Flat Combining Synchronized Global Data Structures

Brandon Holt, Jacob Nelson, Brandon Myers,
Preston Briggs, Luis Ceze, Simon Kahan, Mark Oskin

W UNIVERSITY *of* WASHINGTON

contention → cooperation

simple, distributed,
batched synchronization

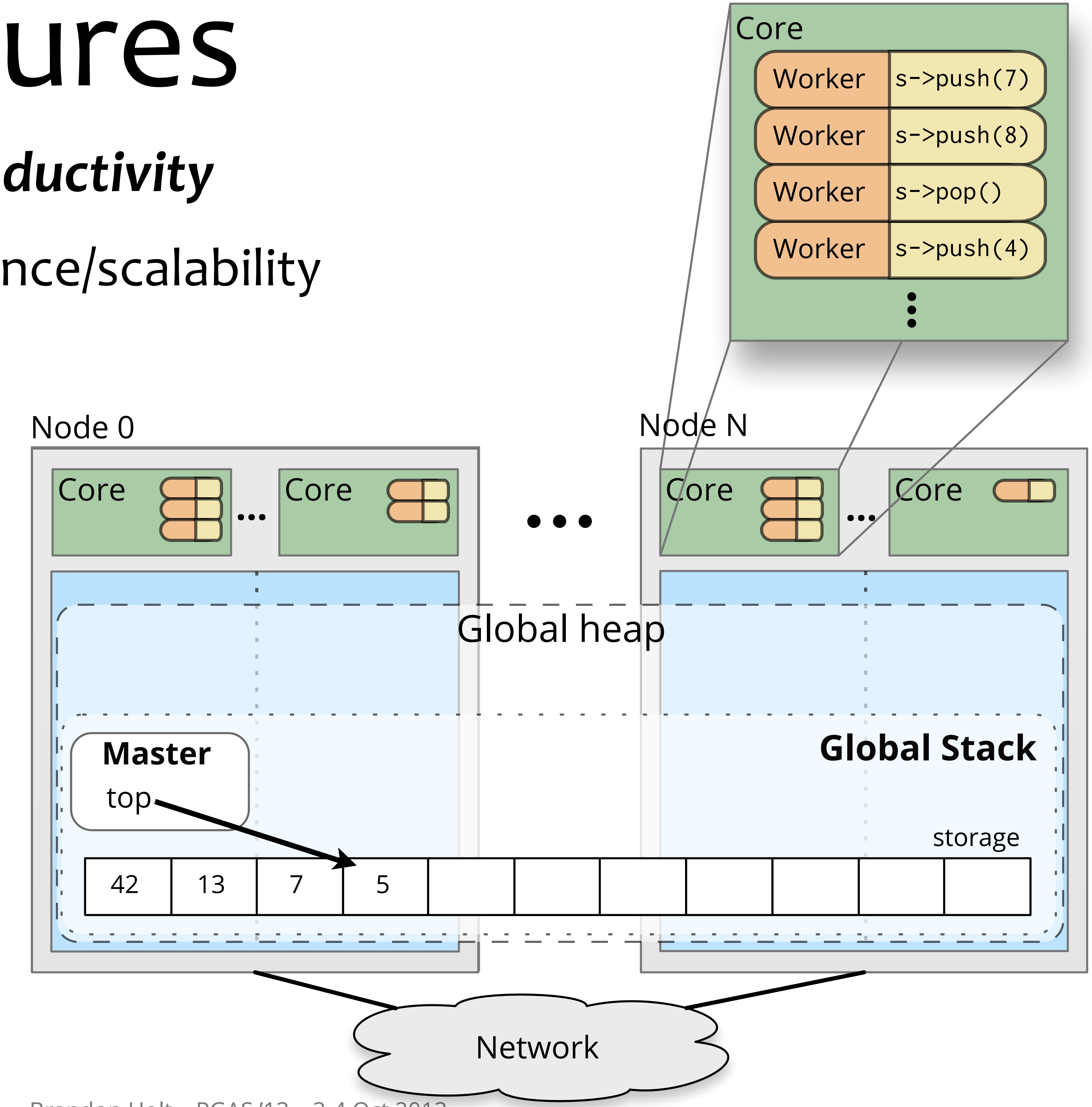
sequential consistency
at cluster scale

© Disney, Inc. *Fantasia (The Pastoral Symphony)*

synchronized shared data structures

Standard library aids ***productivity***

Generality costs performance/scalability

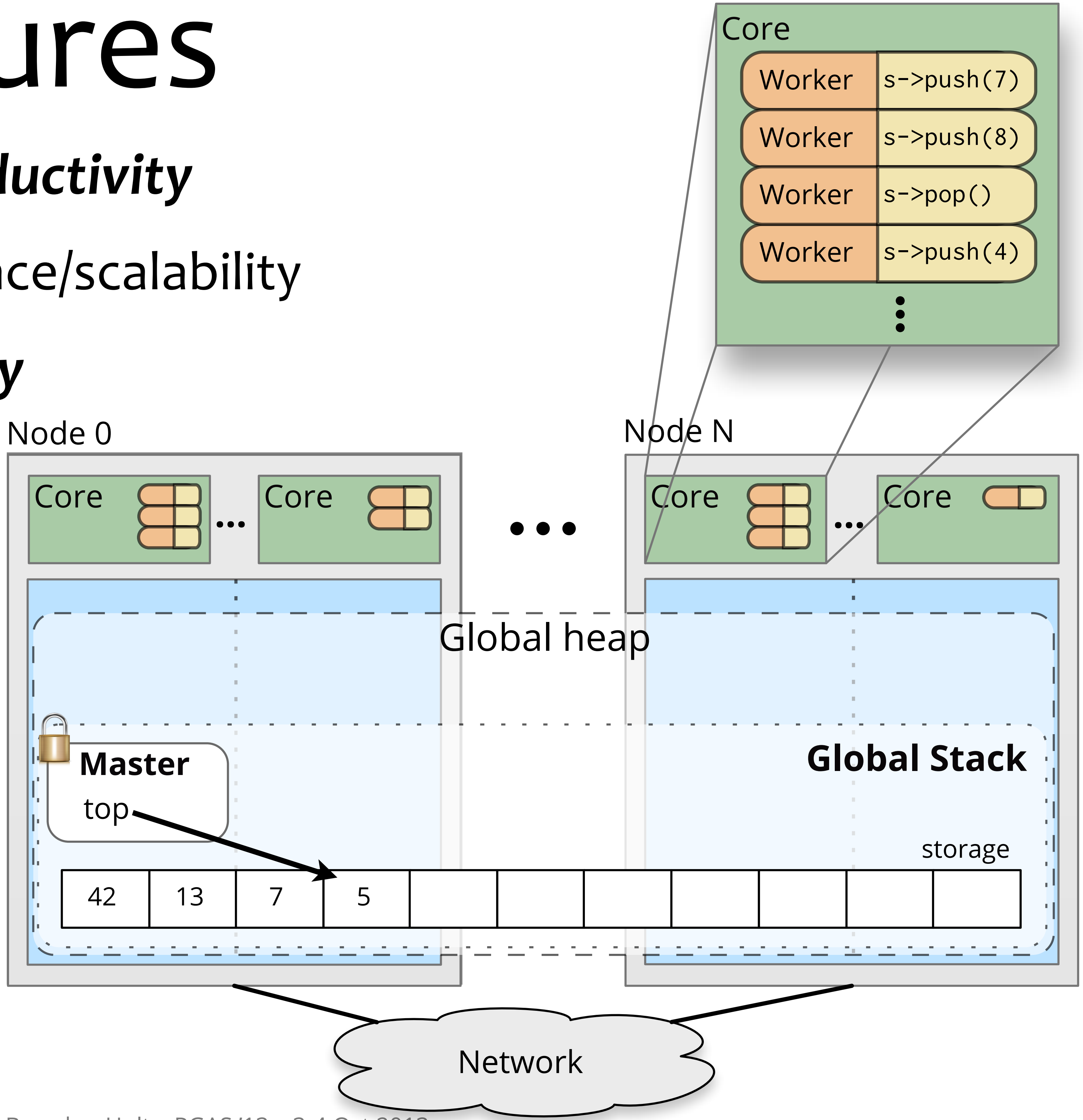


synchronized shared data structures

Standard library aids ***productivity***

Generality costs performance/scalability

Must maintain ***consistency***

