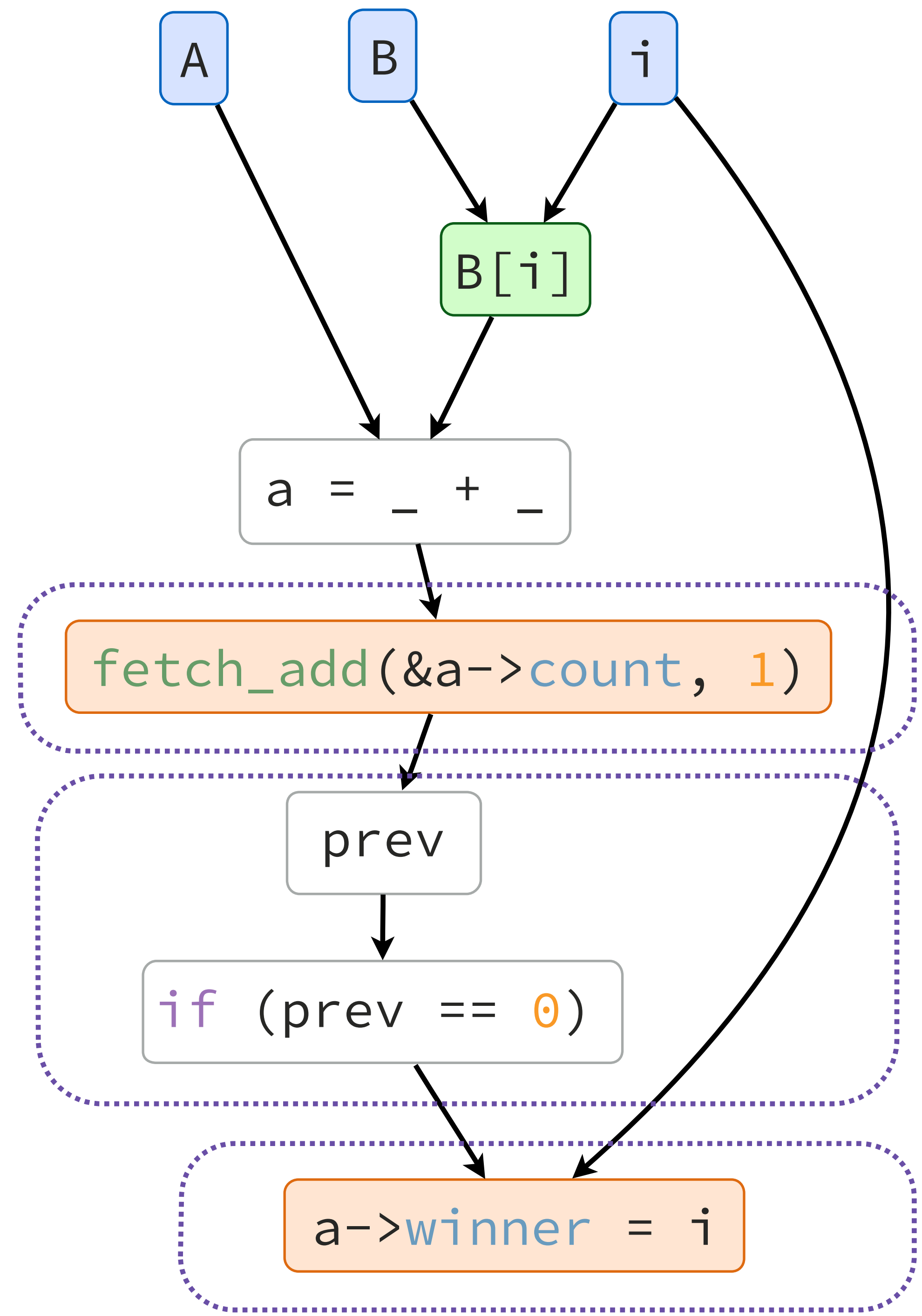


Dependence Graph

Region selection (heuristic optimization)

region:
contiguous sequence of instructions (or a DAG of basic blocks) which can all execute on the same node

communication cost heuristic:
function of *# of messages* and *message size* (continuation size)

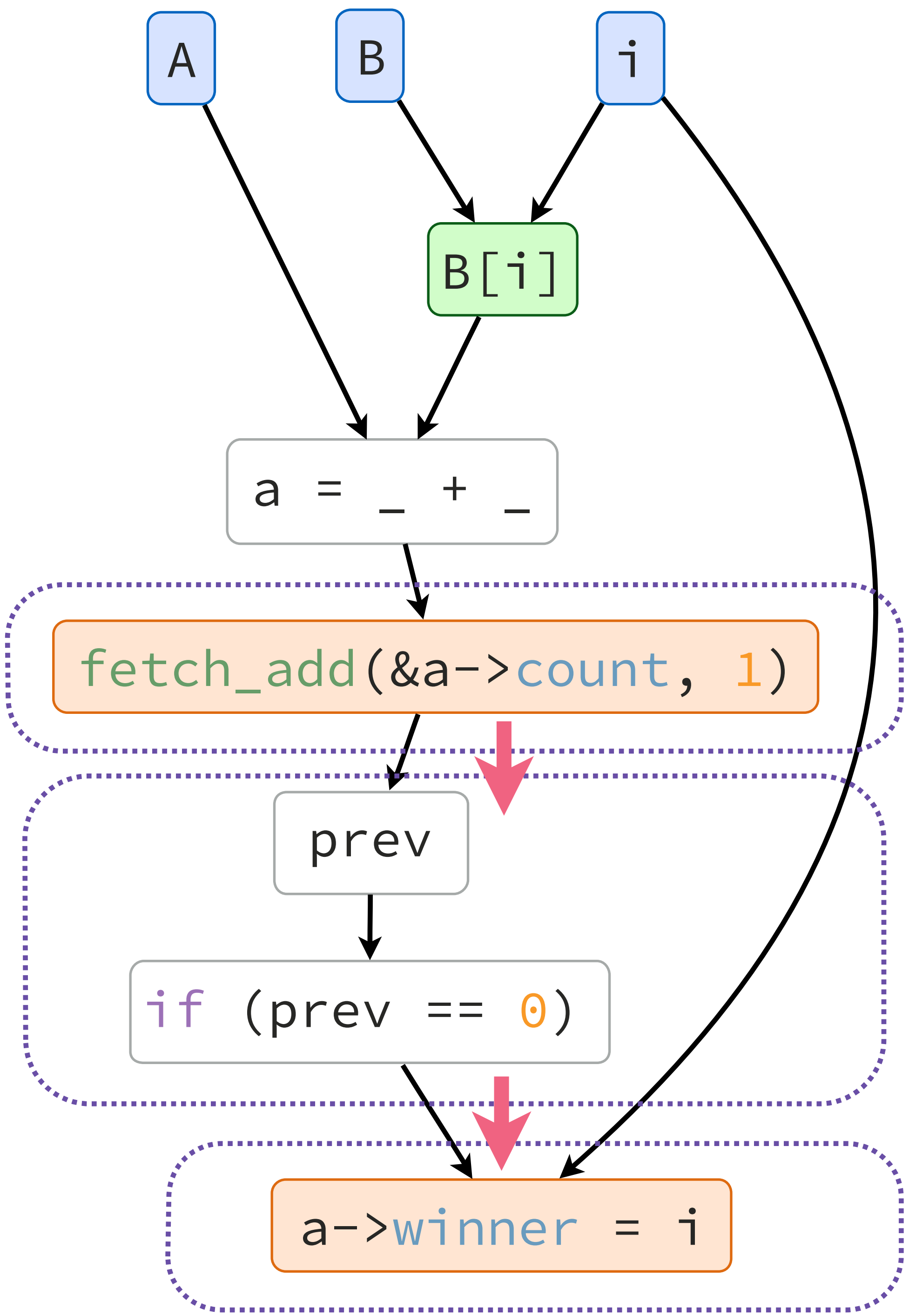


Dependence Graph

Region selection (heuristic optimization)

region:
contiguous sequence of instructions (or a DAG of basic blocks) which can all execute on the same node

communication cost heuristic:
function of *# of messages* and *message size* (continuation size)

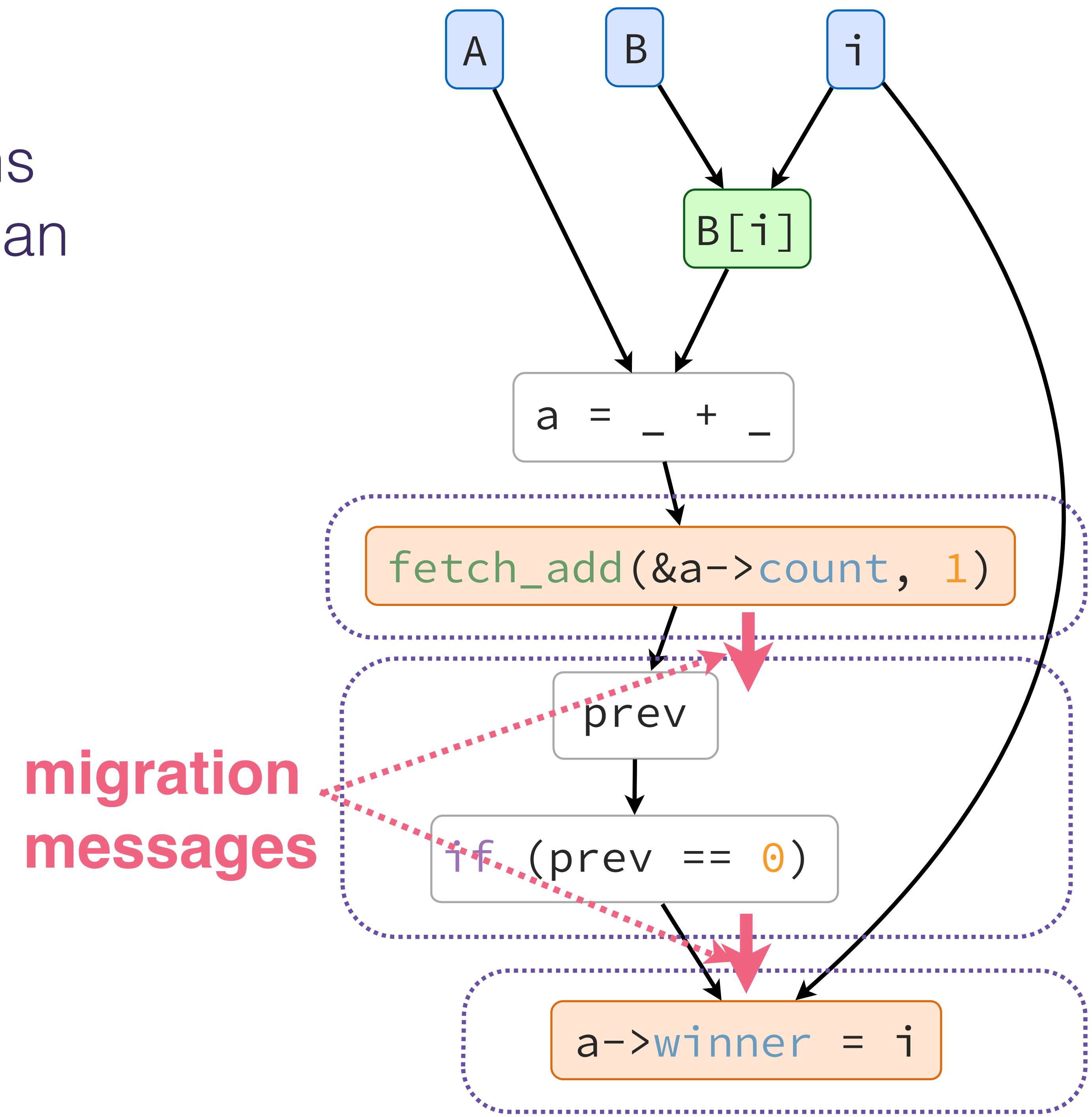


Dependence Graph

Region selection (heuristic optimization)

region:
contiguous sequence of instructions (or a DAG of basic blocks) which can all execute on the same node

communication cost heuristic:
function of *# of messages* and *message size* (continuation size)

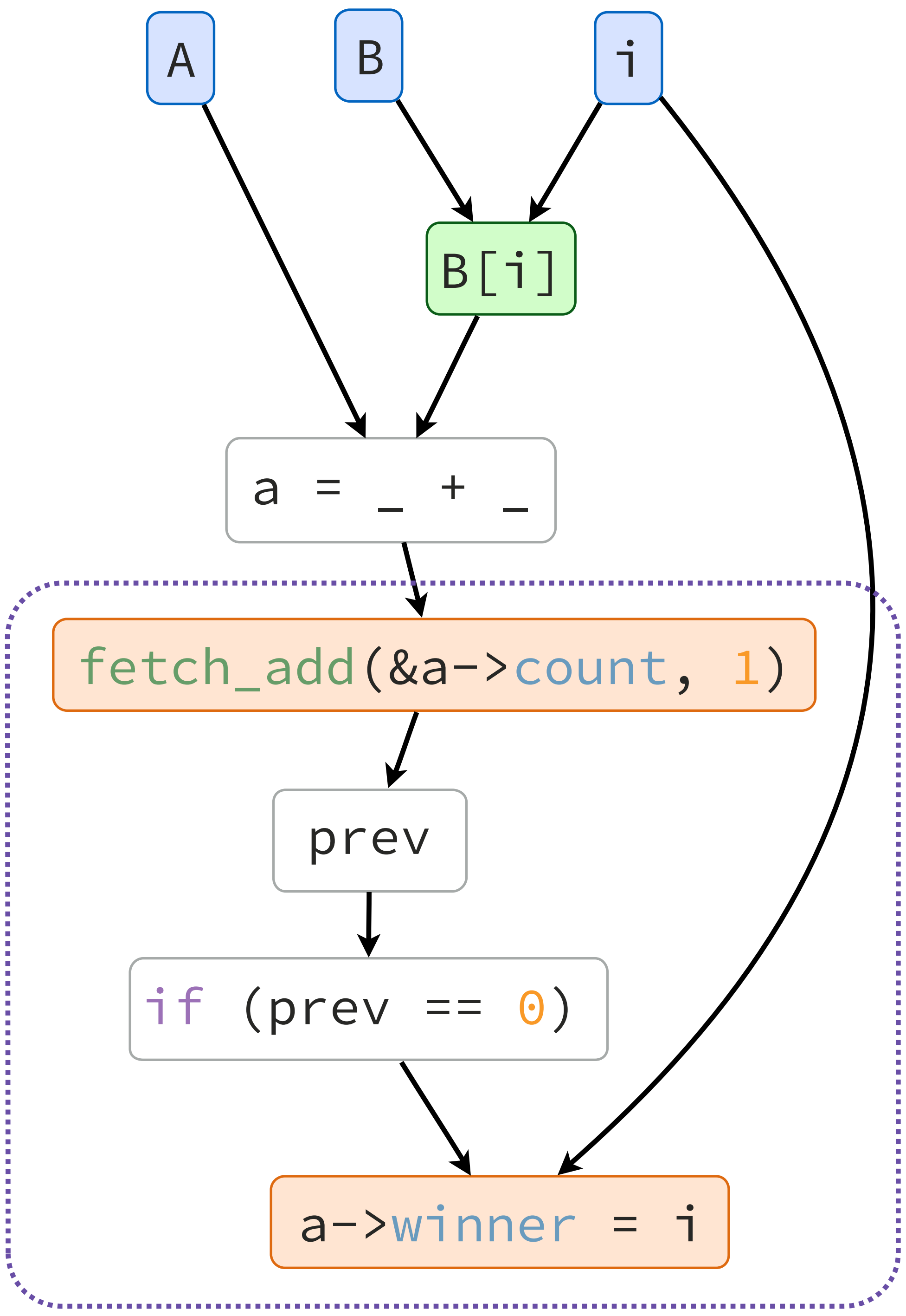


Dependence Graph

Region selection (heuristic optimization)

region:
contiguous sequence of instructions (or a DAG of basic blocks) which can all execute on the same node

communication cost heuristic:
function of *# of messages* and *message size* (continuation size)

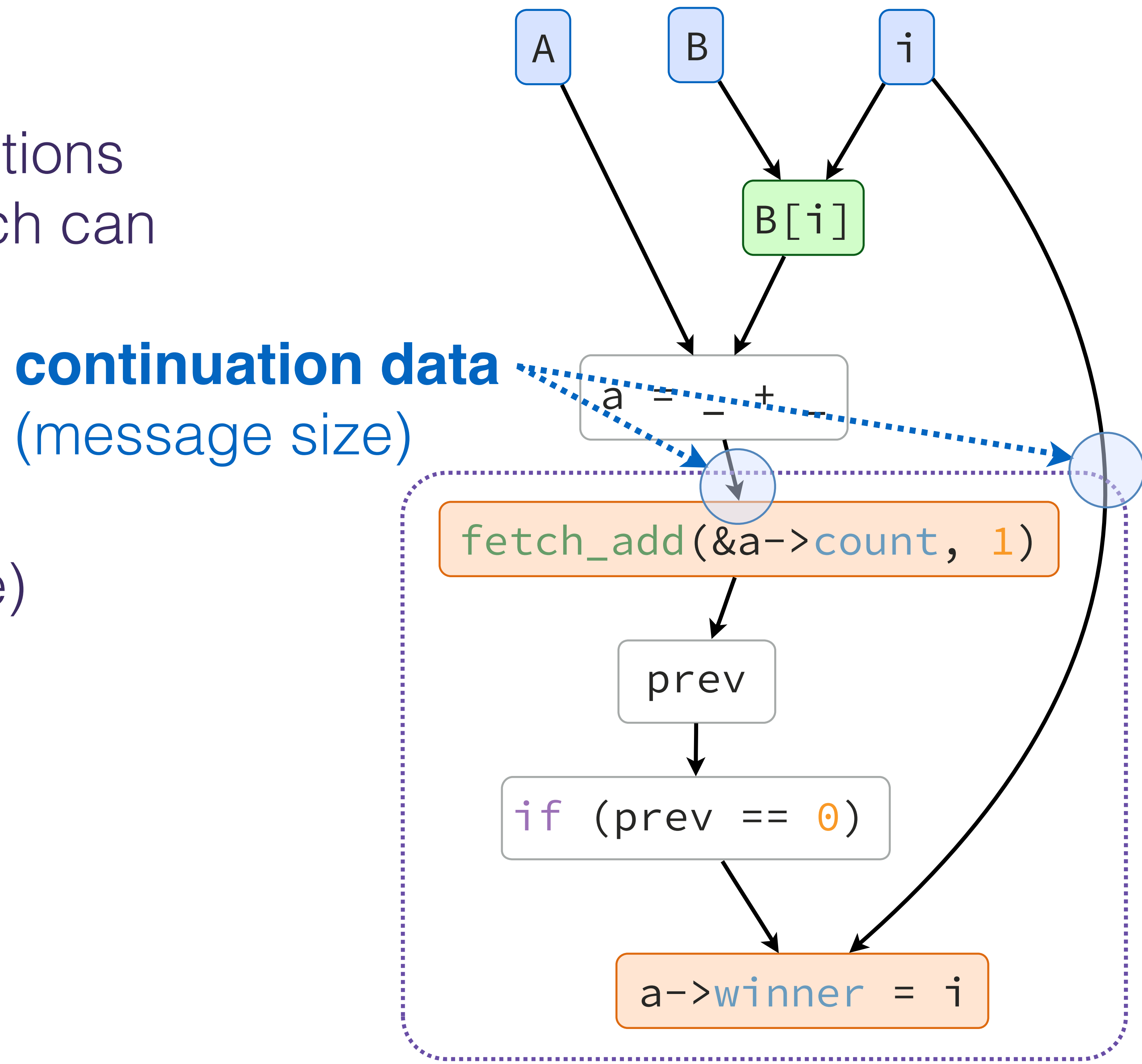


Dependence Graph

Region selection (heuristic optimization)

region:
contiguous sequence of instructions (or a DAG of basic blocks) which can all execute on the same node

communication cost heuristic:
function of *# of messages* and *message size* (continuation size)