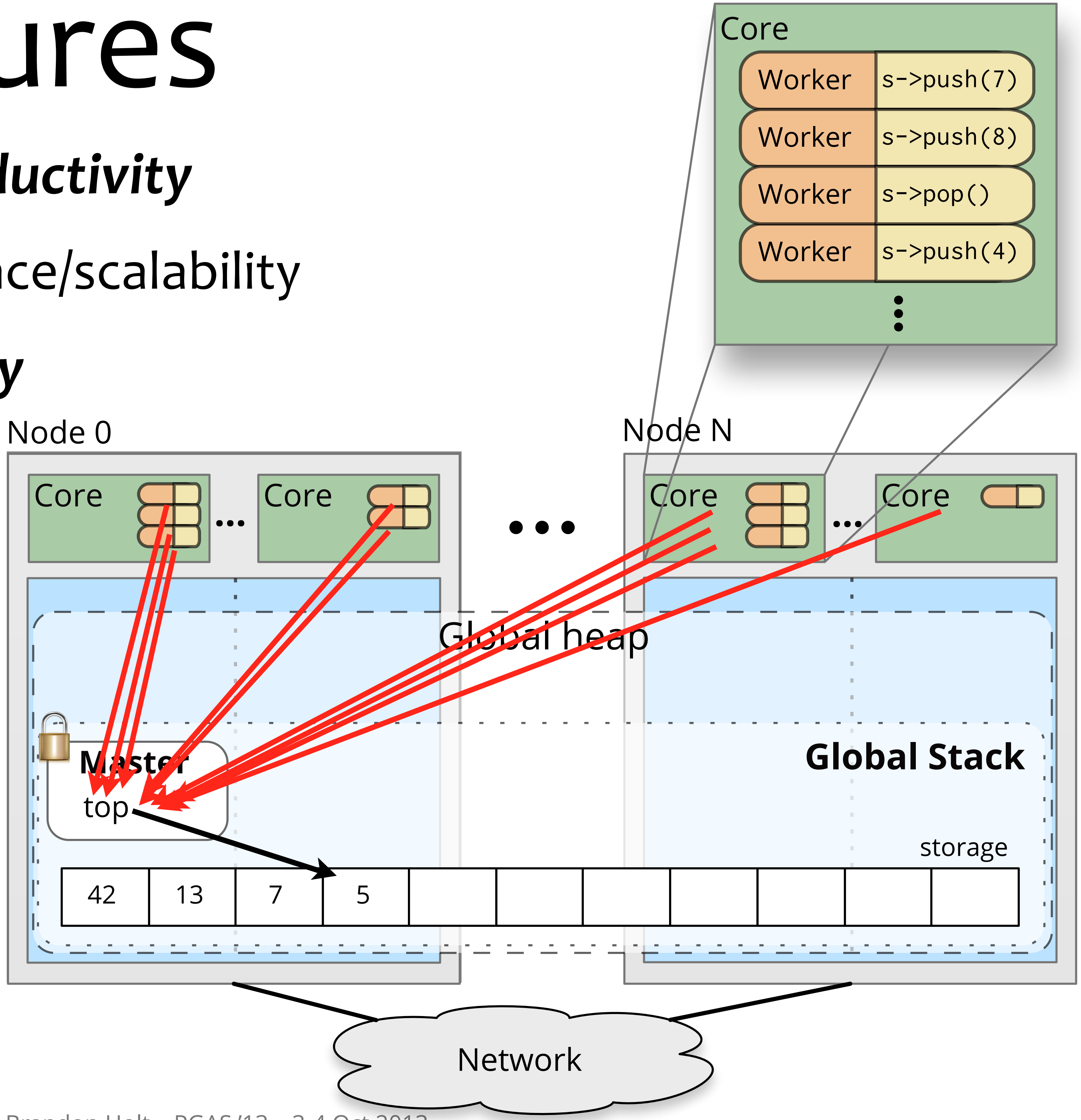


synchronized shared data structures

Standard library aids ***productivity***

Generality costs performance/scalability

Must maintain ***consistency***

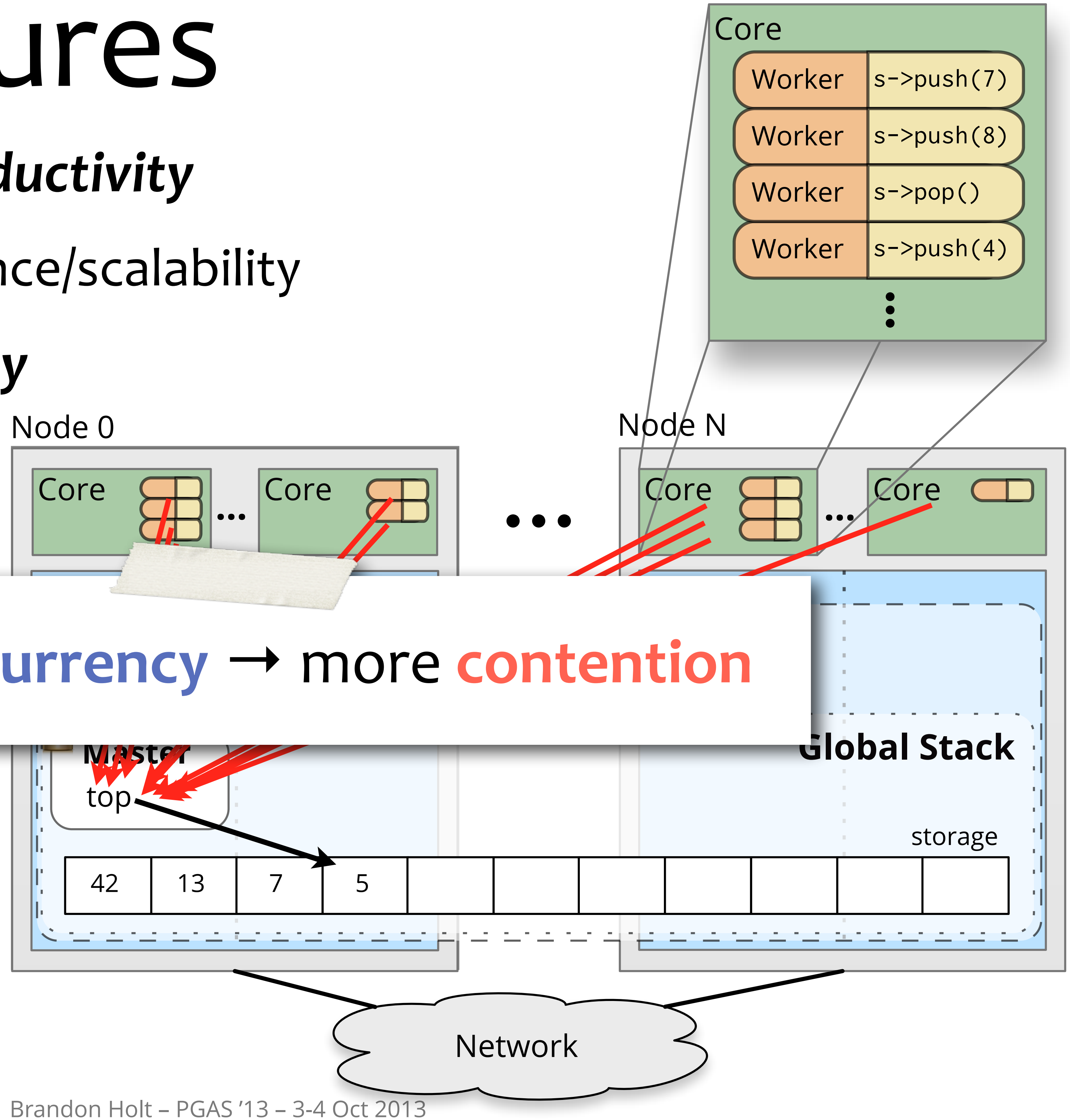


synchronized shared data structures

Standard library aids ***productivity***

Generality costs performance/scalability

Must maintain ***consistency***



more **concurrency** → more **contention**

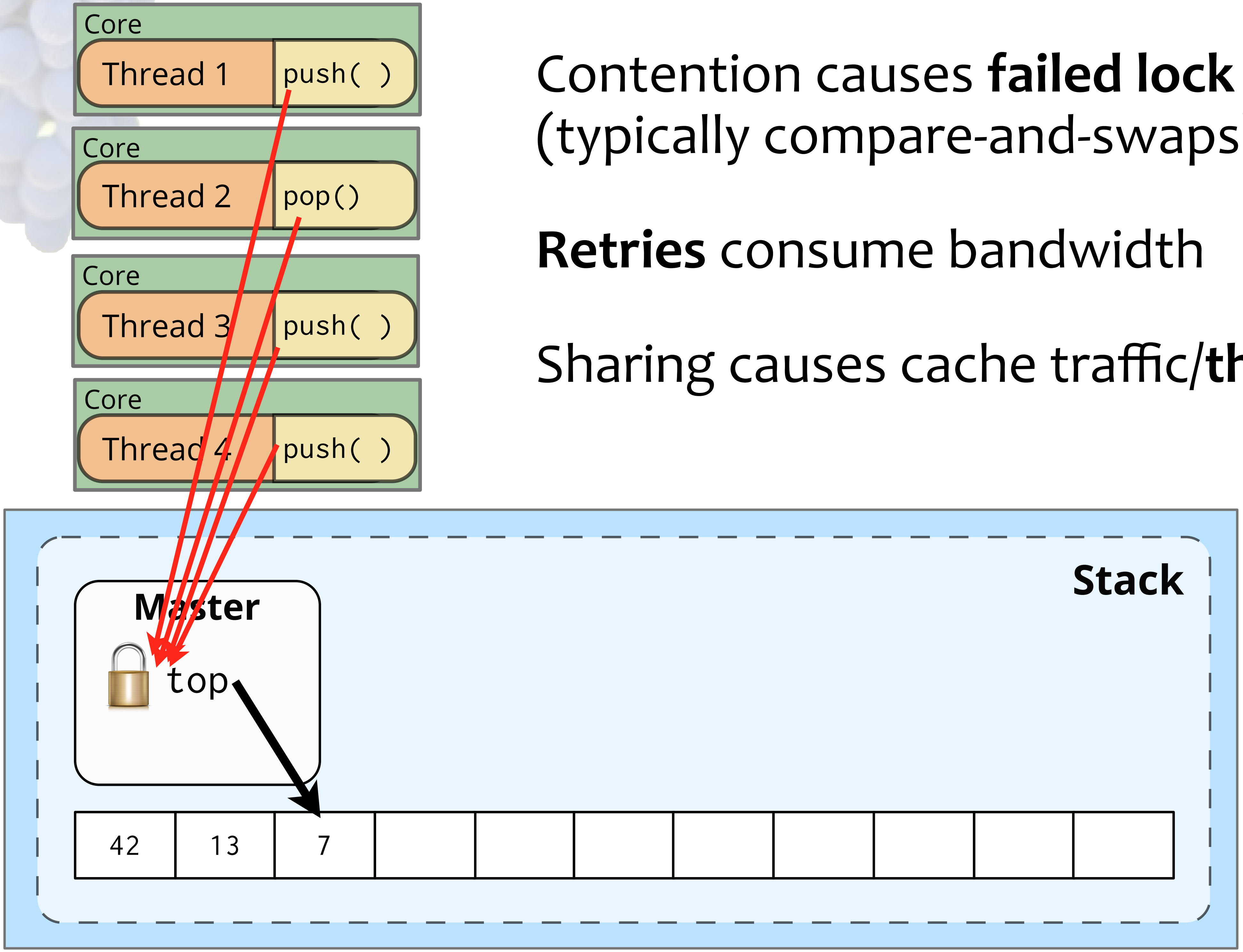
contention → cooperation

contention → cooperation

contention → cooperation

contention → cooperation

contention: global lock

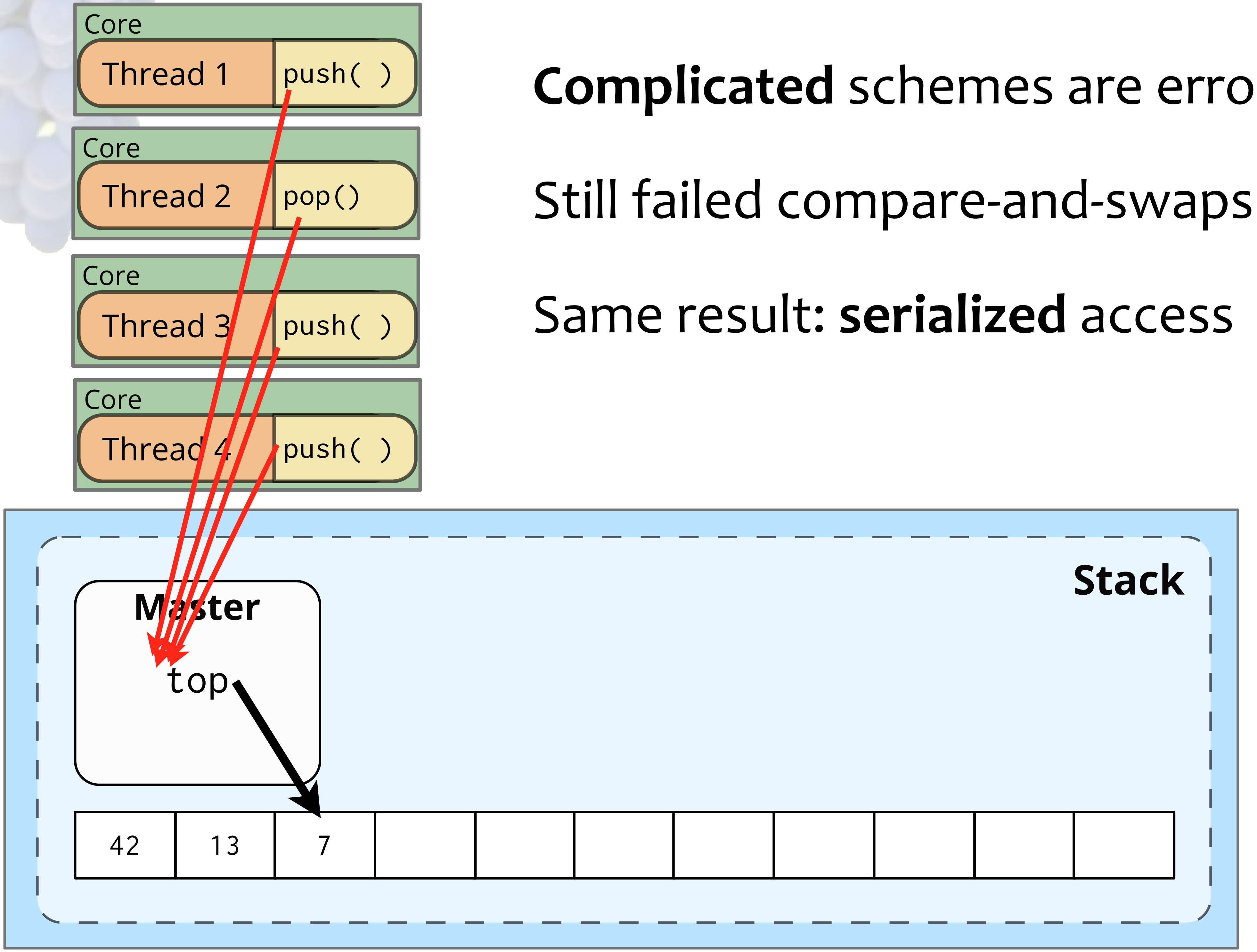


Contention causes **failed lock acquires** (typically compare-and-swaps)