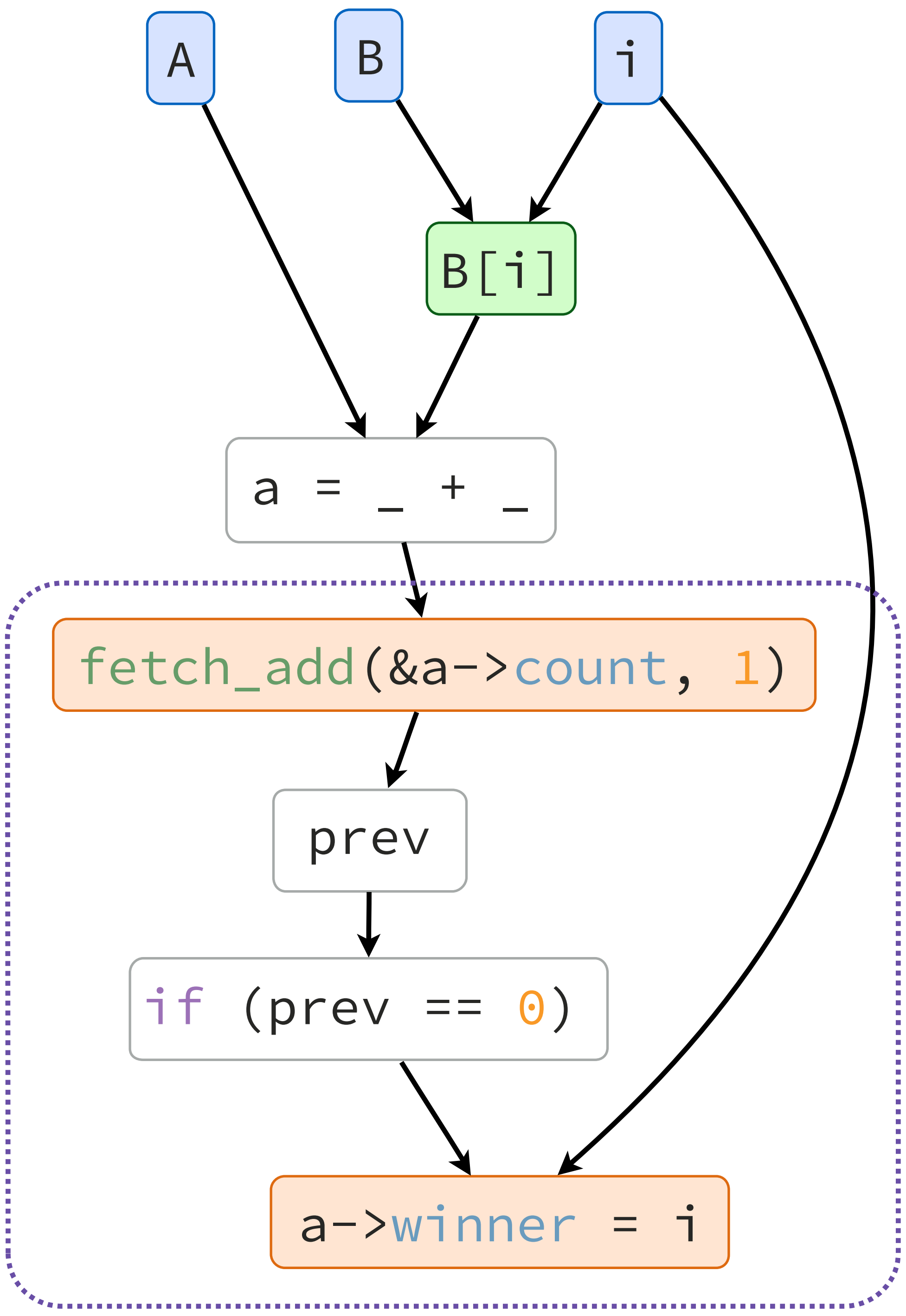


Dependence Graph

Region selection (heuristic optimization)

region:
contiguous sequence of instructions (or a DAG of basic blocks) which can all execute on the same node

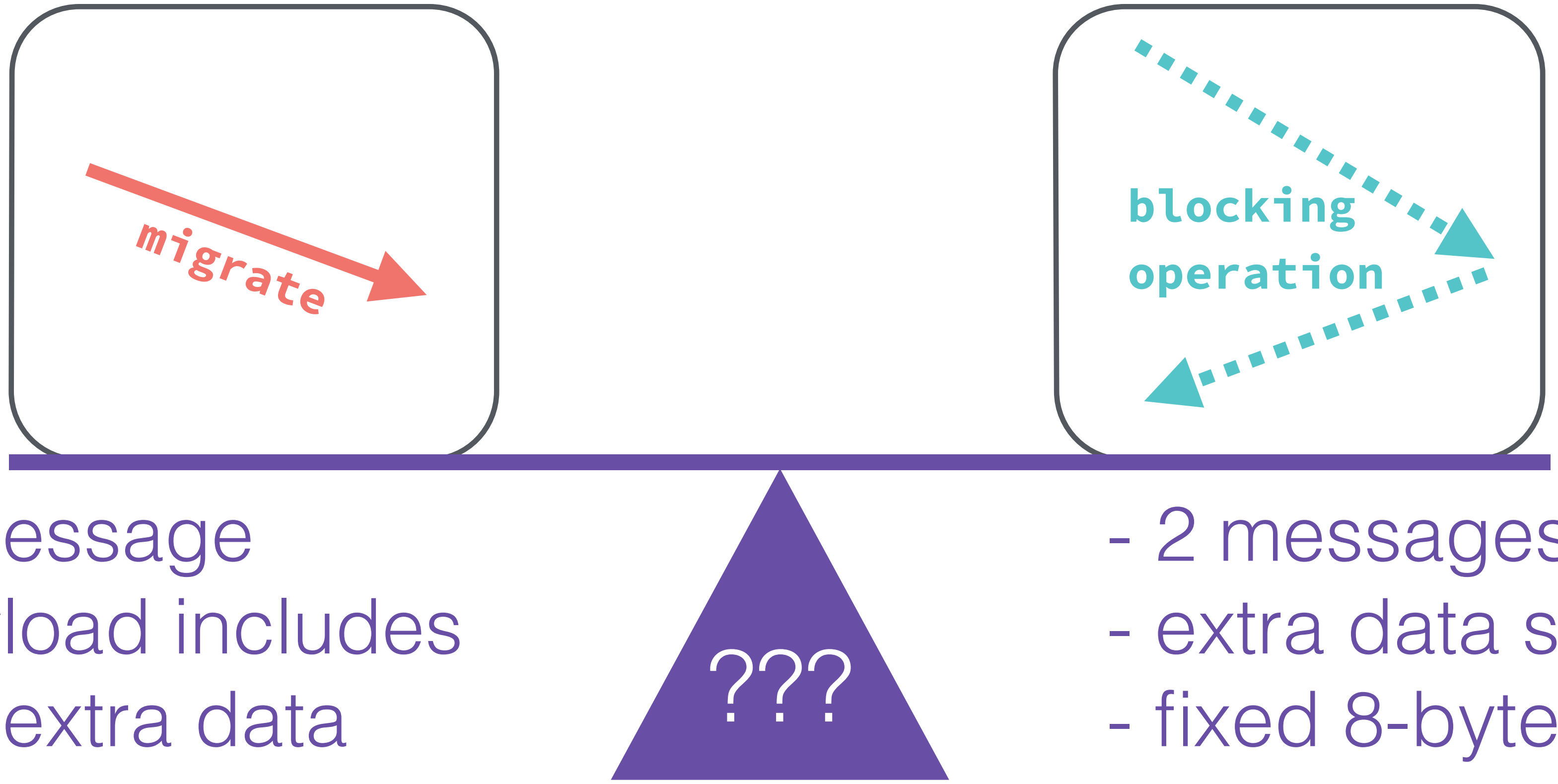
communication cost heuristic:
function of *# of messages* and *message size* (continuation size)



Dependence Graph

Message cost experiment

Modified HOPS to use extra data
after the remote operation

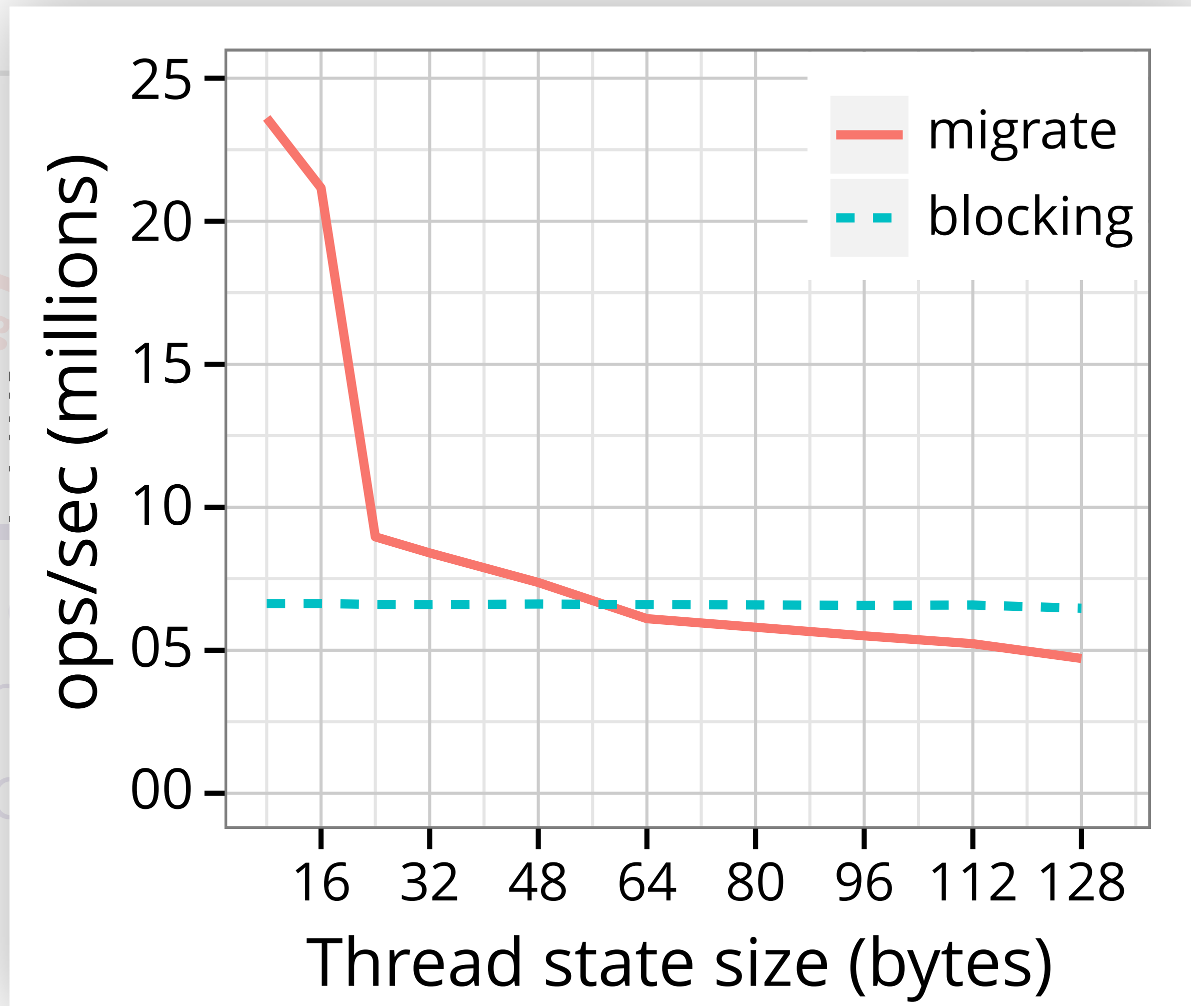


- 1 message
- payload includes the extra data

- 2 messages
- extra data stays
- fixed 8-byte payload

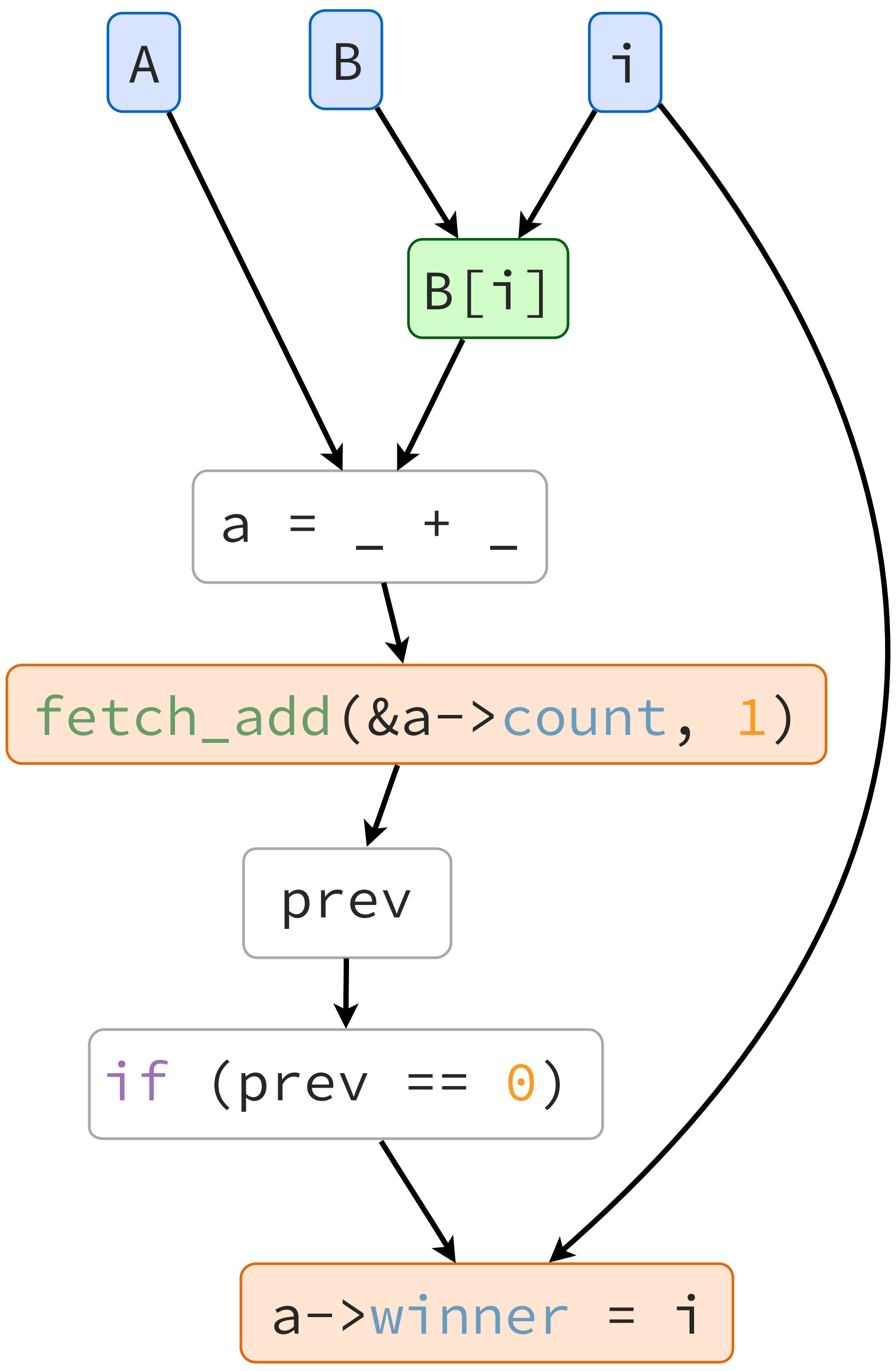
Message cost experiment

Modified HOPS to use extra data
after the remote operation

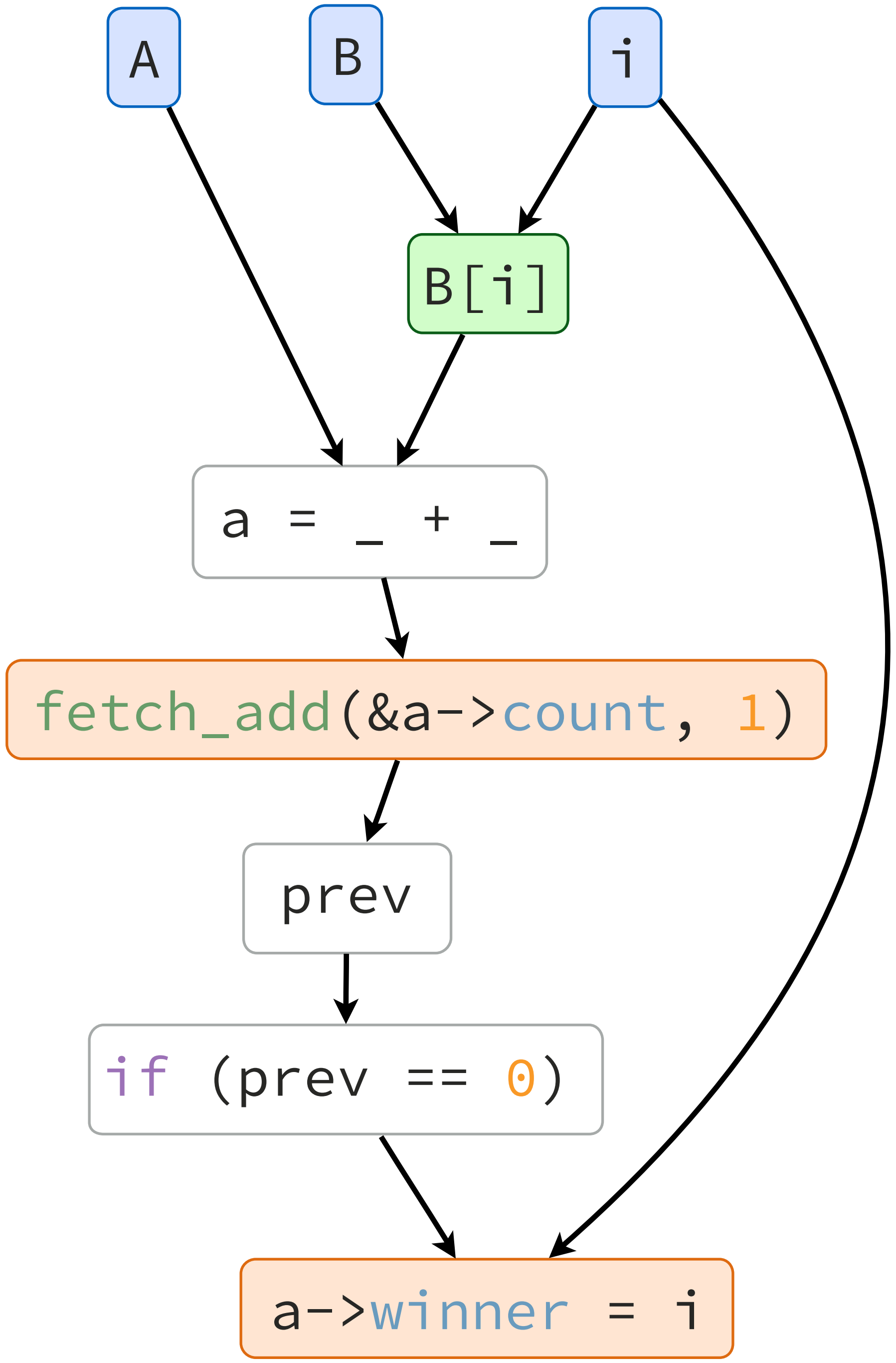


- 1 messag
- payload i