Retries consume bandwidth

Sharing causes cache traffic/**thrashing**

contention: fine-grained sync

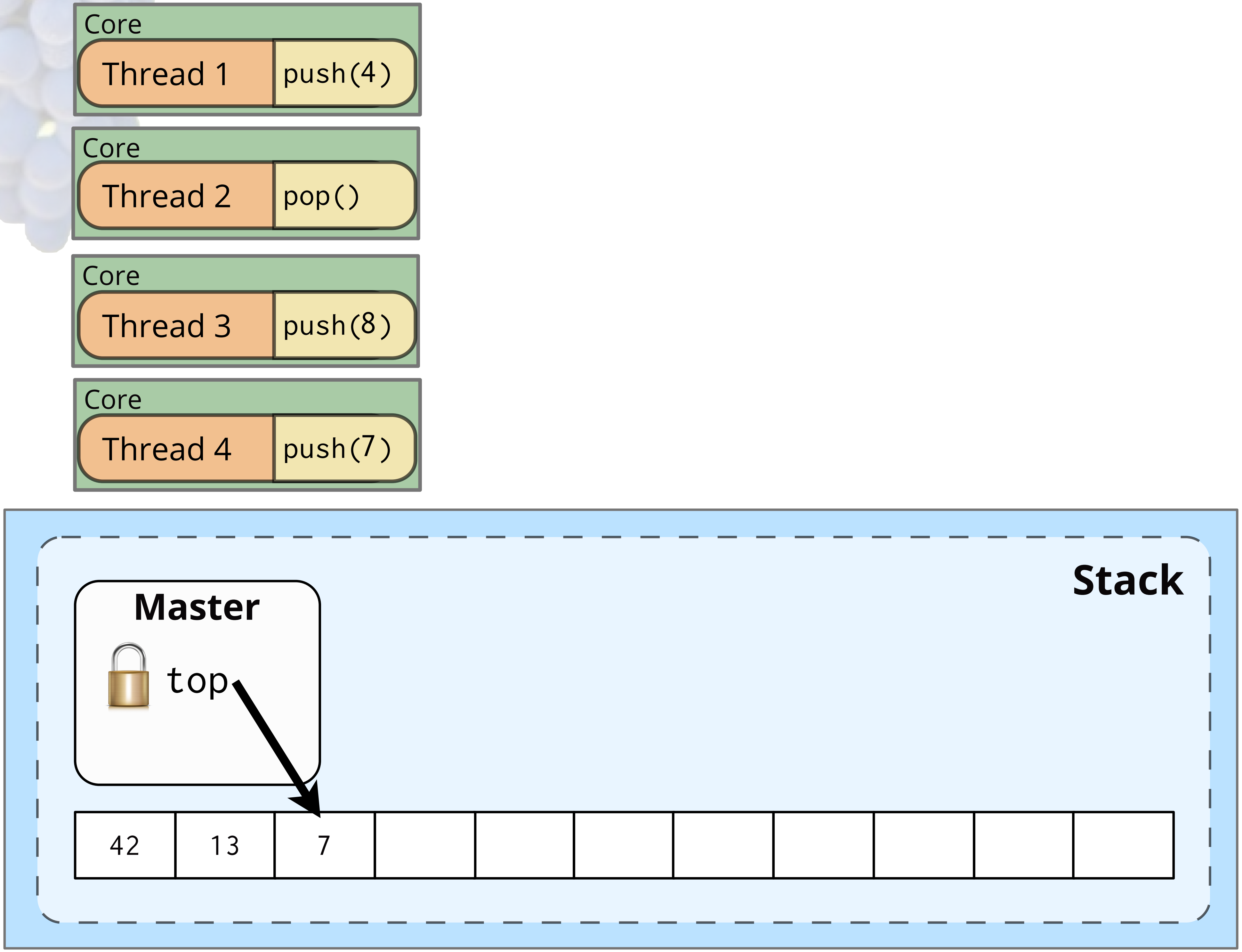


Complicated schemes are error-prone

Still failed compare-and-swaps and **retries**

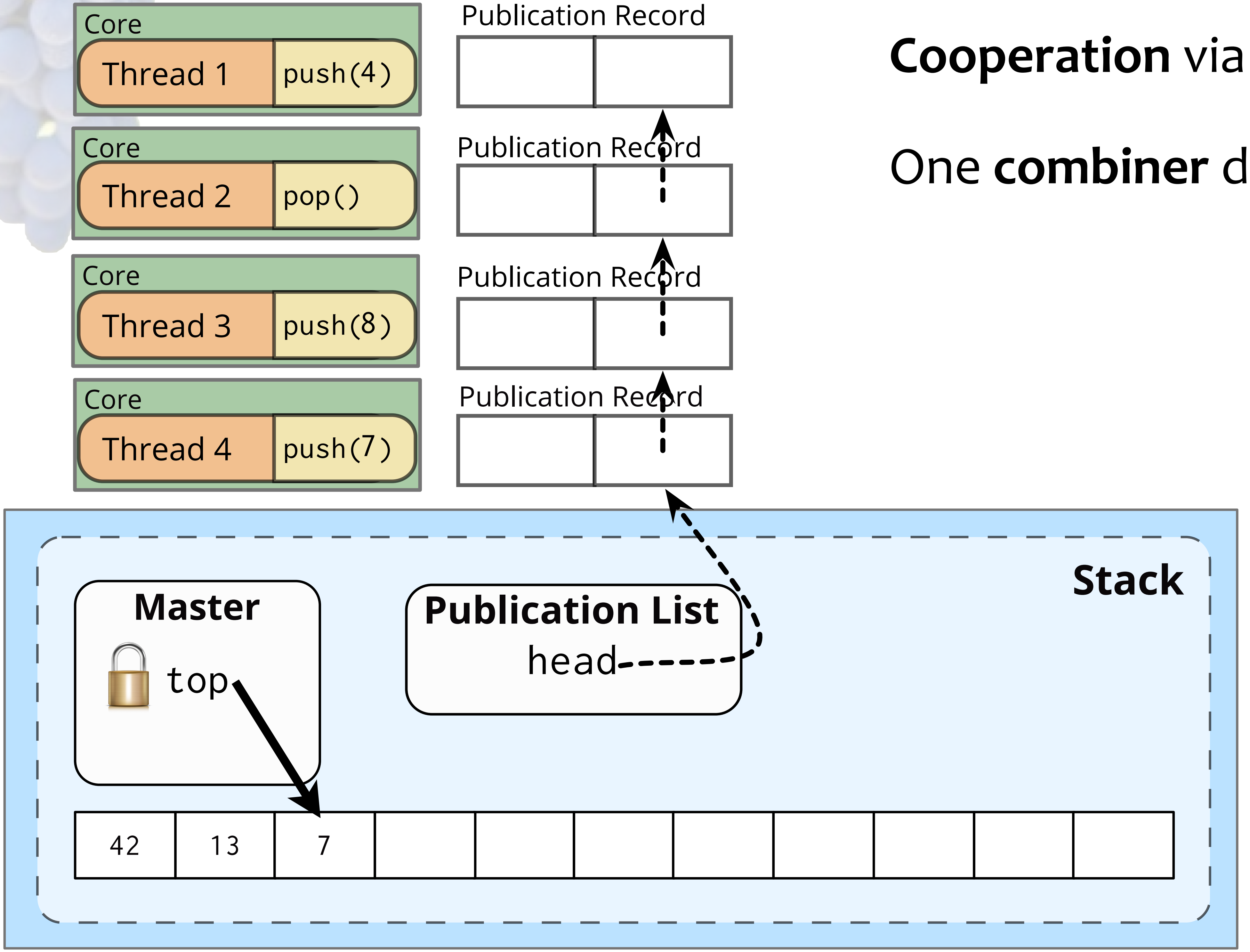
Same result: **serialized** access

cooperation: flat combining [1]



[1] *"Flat Combining and the Synchronization-Parallelism Tradeoff"* Danny Hendler, Itai Incze, Nir Shavit, and Moran Tzafrir (SPAA '10)

cooperation: flat combining [1]

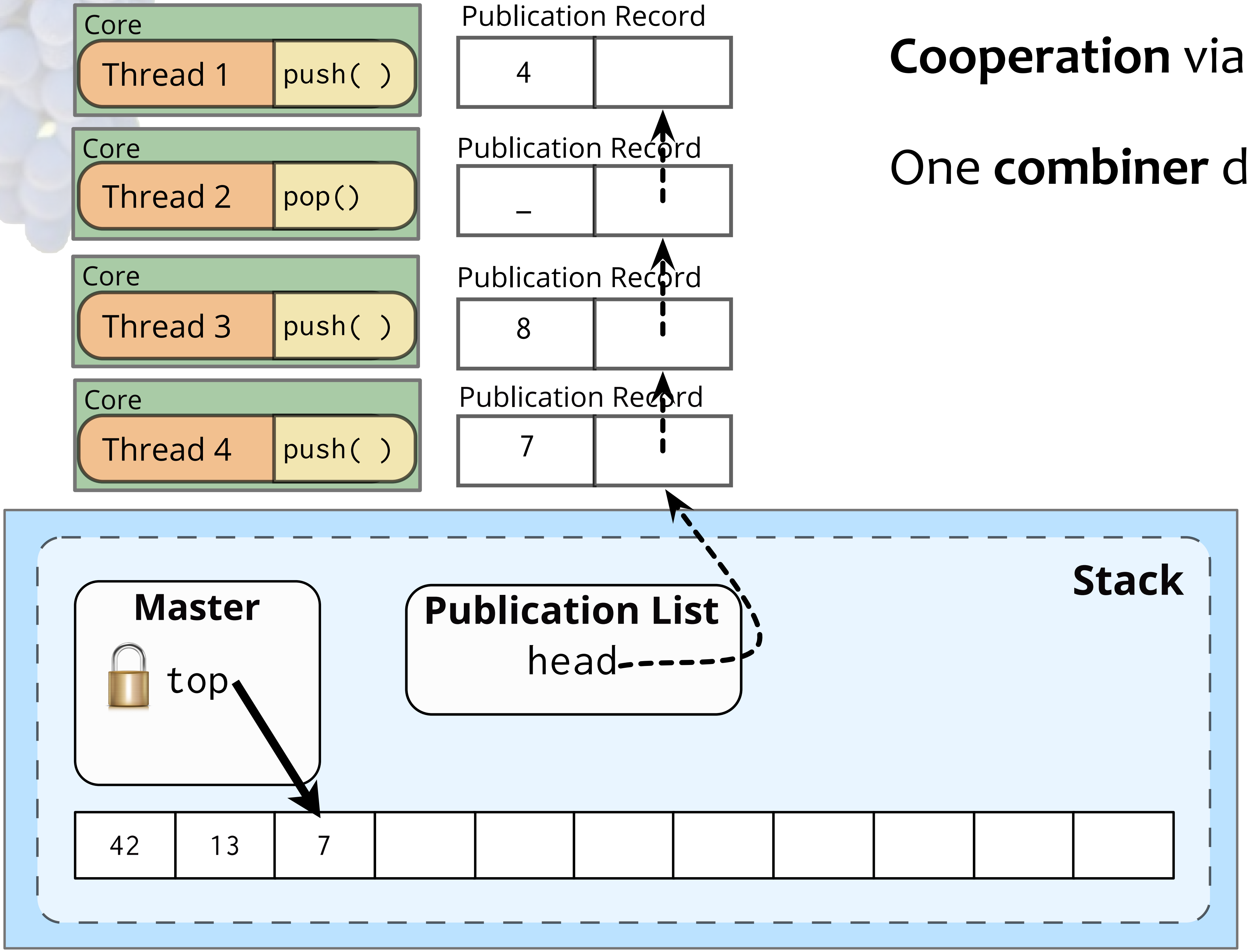


Cooperation via publication list

One **combiner** does all the work

[1] *"Flat Combining and the Synchronization-Parallelism Tradeoff"* Danny Hendler, Itai Incze, Nir Shavit, and Moran Tzafrir (SPAA '10)

cooperation: flat combining[1]

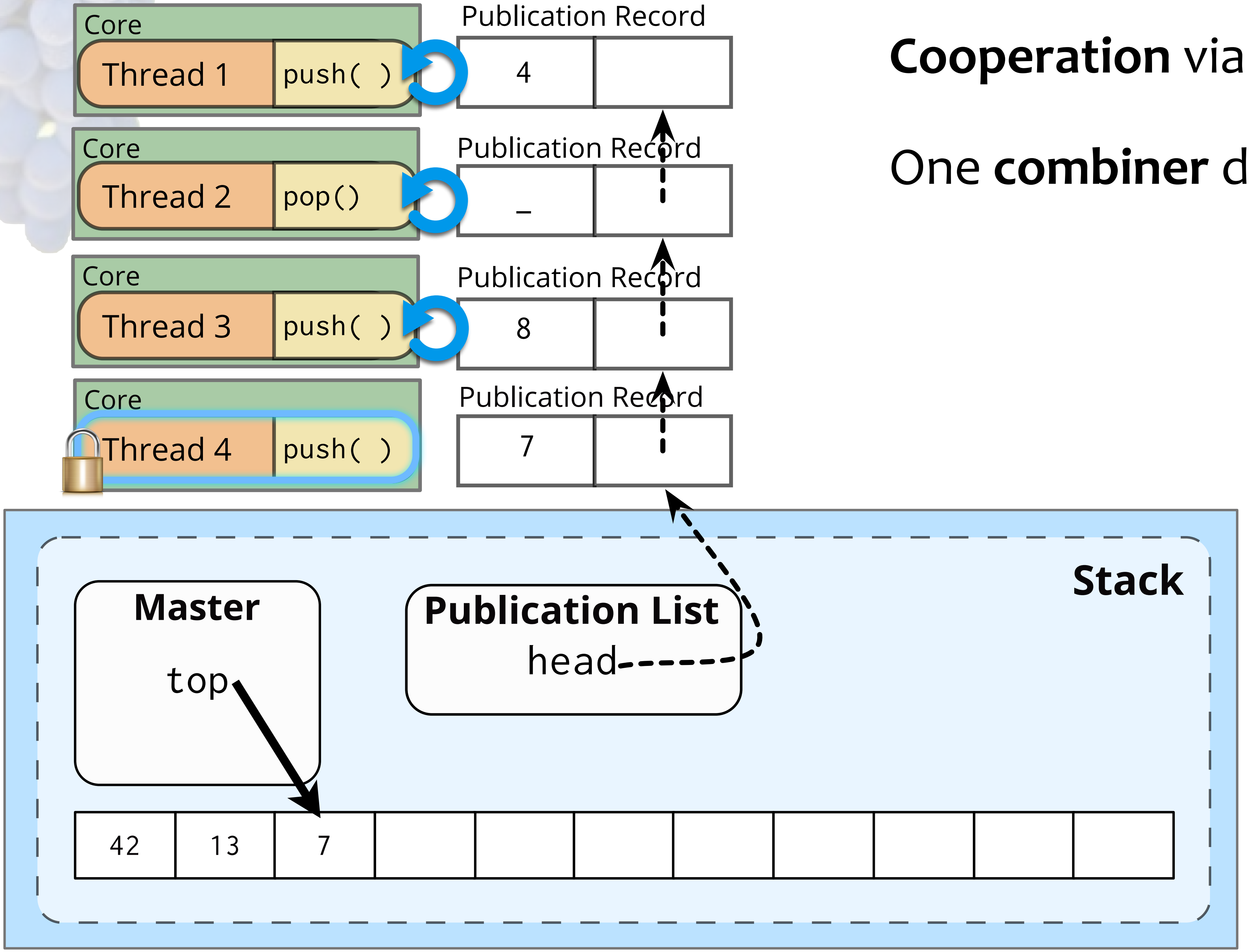


Cooperation via publication list

One **combiner** does all the work

[1] *"Flat Combining and the Synchronization-Parallelism Tradeoff"* Danny Hendler, Itai Incze, Nir Shavit, and Moran Tzafrir (SPAA '10)

cooperation: flat combining [1]

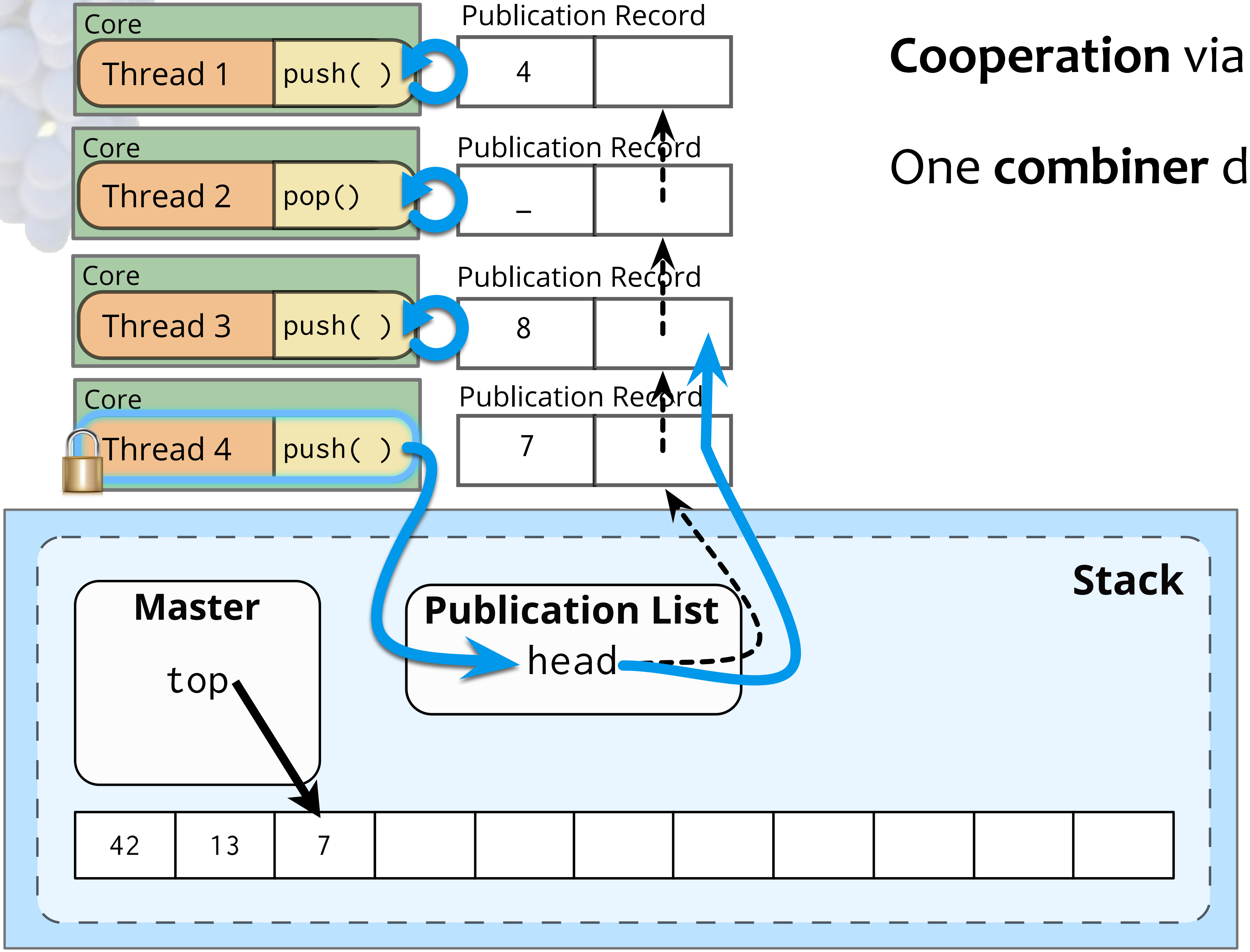


Cooperation via publication list

One **combiner** does all the work

[1] *"Flat Combining and the Synchronization-Parallelism Tradeoff"* Danny Hendler, Itai Incze, Nir Shavit, and Moran Tzafrir (SPAA '10)

cooperation: flat combining [1]