the extra c

ages
ata stays
byte payload

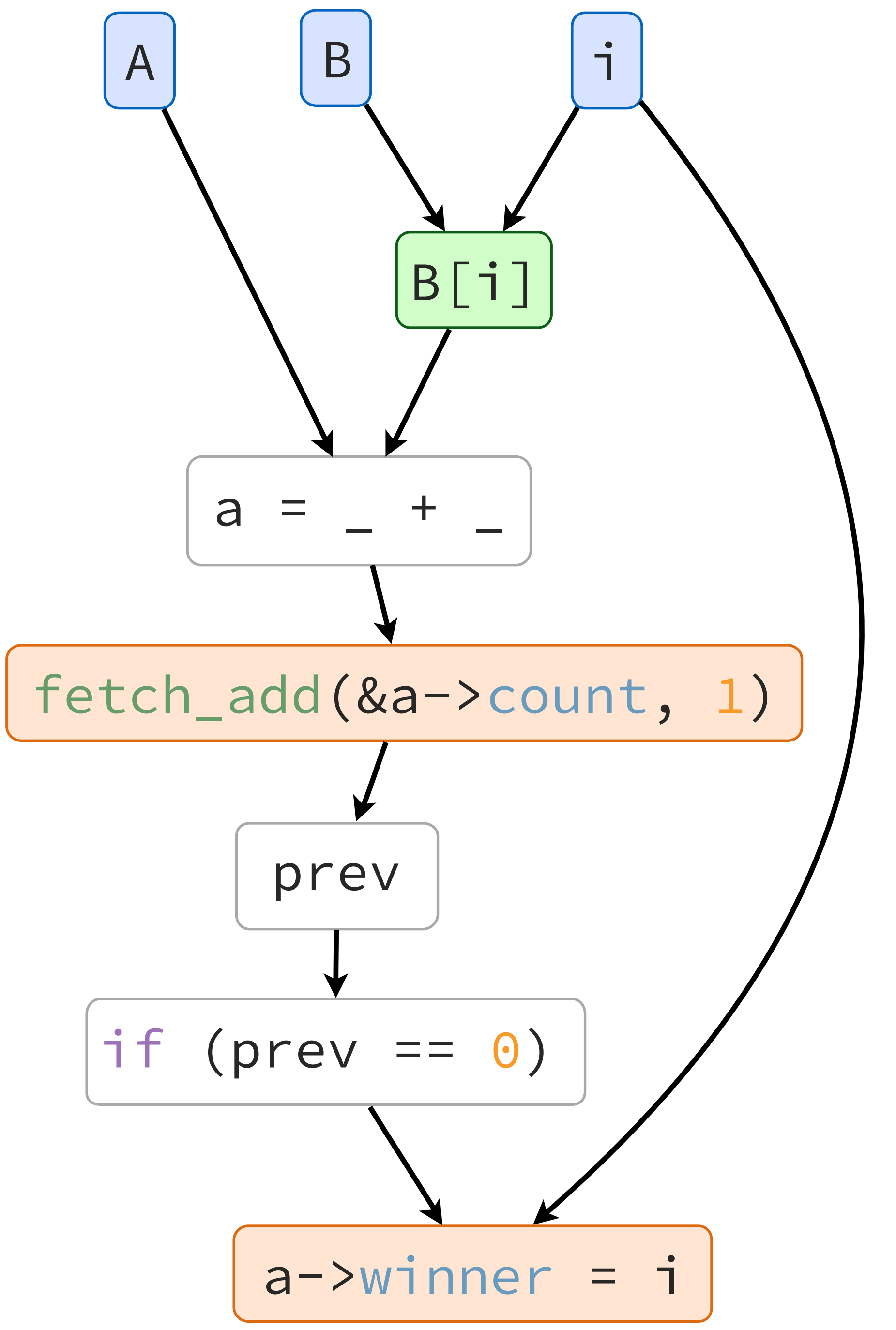

A
B
i
B[i]
a = _ + _
fetch_add(&a->count, 1)
prev
if (prev == 0)
a->winner = i

Region selection (heuristic optimization)



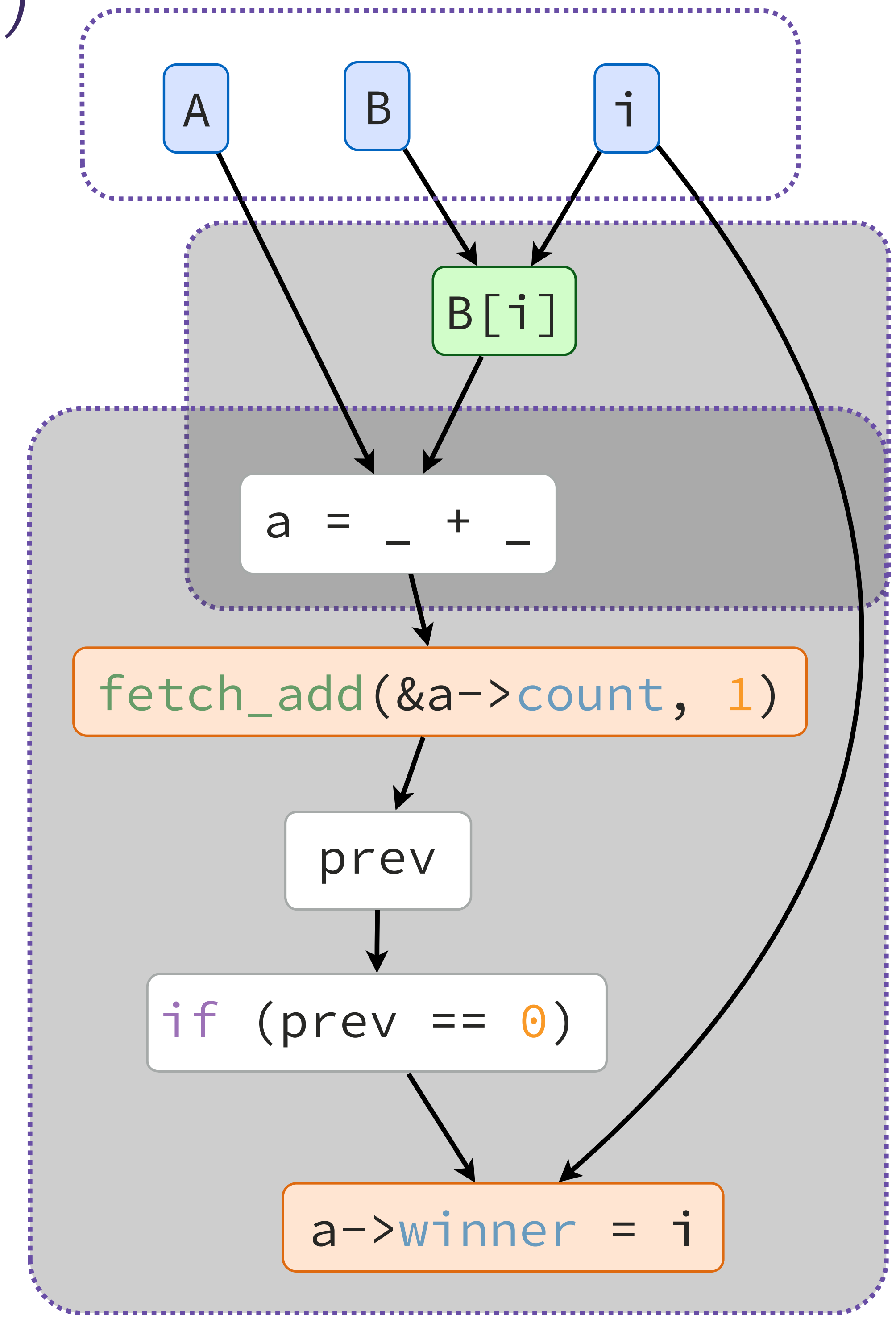
Region selection (heuristic optimization)

- for each anchor, expand a region as far as possible



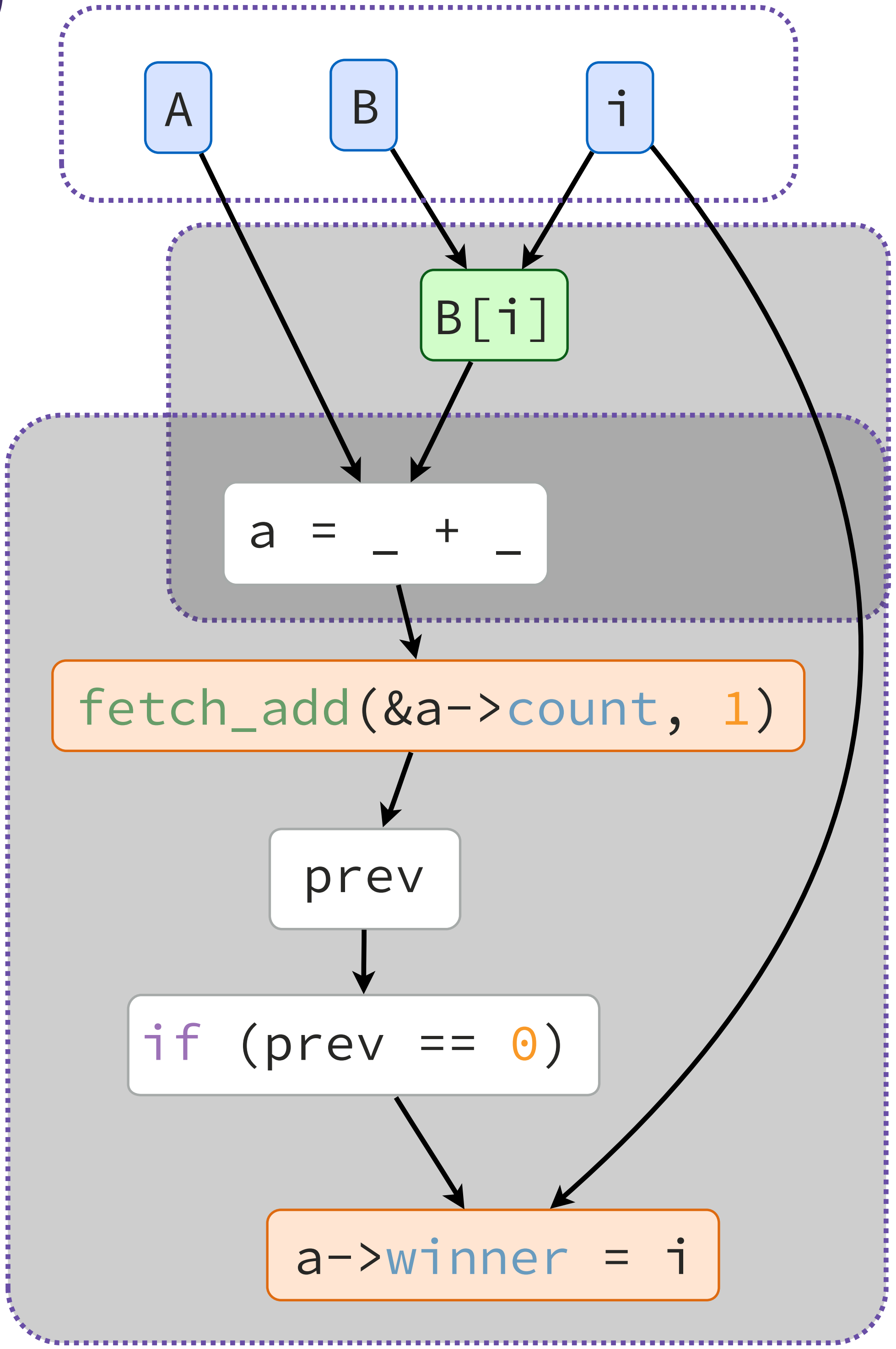
Region selection (heuristic optimization)

- for each anchor, expand a region as far as possible



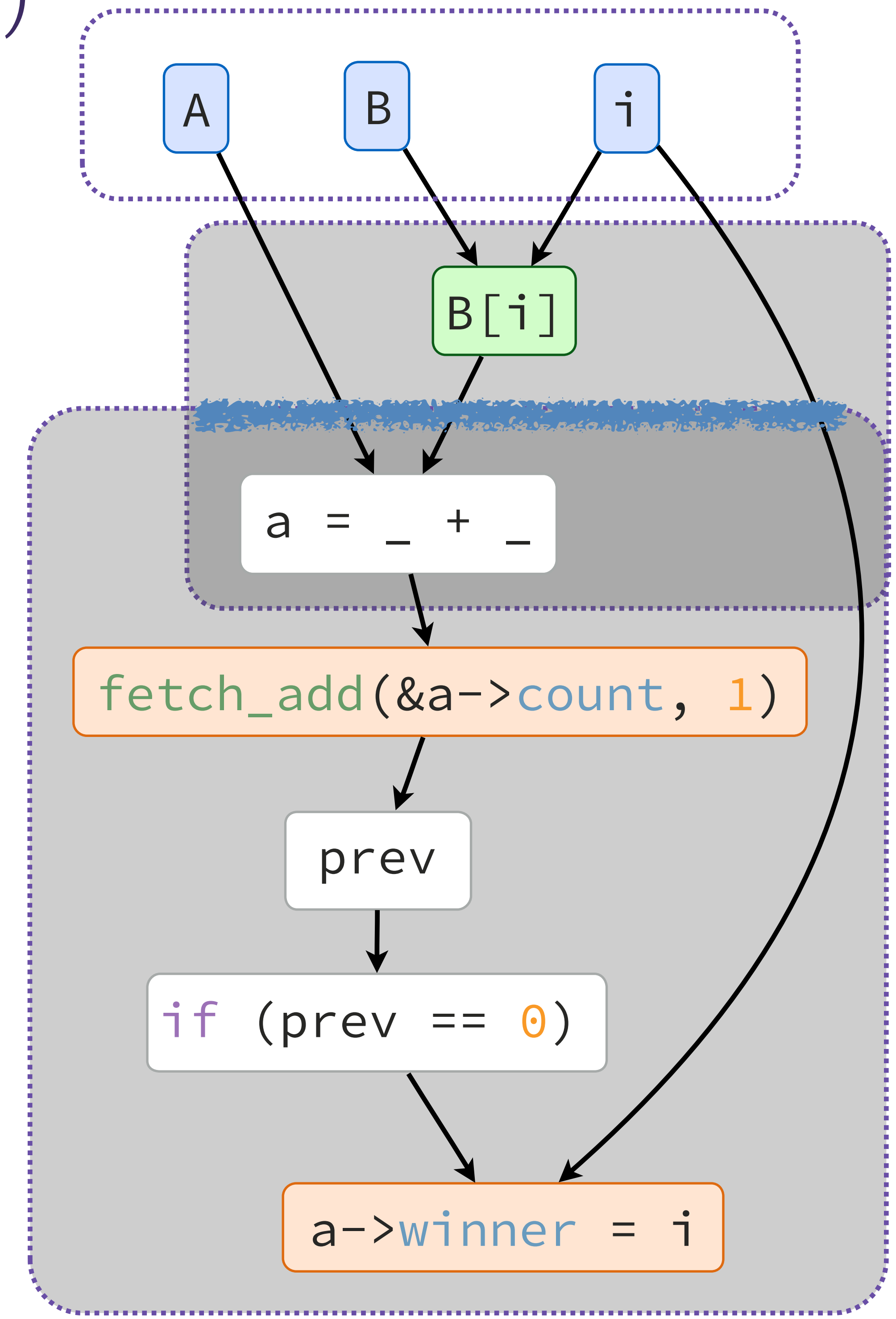
Region selection (heuristic optimization)

- for each anchor, expand a region as far as possible
- at region intersections, compute cost heuristic for the possible choices



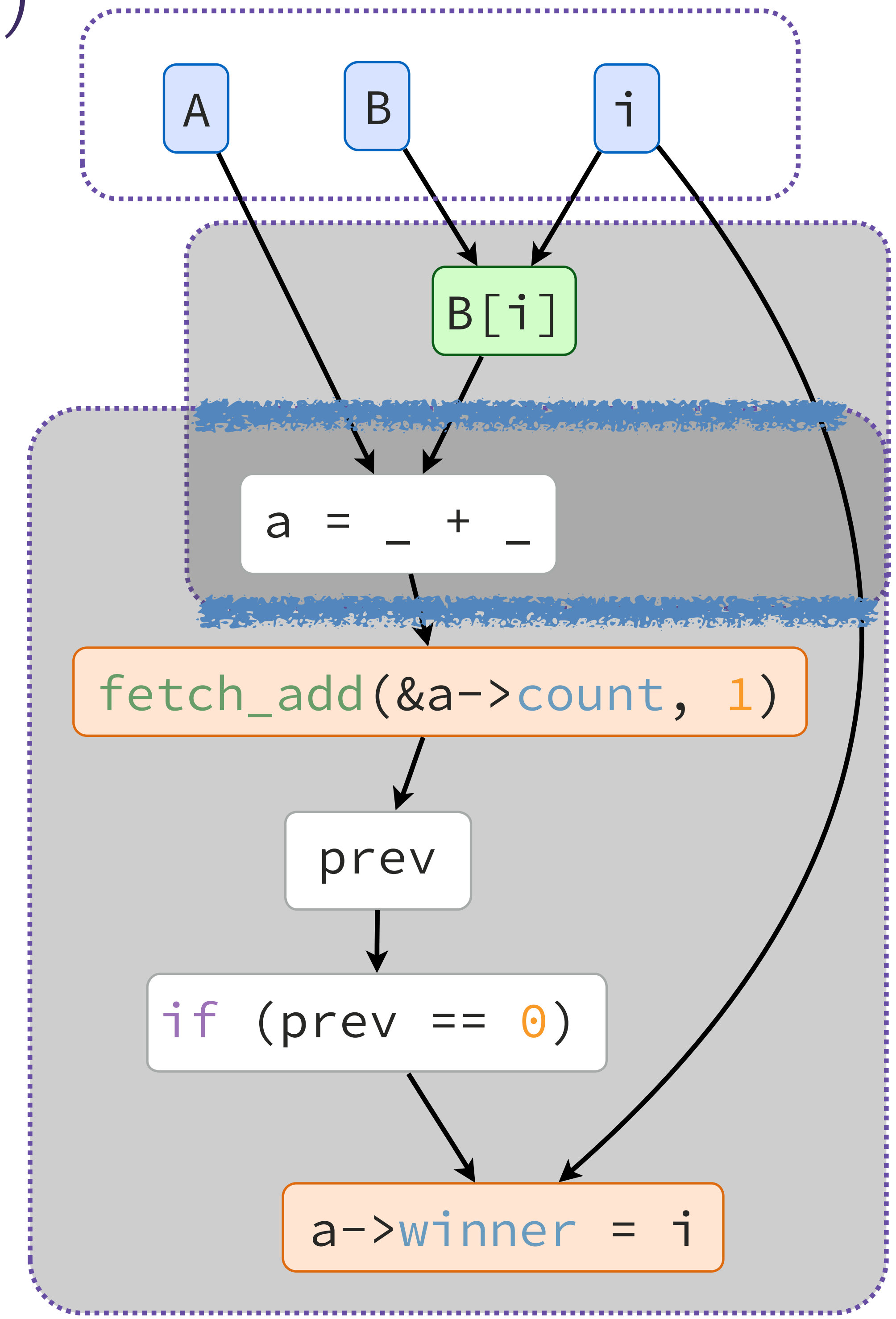
Region selection (heuristic optimization)

- for each anchor, expand a region as far as possible
- at region intersections, compute cost heuristic for the possible choices



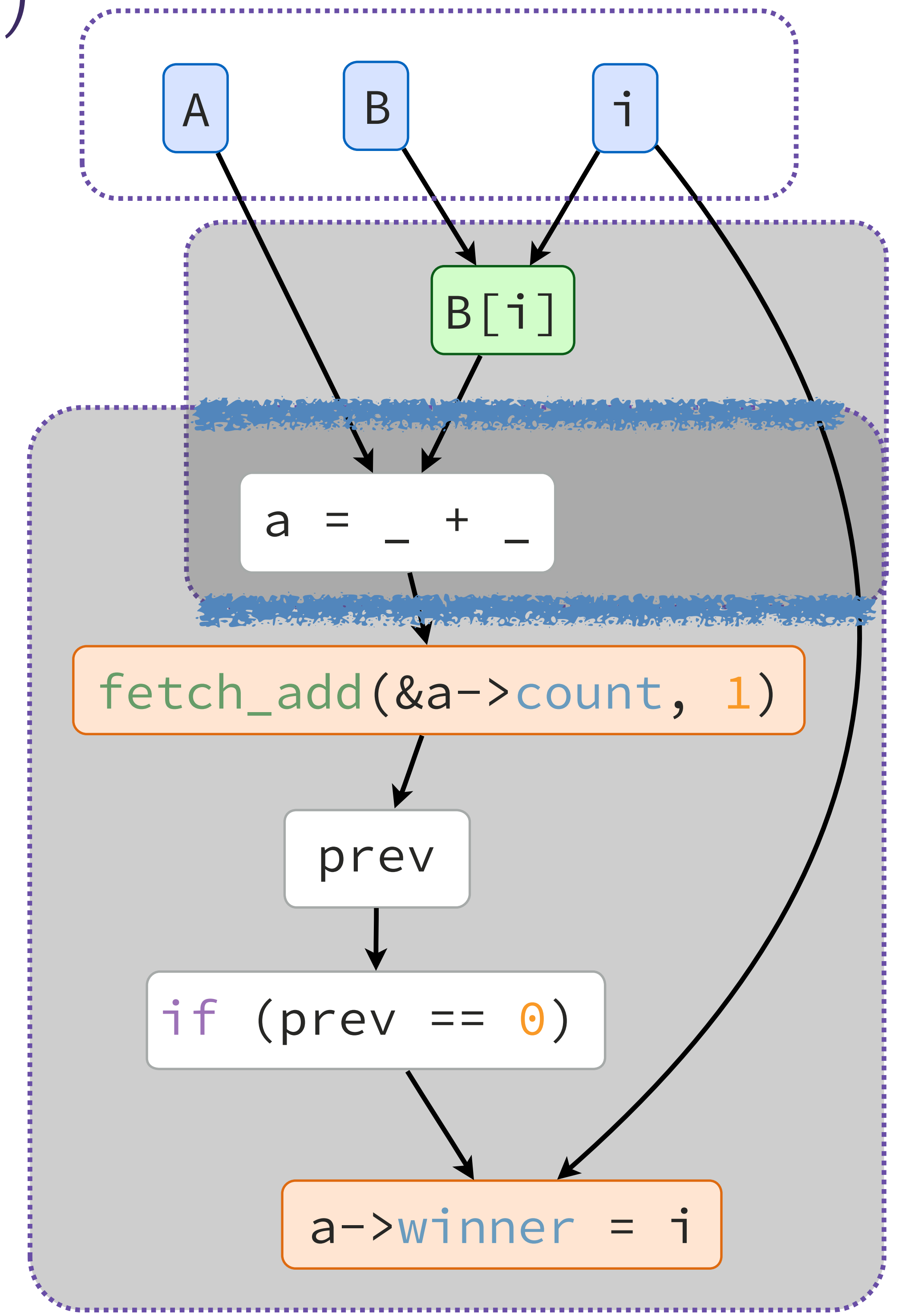
Region selection (heuristic optimization)