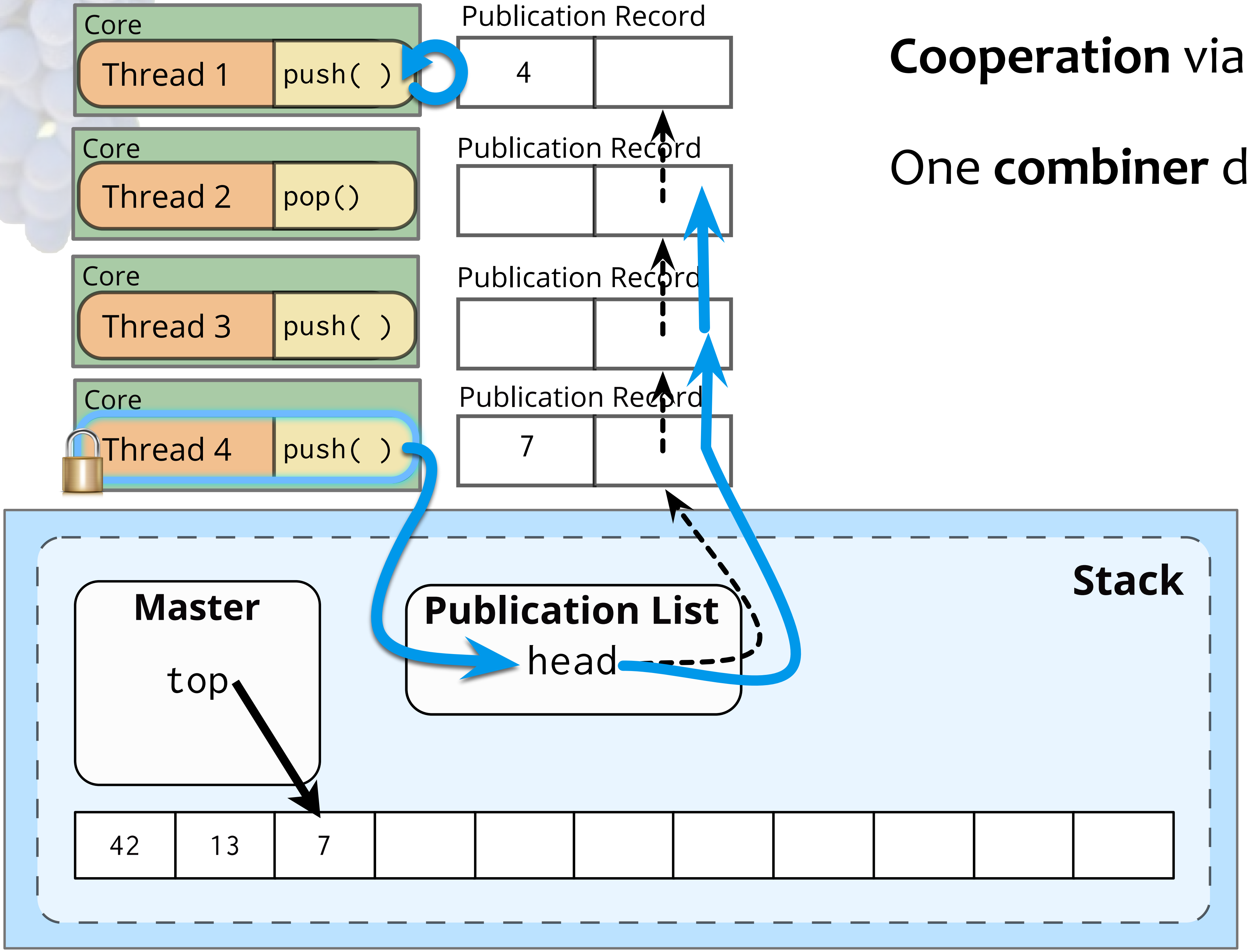


Cooperation via publication list

One **combiner** does all the work

[1] *"Flat Combining and the Synchronization-Parallelism Tradeoff"* Danny Hendler, Itai Incze, Nir Shavit, and Moran Tzafrir (SPAA '10)

cooperation: flat combining [1]

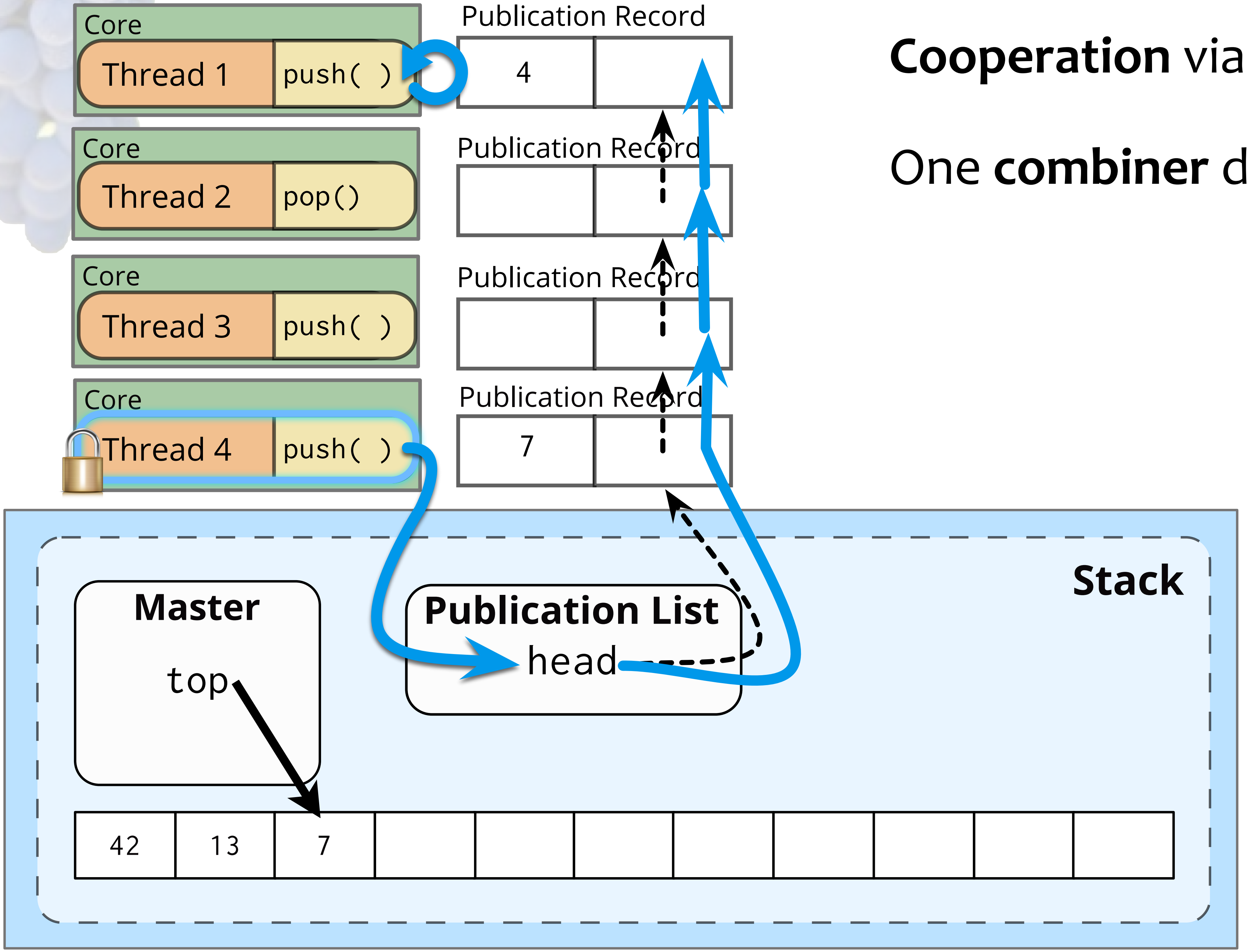


Cooperation via publication list

One **combiner** does all the work

[1] *"Flat Combining and the Synchronization-Parallelism Tradeoff"* Danny Hendler, Itai Incze, Nir Shavit, and Moran Tzafrir (SPAA '10)

cooperation: flat combining [1]

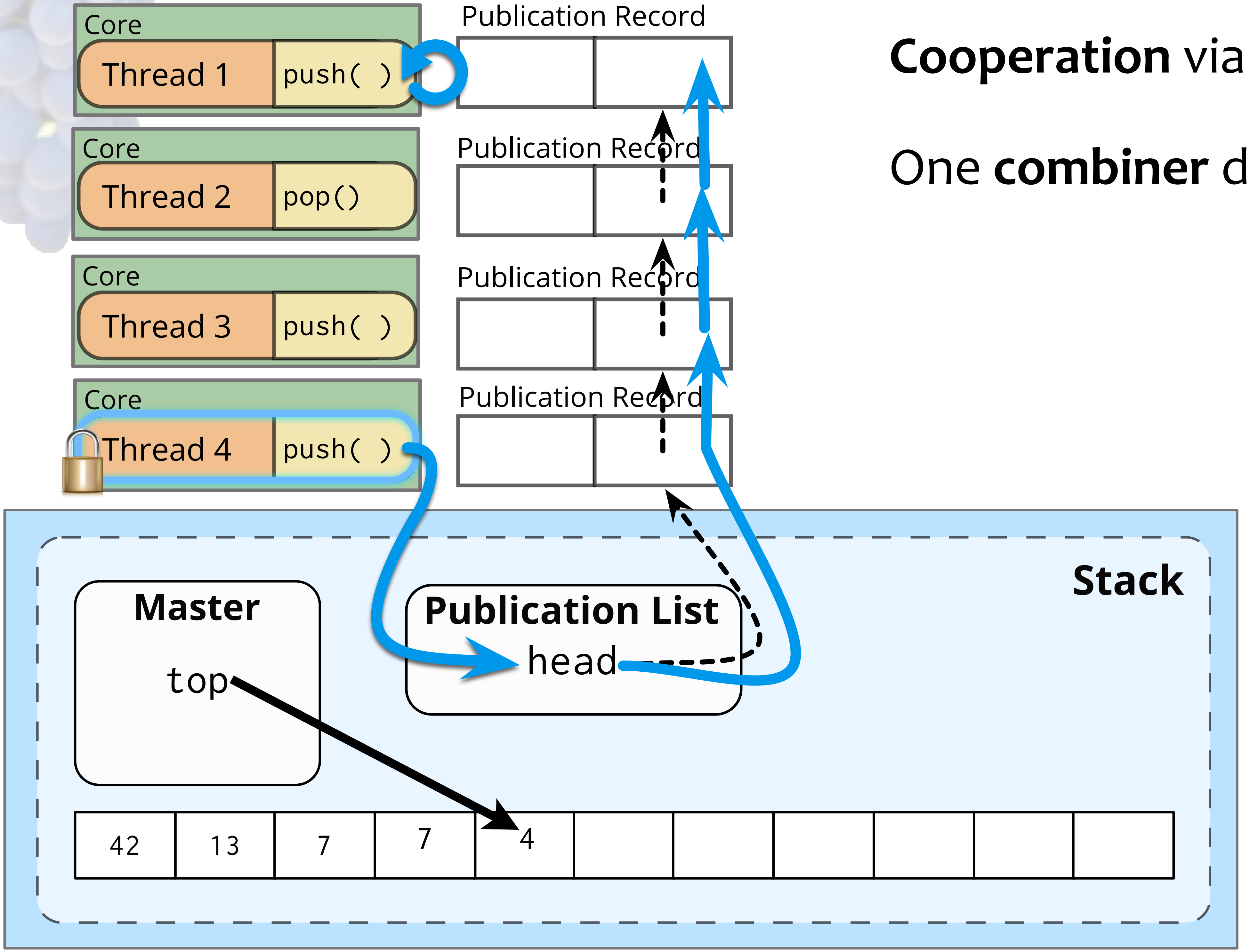


Cooperation via publication list

One **combiner** does all the work

[1] *"Flat Combining and the Synchronization-Parallelism Tradeoff"* Danny Hendler, Itai Incze, Nir Shavit, and Moran Tzafrir (SPAA '10)

cooperation: flat combining [1]