- for each anchor, expand a region as far as possible
- at region intersections, compute cost heuristic for the possible choices



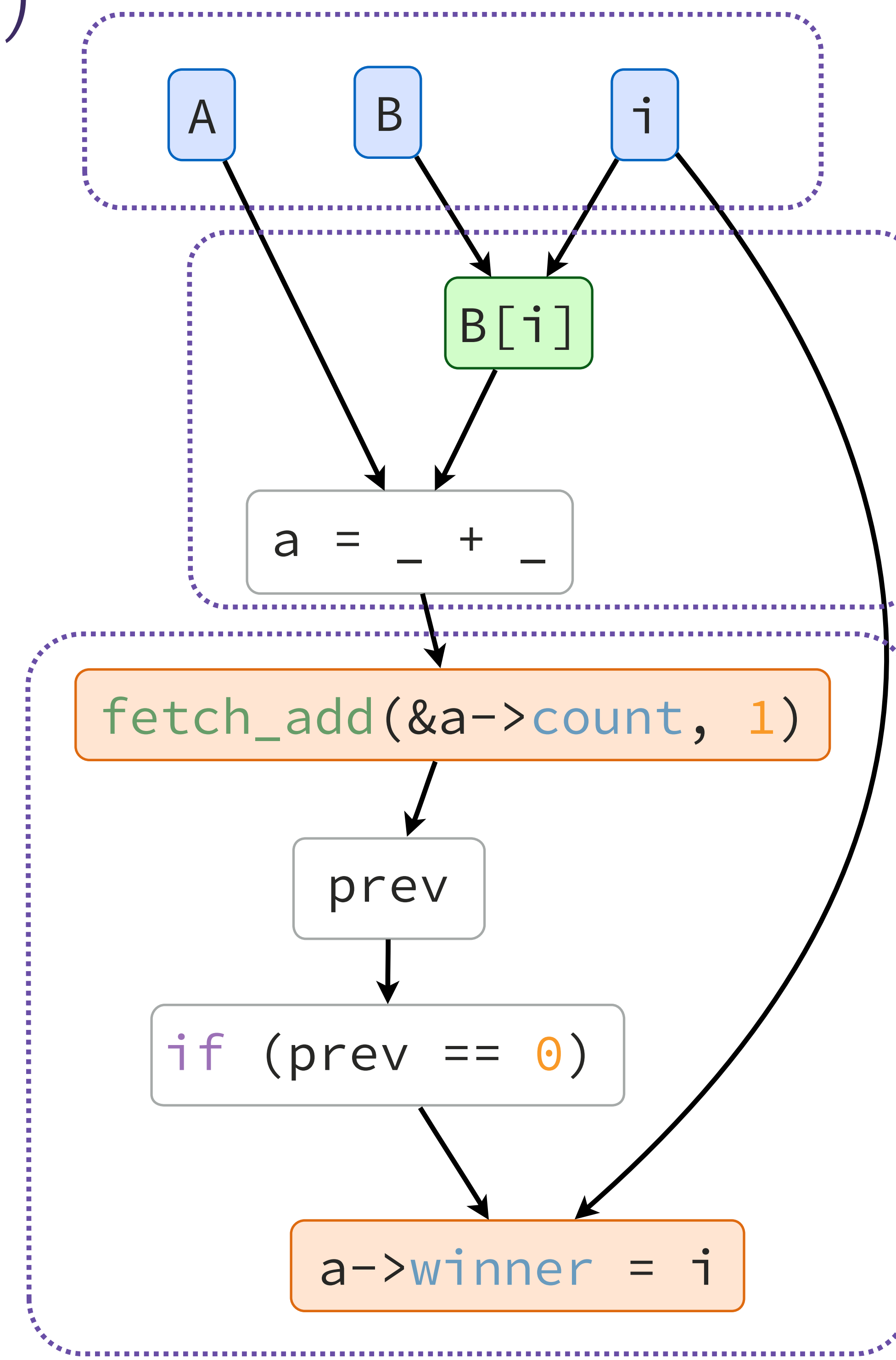
Region selection (heuristic optimization)

- for each anchor, expand a region as far as possible
- at region intersections, compute cost heuristic for the possible choices
- evaluate intersections pair-wise, greedily choose the best in each case



Region selection (heuristic optimization)

- for each anchor, expand a region as far as possible
- at region intersections, compute cost heuristic for the possible choices
- evaluate intersections pair-wise, greedily choose the best in each case



Transform thread to migrate at region boundaries

- create continuations for values that cross regions,
  and pack them into active messages

Transform thread to migrate at region boundaries

- create continuations for values that cross regions, and pack them into active messages

```
[A,B](long i) {
  Counter global* a = A + B[i];
  long prev = fetch_add(&a->count, 1);
  if (prev == 0)    // first to arrive
    a->winner = i; // is winner
}
```

Transform thread to migrate at region boundaries

– create continuations for values that cross regions, and pack them into active messages

```
[A,B](long i) {
  migrate(node(B+i), _);
}
```

```
[A,B,i]{
  Counter global* a = A + B[i];
  migrate(node(a), _);
}
```

```
[a,i]{
  long prev = fetch_add(&a->count, 1);
  if (prev == 0)    // first to arrive
    a->winner = i; // is winner
}
```

Transform thread to migrate at region boundaries

- create continuations for values that cross regions, and pack them into active messages

```
[A,B](long i) {
  migrate(node(B+i), _);
}
```

```
[A,B,i]{
  Counter global* a = A + B[i];
  migrate(node(a), _);
}
```

```
[a,i]{
  long prev = fetch_add(&a->count, 1);
  if (prev == 0)    // first to arrive
    a->winner = i; // is winner
}
```

Alembic

Static optimizing migration algorithm

- Constrained by anchor points
- Greedy heuristic to reduce communication

Implementation for C++ in LLVM

Evaluation

- ***6x better*** than naive compiler-generated communication
- 82% of hand-tuned performance

Implementation

C++ extensions to support global pointers

Anchor point / locality partitioning analysis pass

Region selection and continuation-passing transform pass

Alembic

Static optimizing migration algorithm

- Constrained by anchor points
- Greedy heuristic to reduce communication

Implementation for C++ in LLVM

Evaluation

- ***6x better*** than naive compiler-generated communication
- 82% of hand-tuned performance

Benchmarks

- Ported Grappa applications (irregular, data-intensive, …)

  BFS

  Pagerank

  Connected Components

  Intsort

Performance (12 nodes)

- naive put/get compiler-generated communication
- hand-tuned migration decisions
- Alembic-generated migrations

Benchmarks

- Ported Grappa applications (irregular, data-intensive, …)

  BFS

  Pagerank

  Connected Components

  Intsort

```
GlobalHashSet symmetric* set;
Graph symmetric* g;

void explore(VertexID r, color_t color) {
  Vertex global* vs = g->vertices();
  phaser.enroll(vs[r].nadj)
  forall<async>(adj(g,vs+r), [=](VertexID j){
    auto& v = vs[j];
    if (cmp_swap(&v.color, -1, color)){
      spawn([=]{ explore(j, color); });
    } else if (v.color != color) {
      Edge edge(color, v.color);
      set->insert(edge);
      phaser.complete(1);
    }
  });
  phaser.complete(1);
}
```

Performance (12 nodes)

- naive put/get compiler-generated communication
- hand-tuned migration decisions
- Alembic-generated migrations

Benchmarks

- Ported Grappa applications (irregular, data-intensive, ...)