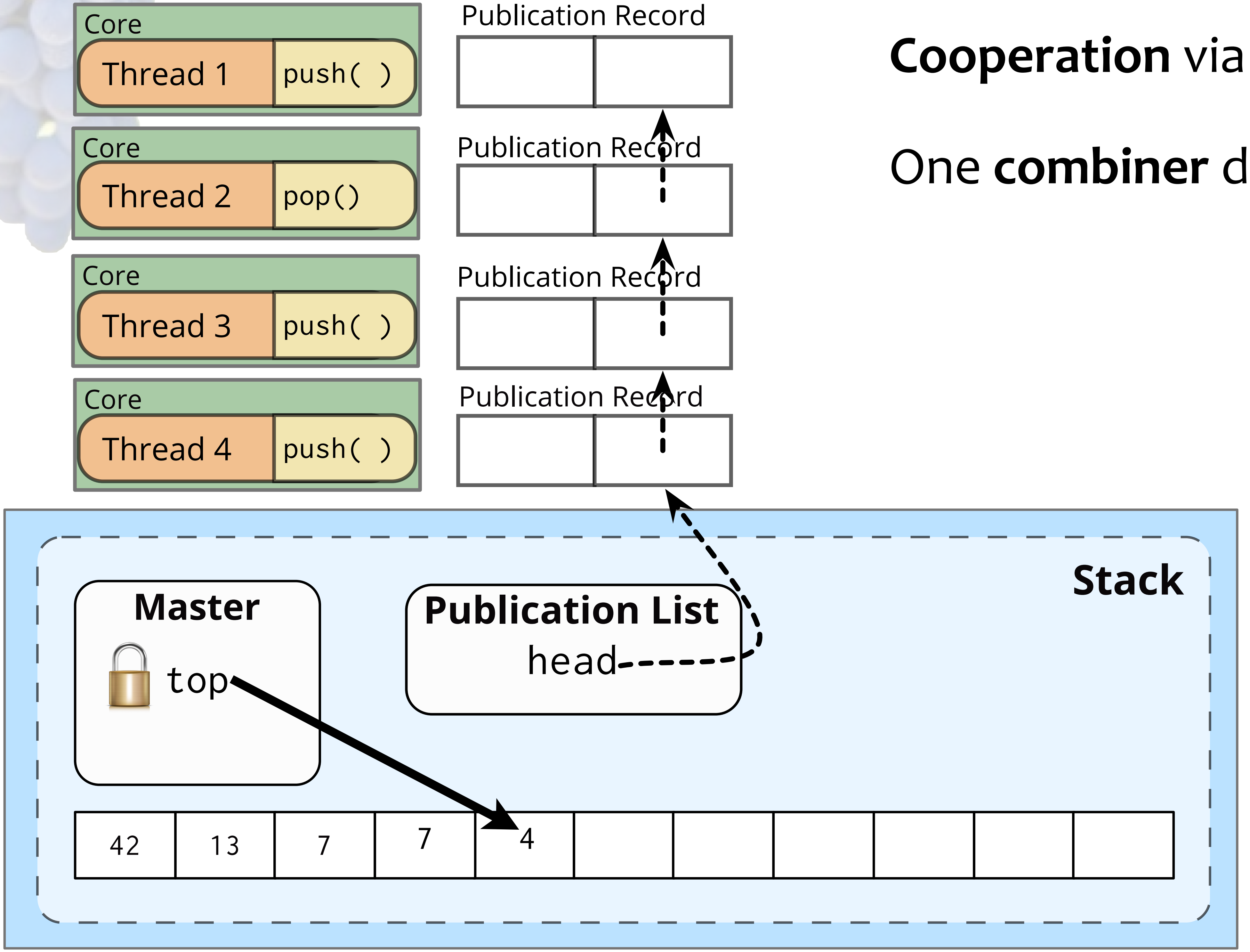


Cooperation via publication list

One **combiner** does all the work

[1] *"Flat Combining and the Synchronization-Parallelism Tradeoff"* Danny Hendler, Itai Incze, Nir Shavit, and Moran Tzafrir (SPAA '10)

cooperation: flat combining [1]



Cooperation via publication list

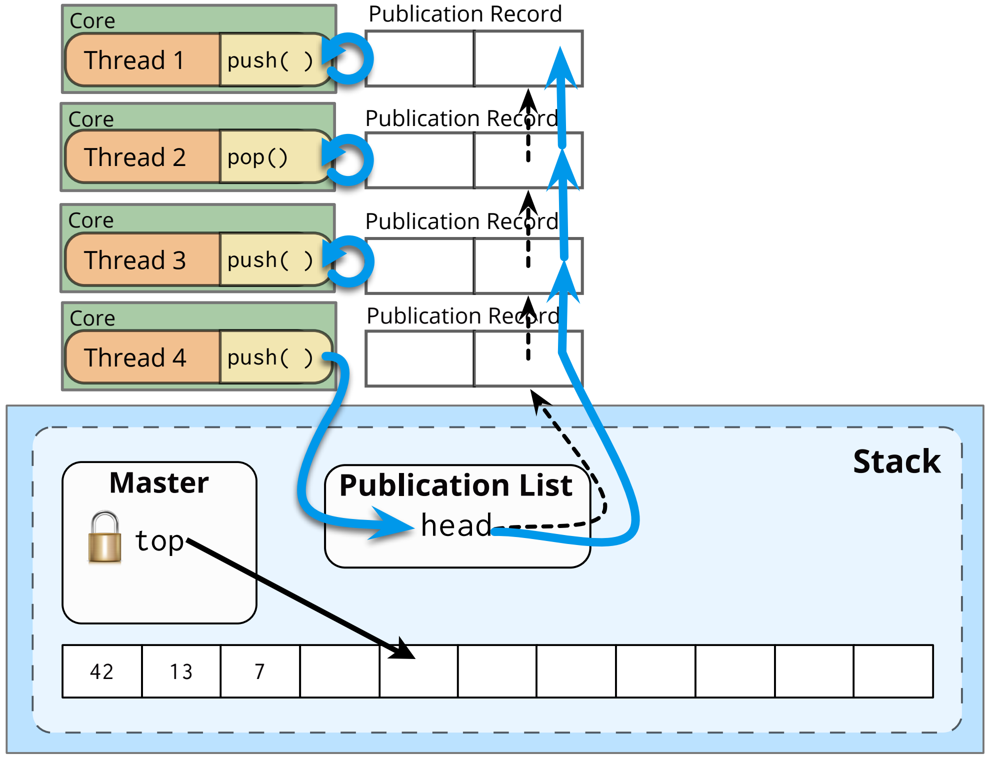
One **combiner** does all the work

[1] *"Flat Combining and the Synchronization-Parallelism Tradeoff"* Danny Hendler, Itai Incze, Nir Shavit, and Moran Tzafrir (SPAA '10)

Flat combining[1,2] in multicore

Simple locking scheme, but maximum of 1 failed CAS per thread

- beats combining ***trees***[5] and ***funnels***[3,4]
- beats fine-grained synchronization



Applicable if combined ops are faster than individually, due to:

- cache locality
- shared traversal (e.g. some linked list)
- better sequential algorithm
  (priority queue: pairing heap vs. skiplist)

[1] D. Handler, I. Incze, N. Shavit, M. Tzafrir. "*Flat Combining and the Synchronization-Parallelism Tradeoff*" (SPAA 2010)

[2] D. Hendler, I. Incze, N. Shavit, M. Tzafrir. "*Scalable Flat-Combining Based Synchronous Queues*" (DISC 2010)

[3] S. Kahan and P. Konecny. "*MAMA!*" (2006)

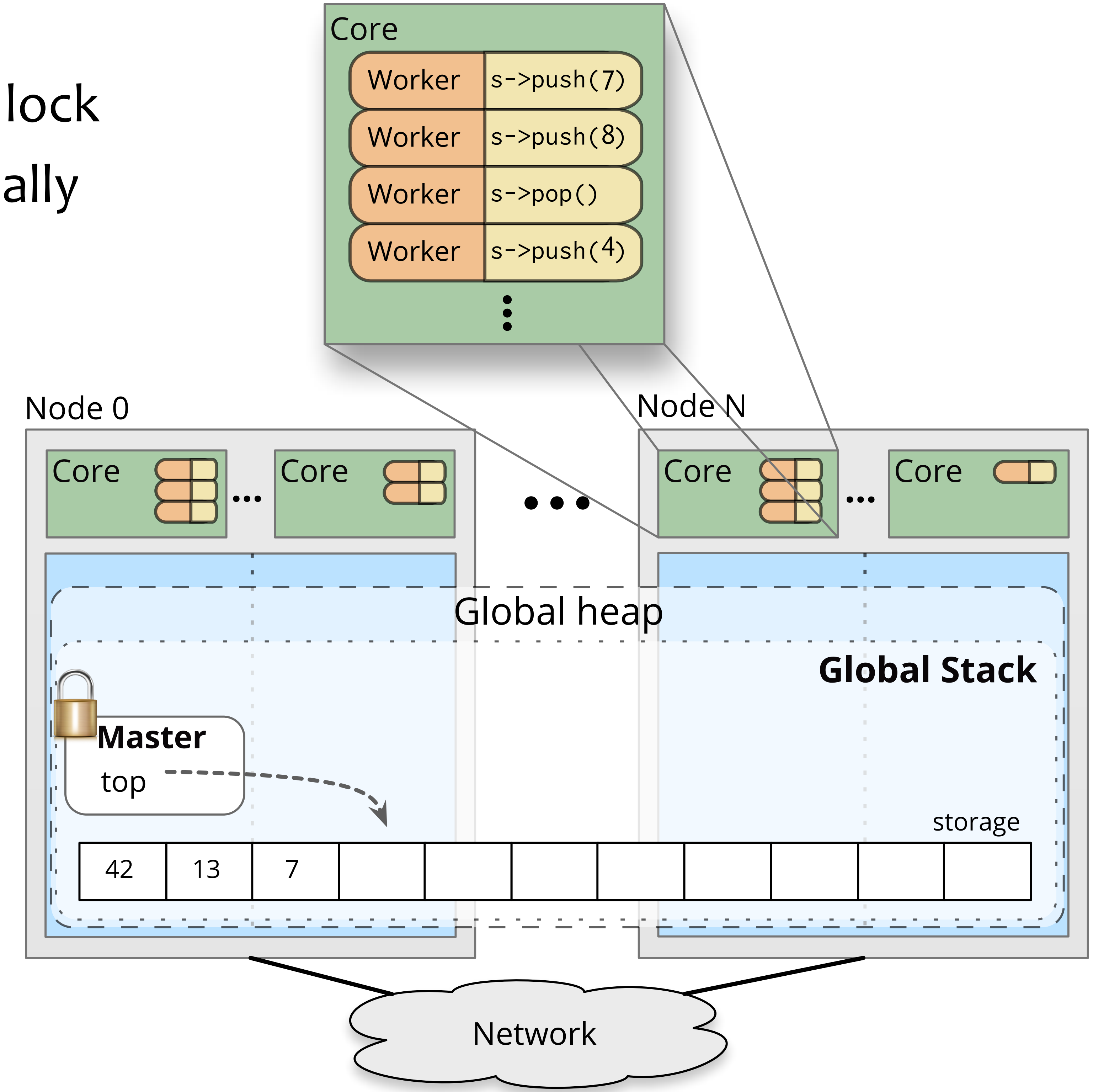
[4] N. Shavit and A. Zemach. "*Combining funnels*" (2000)

[5] P.-C. Yew, N.-F. Tzeng, and D. H. Lawrie. "*Combining trees*" (1987)

Flat combining in PGAS

Distributed synchronization

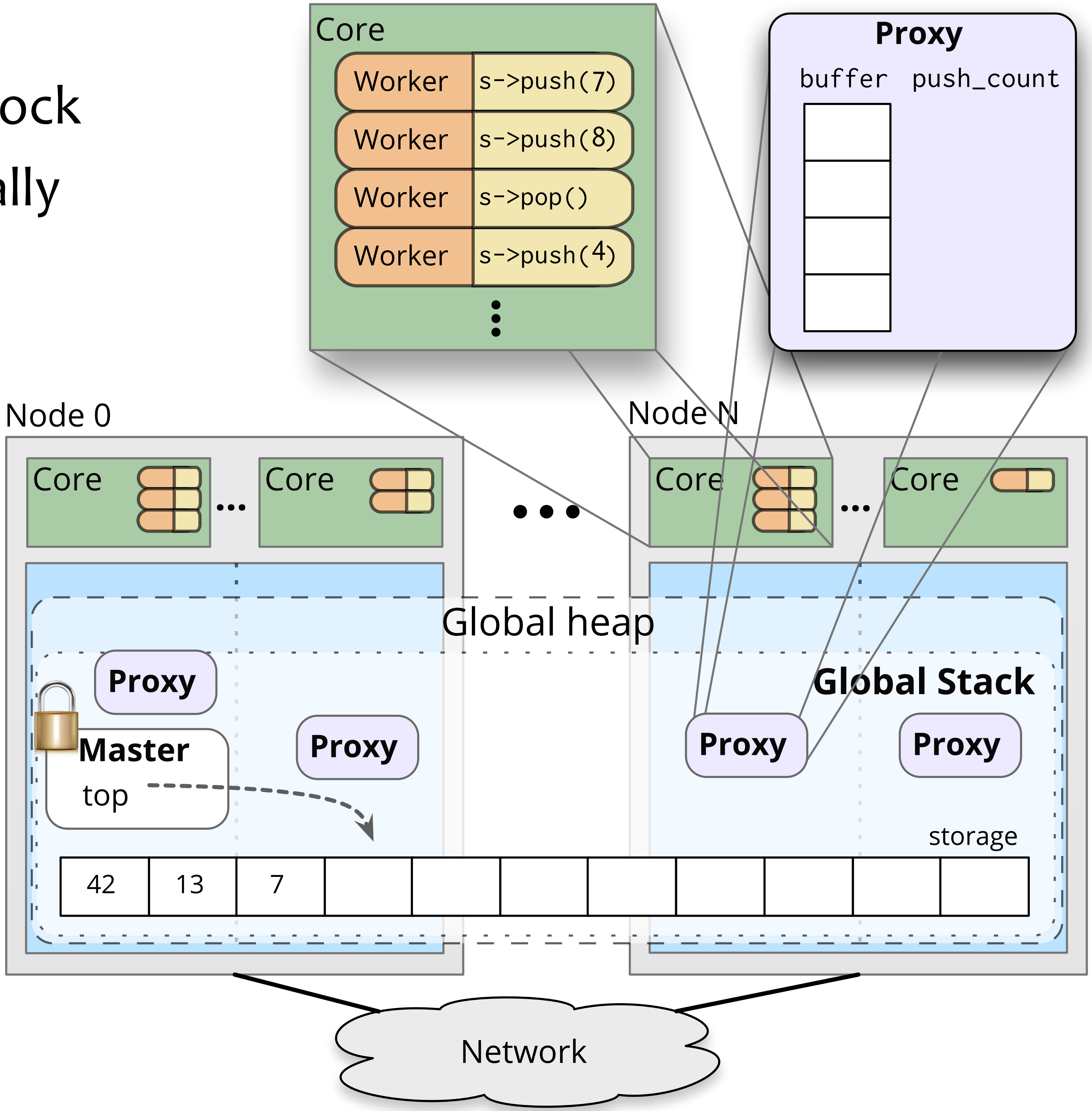
- reduce serialization on global lock
- avoid making operations globally visible if possible



Flat combining in PGAS

Distributed synchronization
- reduce serialization on global lock
- avoid making operations globally visible if possible

Combining structure: local **proxy**
- calls operate on this instead
- resolve locally if possible



Flat combining in PGAS

Distributed synchronization

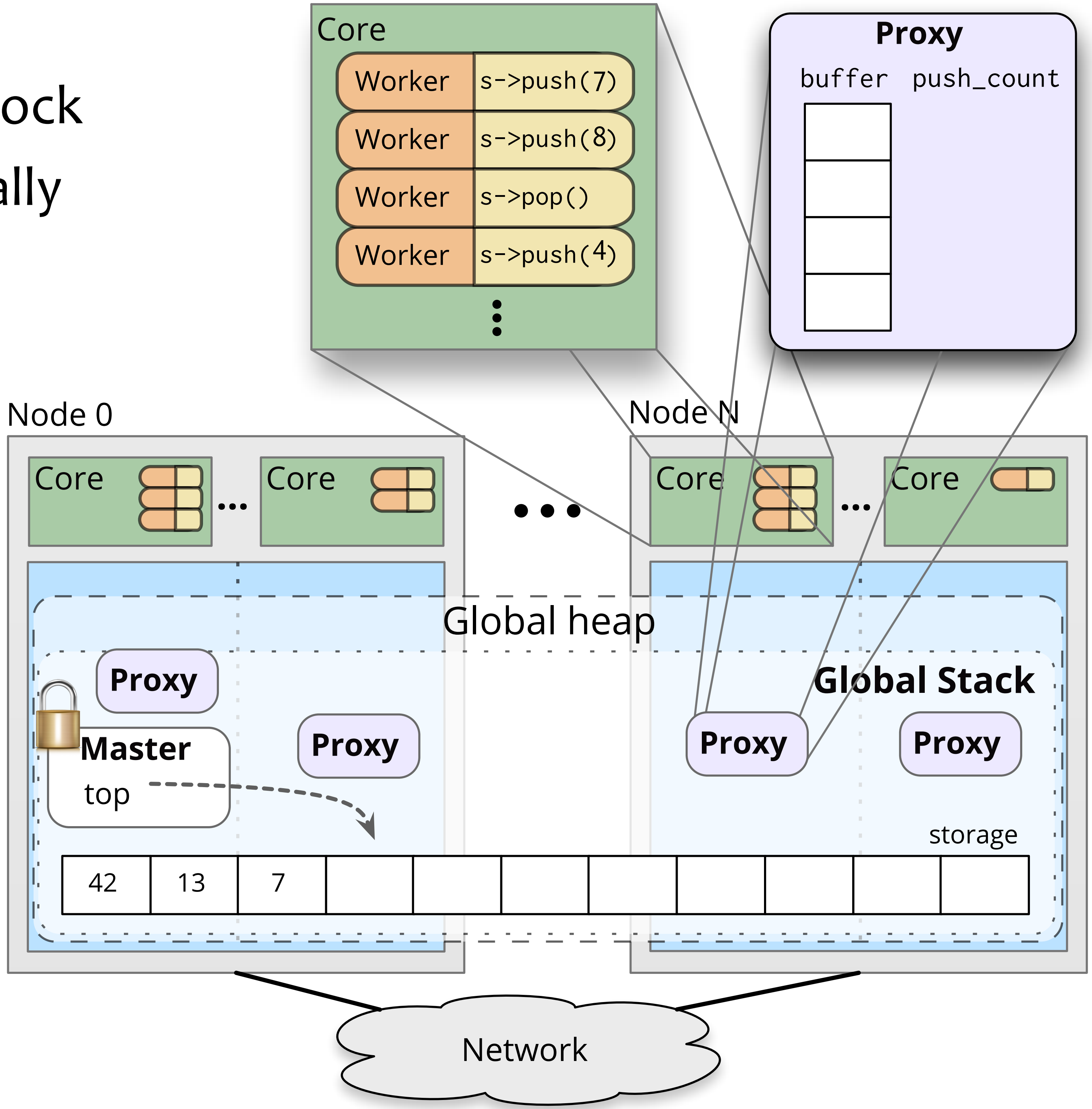
- reduce serialization on global lock
- avoid making operations globally visible if possible

Combining structure: local **proxy**

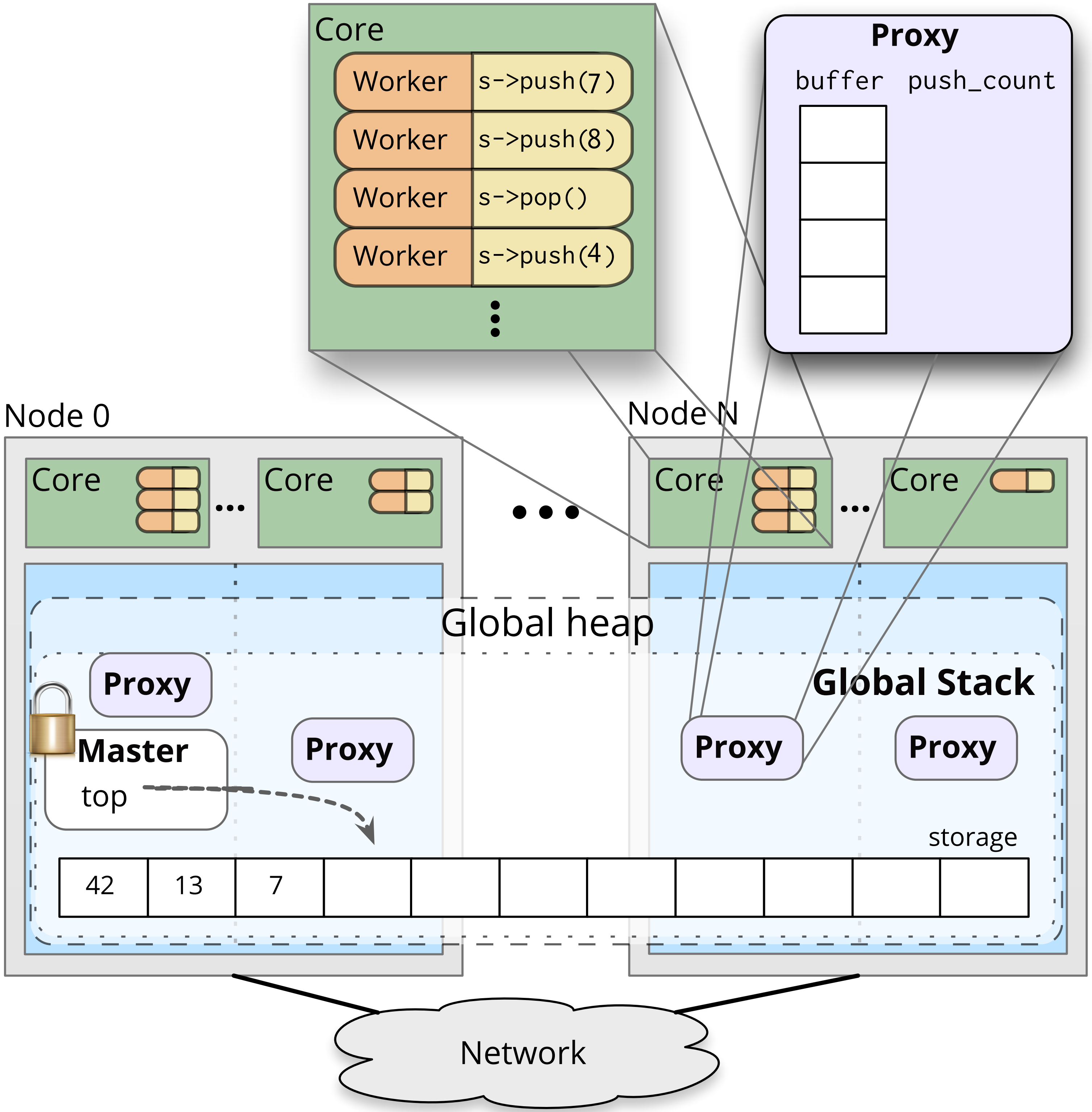
- calls operate on this instead
- resolve locally if possible