  BFS

  Pagerank

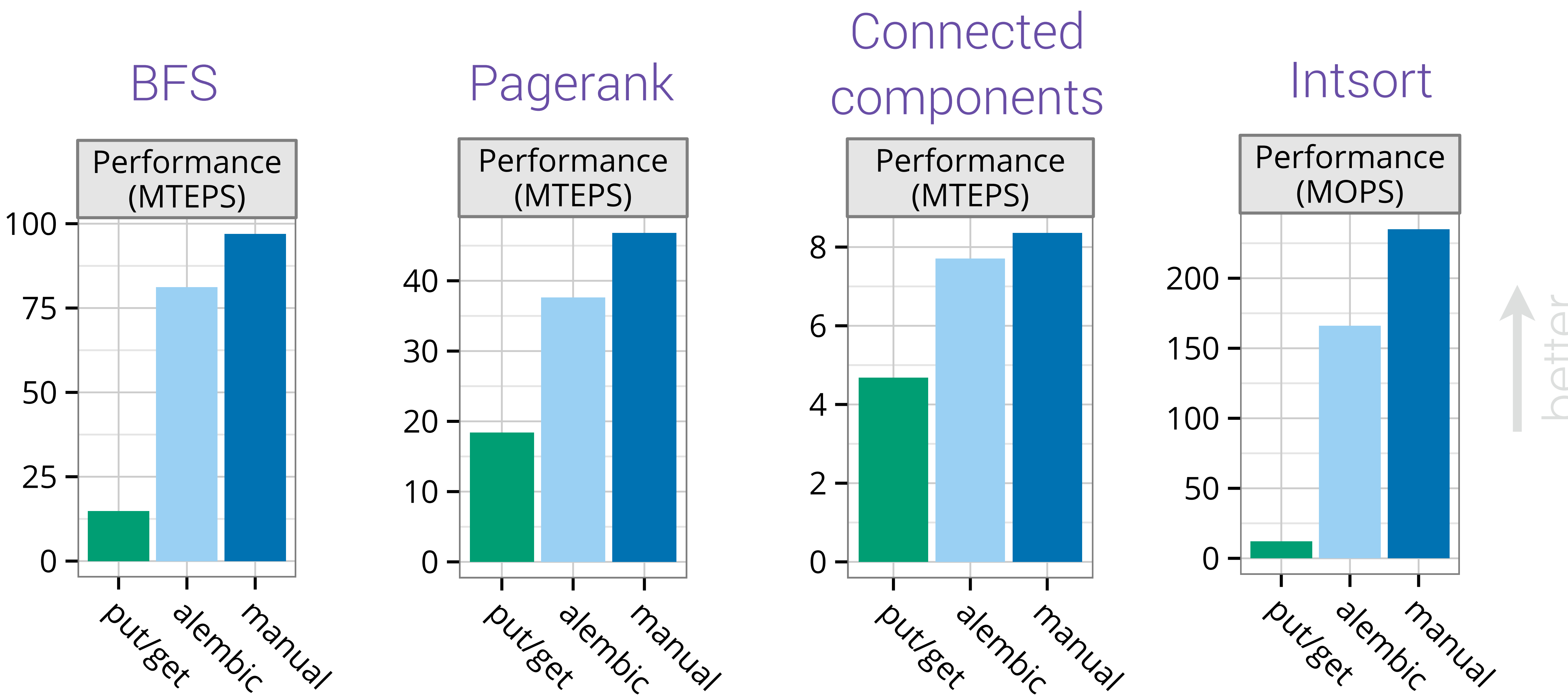
  Connected Components

  Intsort

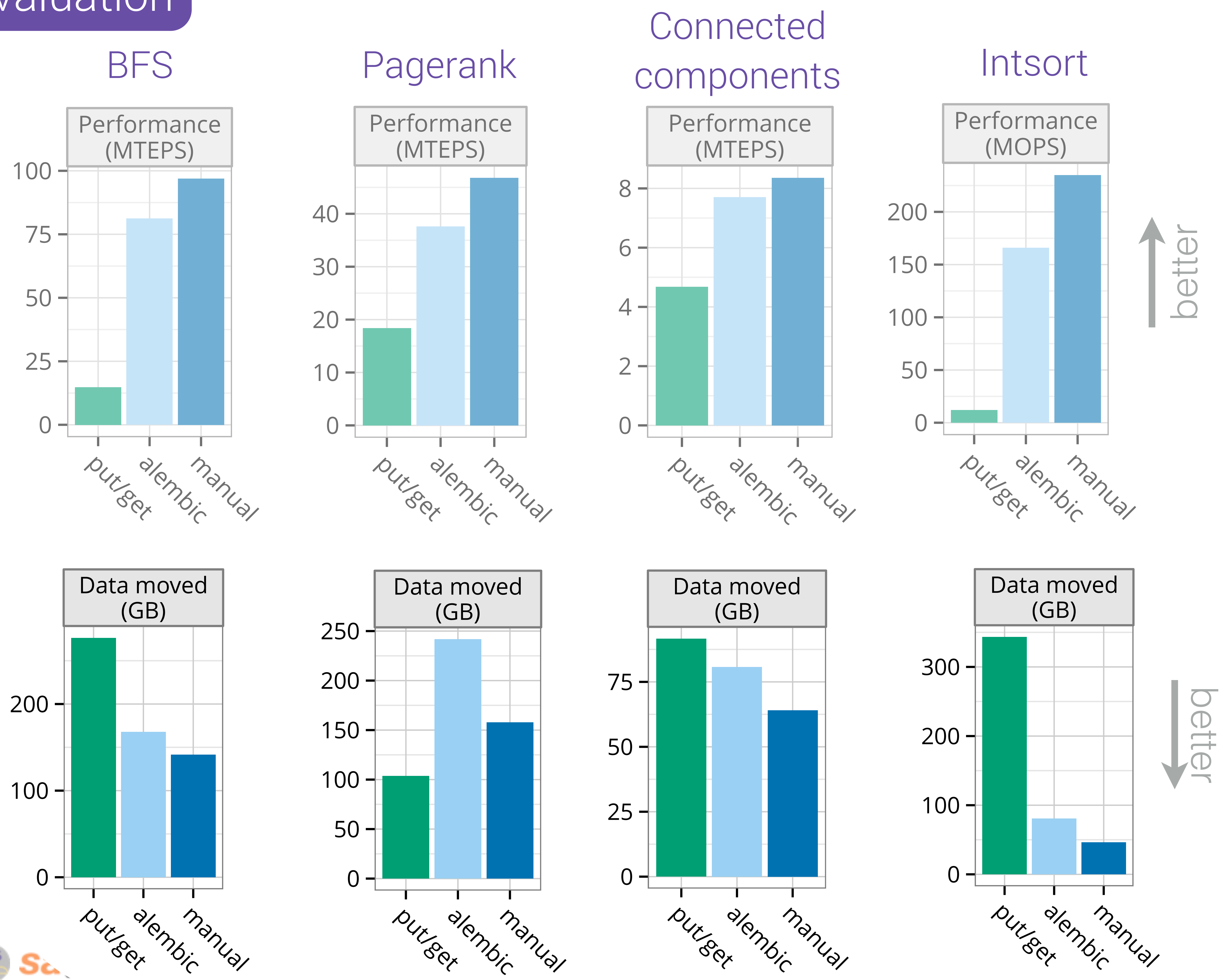
```
GlobalHashSet symmetric* set;
Graph symmetric* g;

void explore(VertexID r, color_t color) {
  Vertex global* vs = g->vertices();
  phaser.enroll(vs[r].nadj);
  forall<async>(adj(g,vs+r), [=](VertexID j){
    auto& v = vs[j];
    if (cmp_swap(&v.color, -1, color)){
      spawn([=]{ explore(j, color); });
    } else if (v.color != color) {
      Edge edge(color, v.color);
      set->insert(edge);
      phaser.complete(1);
    }
  });
  phaser.complete(1);
}
```

Performance (12 nodes)

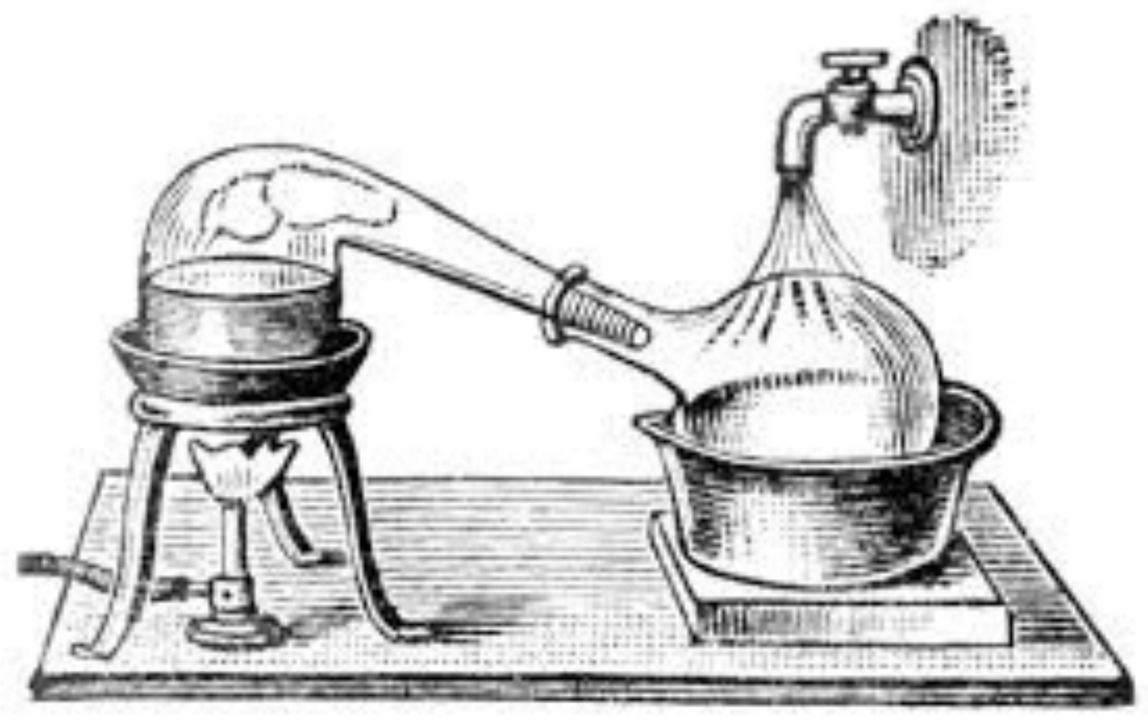
- naive put/get compiler-generated communication
- hand-tuned migration decisions
- Alembic-generated migrations



BFS
Pagerank
Connected components
Intsort
Performance (MTEPS)
Performance (MTEPS)
Performance (MTEPS)
Performance (MOPS)
100
75
50
25
0
40
30
20
10
0
8
6
4
2
0
200
150
100
50
0
put/get
alembic
manual
better



BFS
Pagerank
Connected components
Intsort
Performance (MTEPS)
Performance (MOPS)
better
Data moved (GB)
put/get
alembic
manual



Alembic

Algorithm to make automatic migration decisions

- Analyze locality by partitioning *anchors*
- Greedy optimization to reduce communication cost heuristic

LLVM implementation for Grappa C++

Performance — near hand-tuned, much better than PGAS baseline

Brandon Holt, Preston Briggs, Luis Ceze, Mark Oskin