One worker commits combined op

- progress guarantee: always one in flight per core



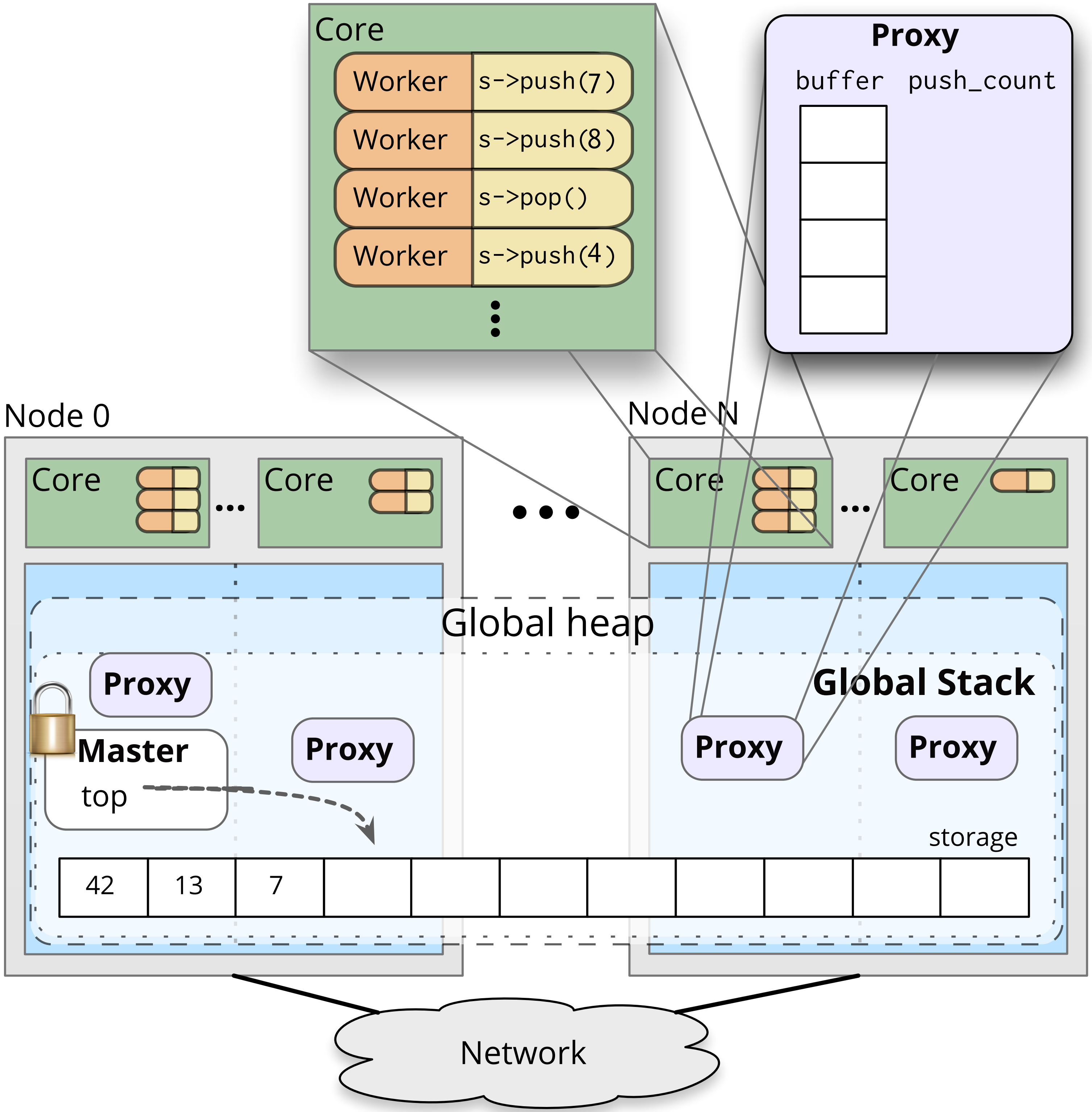
Flat combining in PGAS



Flat combining in PGAS

Workers operate on local proxy

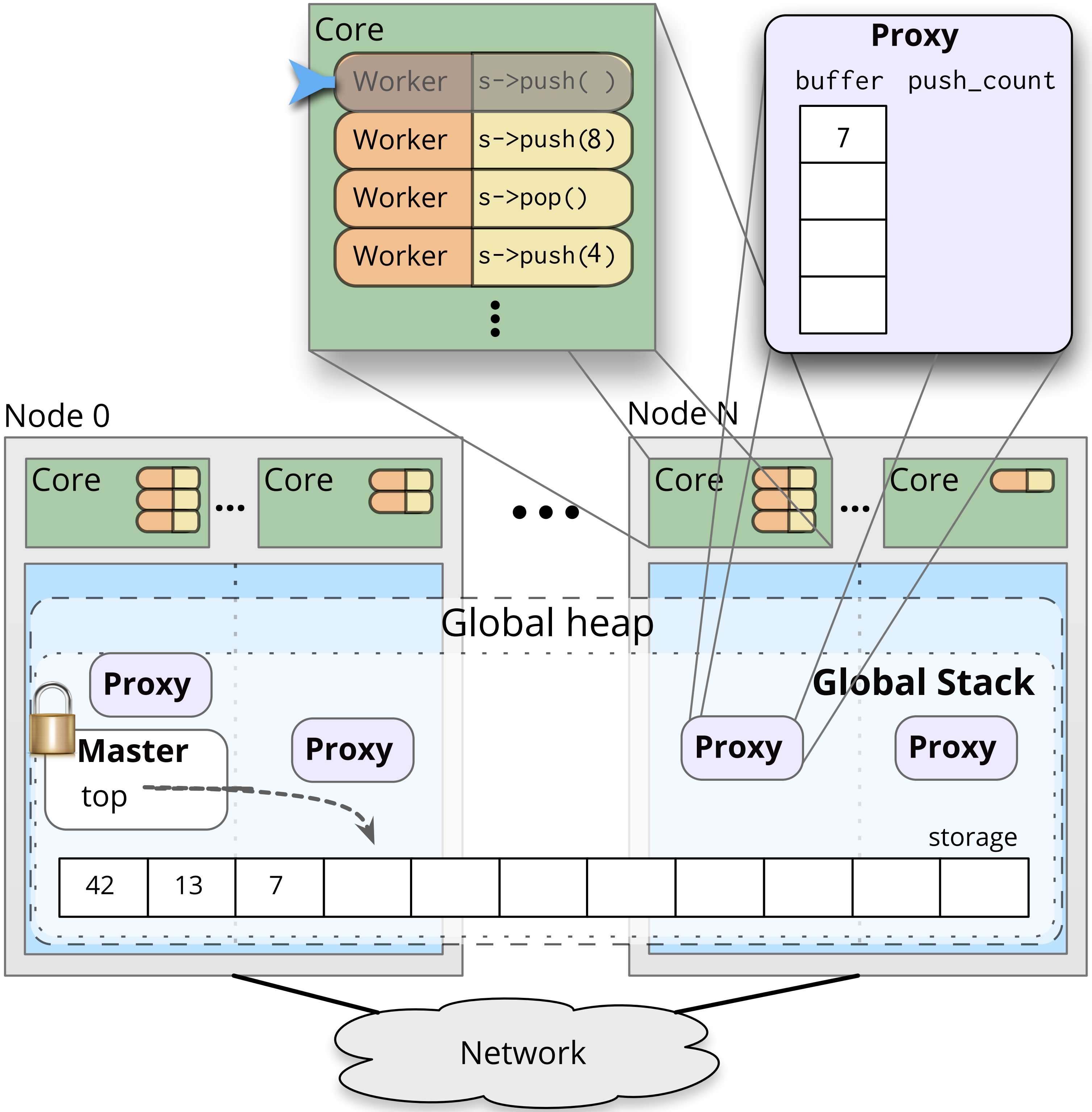
- resolve locally where possible



Flat combining in PGAS

Workers operate on local proxy

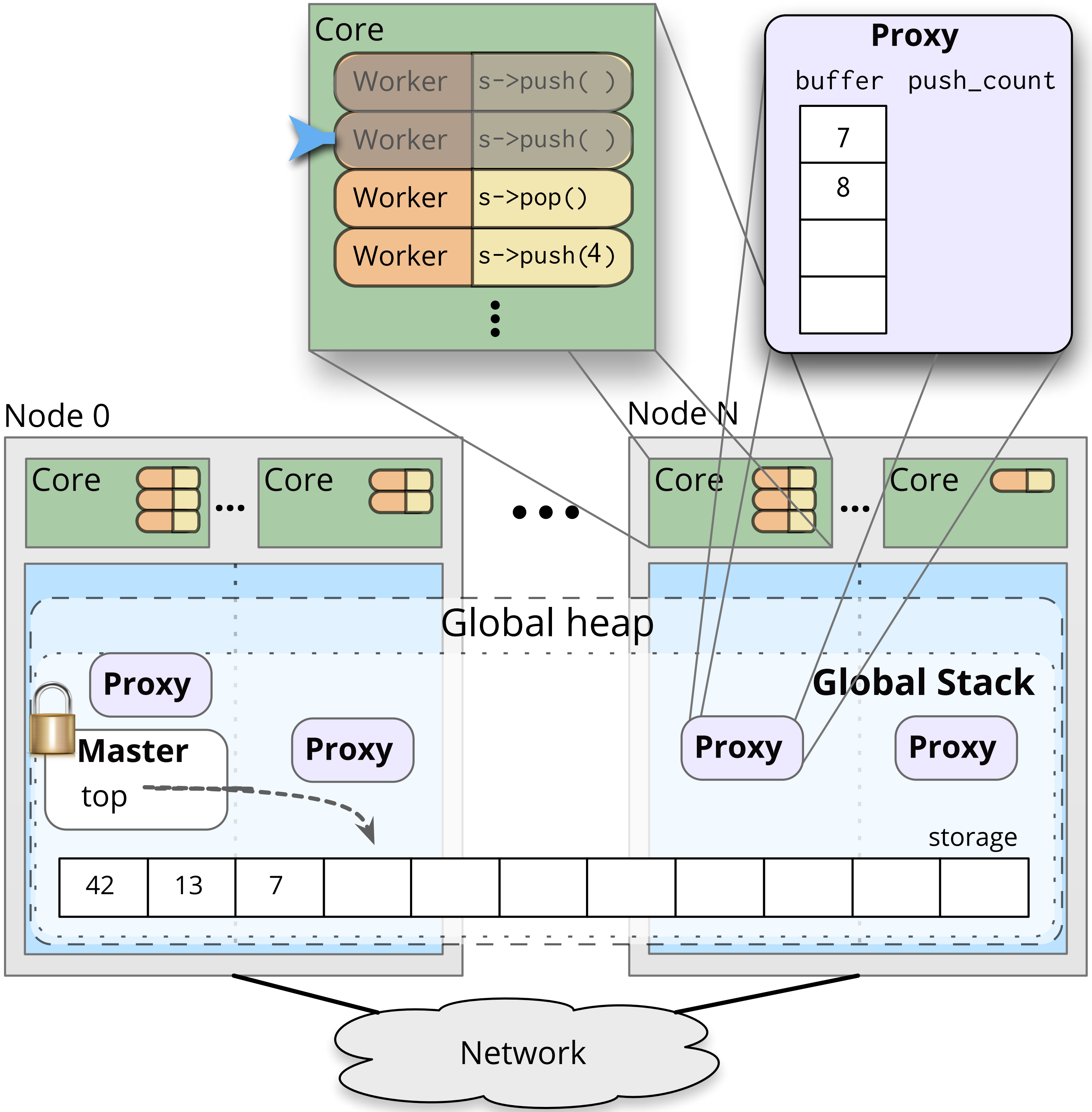
- resolve locally where possible



Flat combining in PGAS

Workers operate on local proxy

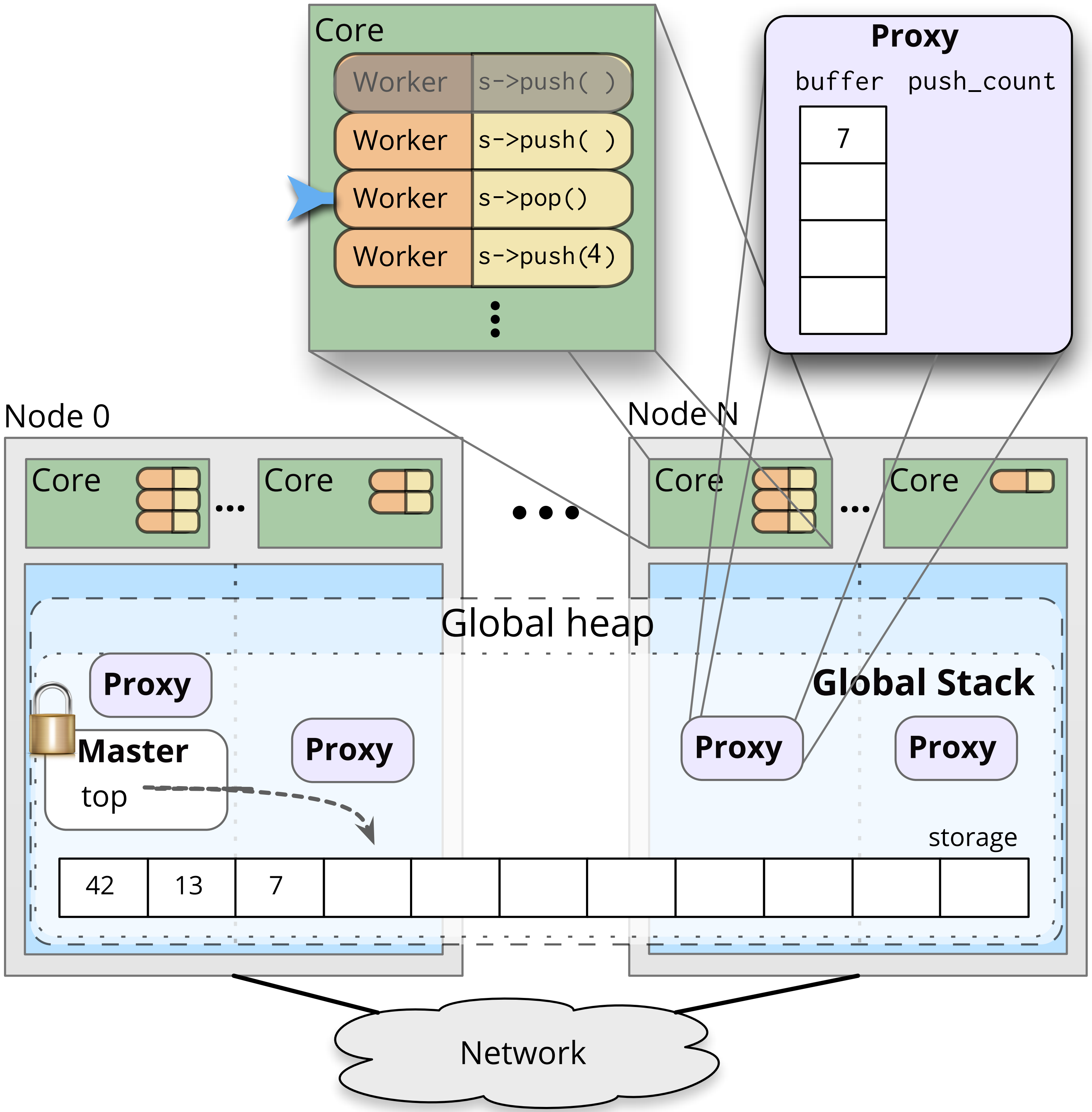
- resolve locally where possible



Flat combining in PGAS

Workers operate on local proxy

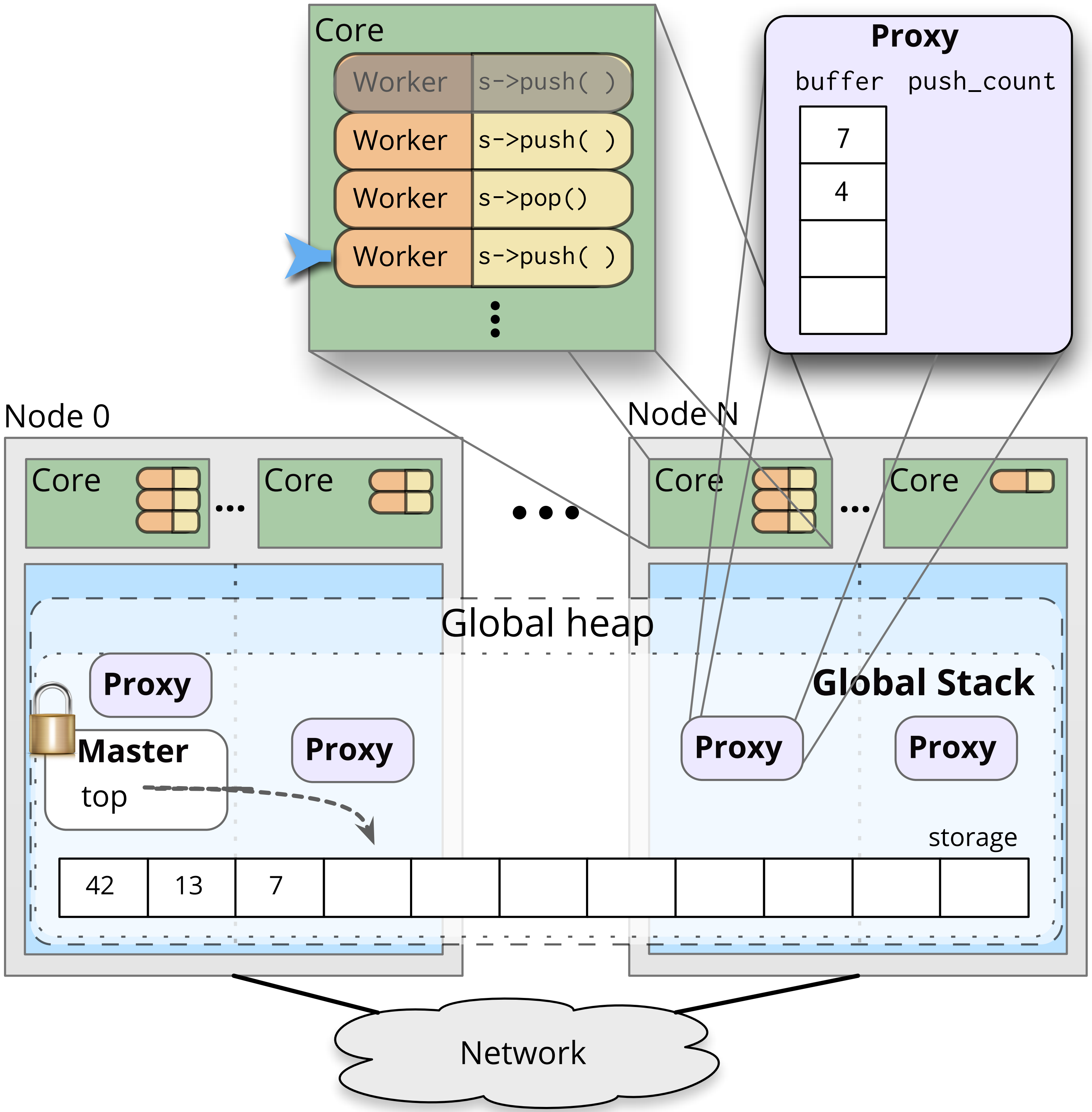
- resolve locally where possible



Flat combining in PGAS

Workers operate on local proxy

- resolve locally where possible



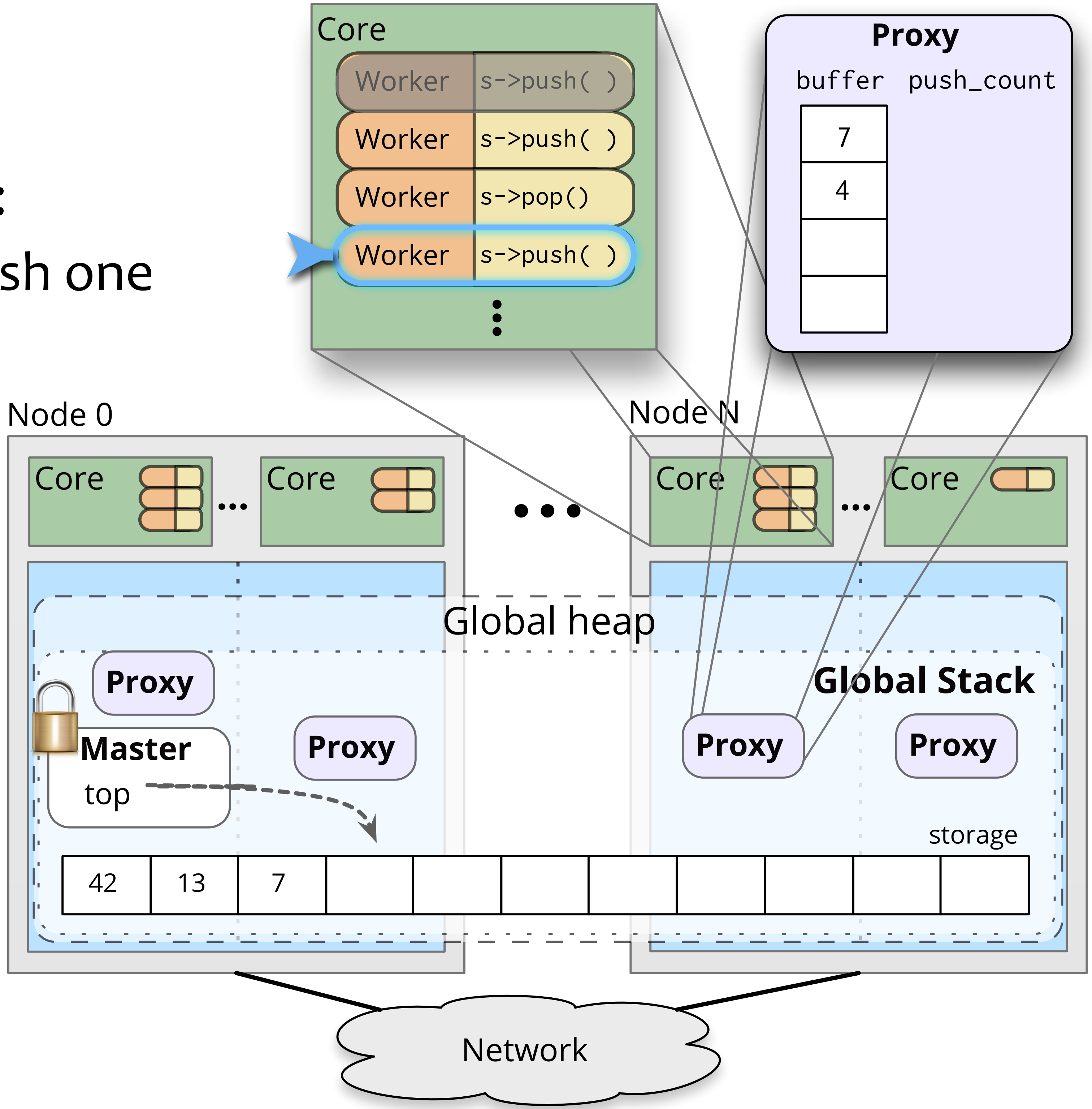
Flat combining in PGAS

Workers operate on local proxy

- resolve locally where possible

One worker becomes **combiner:**

- freeze current Proxy, create fresh one for next round
- globally commit
- wake blocked workers when finished
- trigger next Proxy to go



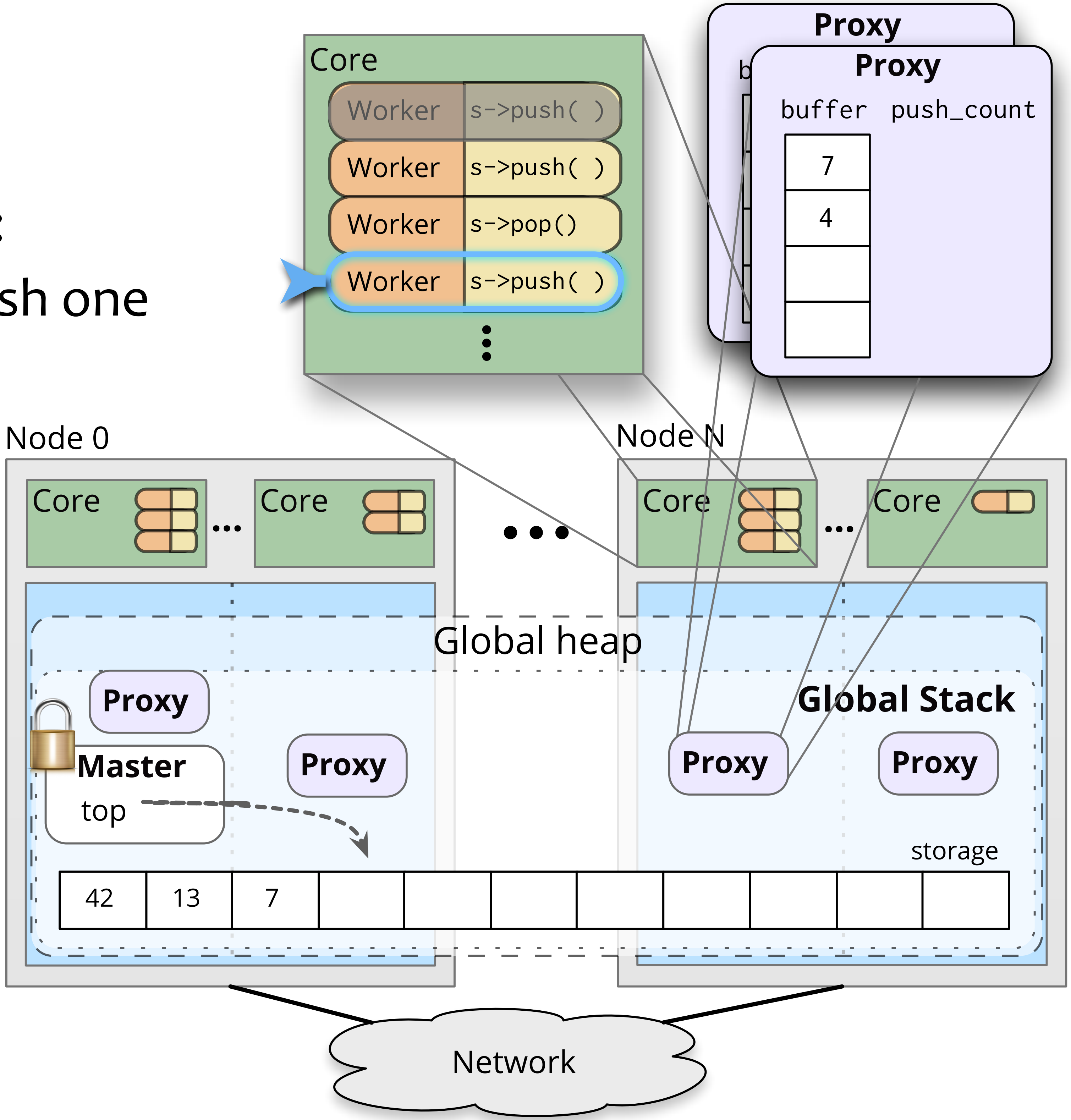
Flat combining in PGAS

Workers operate on local proxy

- resolve locally where possible

One worker becomes **combiner:**

- freeze current Proxy, create fresh one for next round
- globally commit
- wake blocked workers when finished
- trigger next Proxy to go



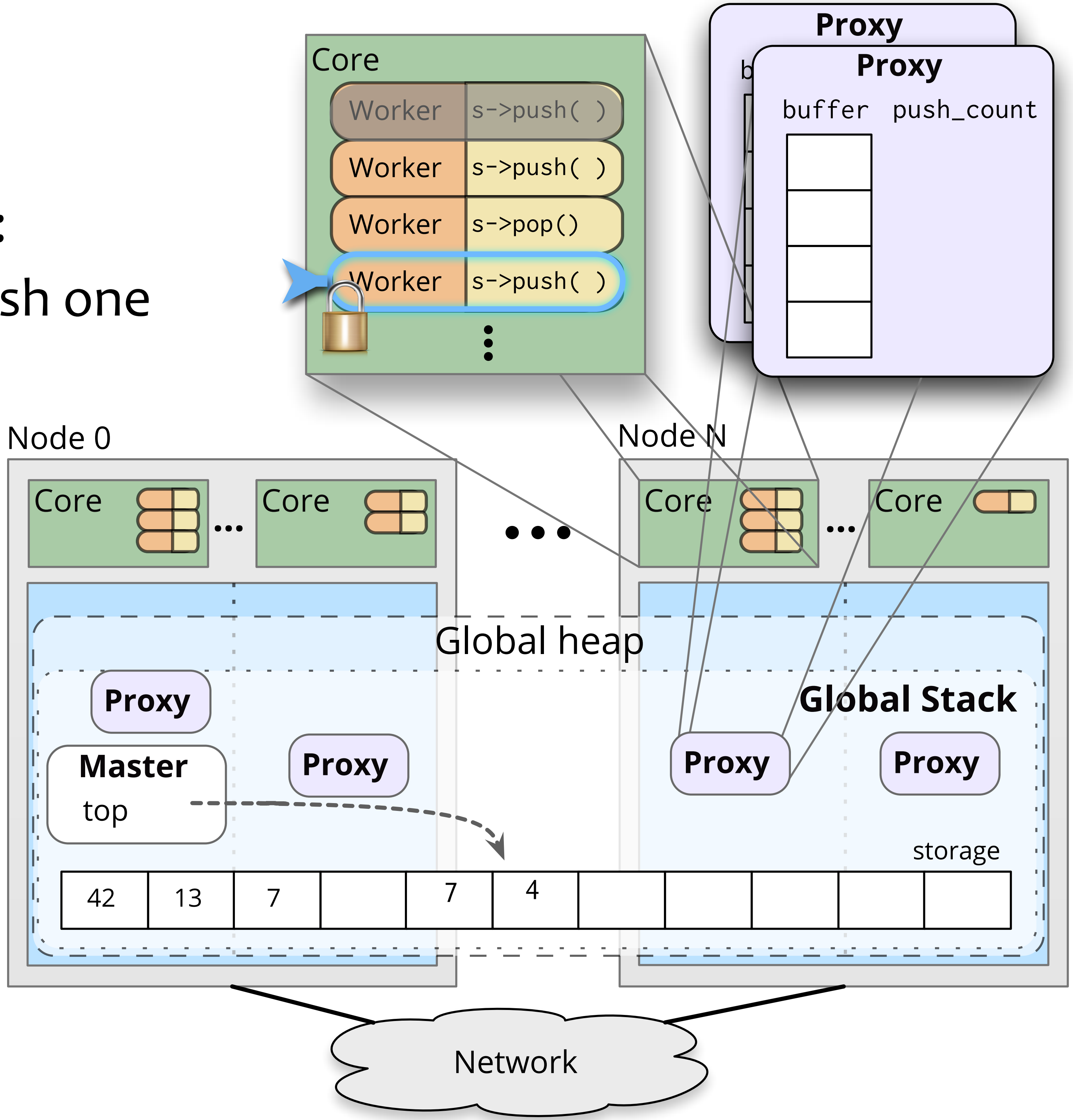
Flat combining in PGAS

Workers operate on local proxy

- resolve locally where possible

One worker becomes **combiner:**

- freeze current Proxy, create fresh one for next round
- globally commit
- wake blocked workers when finished
- trigger next Proxy to go



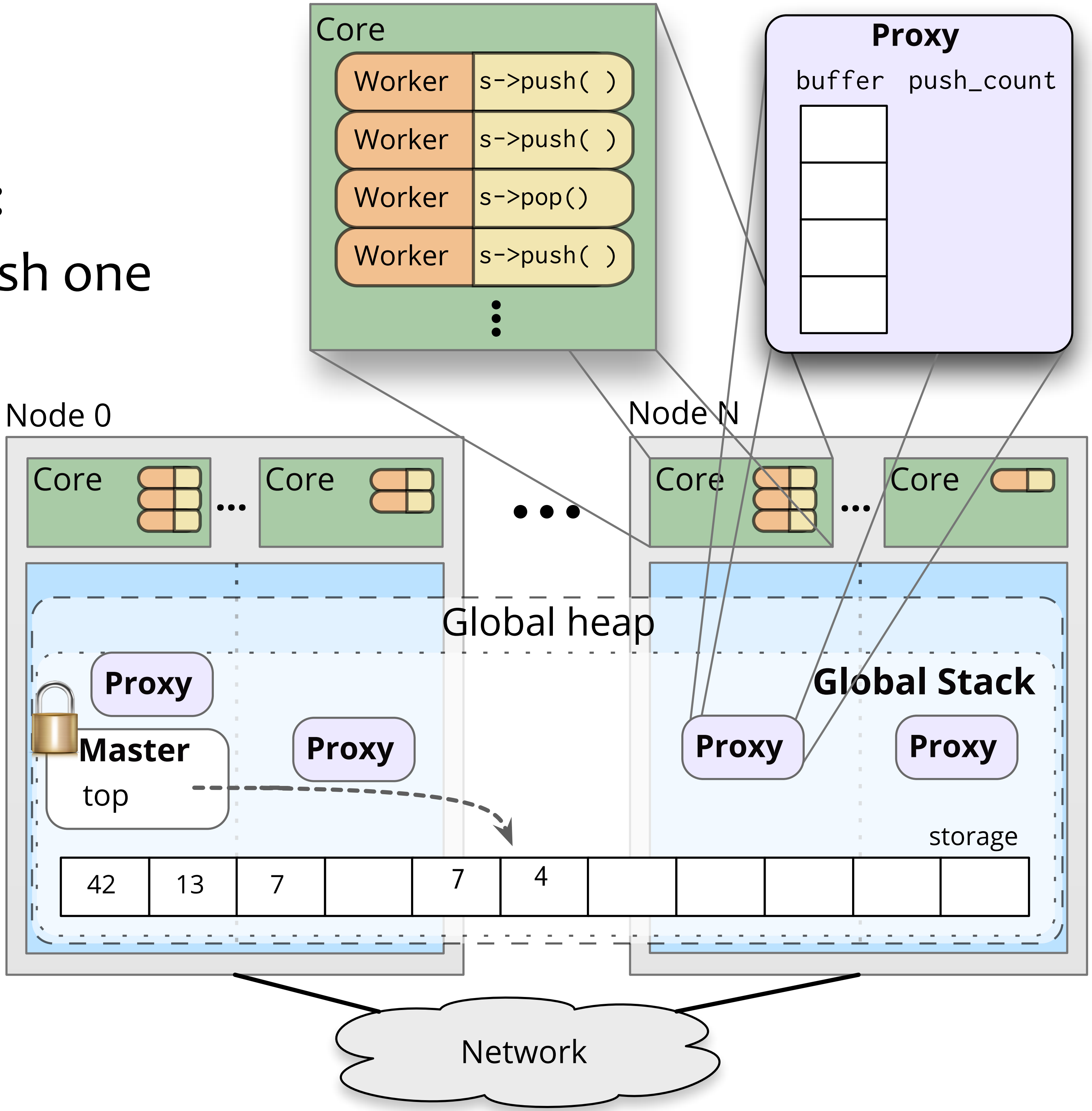
Flat combining in PGAS

Workers operate on local proxy

- resolve locally where possible

One worker becomes **combiner:**

- freeze current Proxy, create fresh one for next round
- globally commit
- wake blocked workers when finished
- trigger next Proxy to go



Flat combining in PGAS

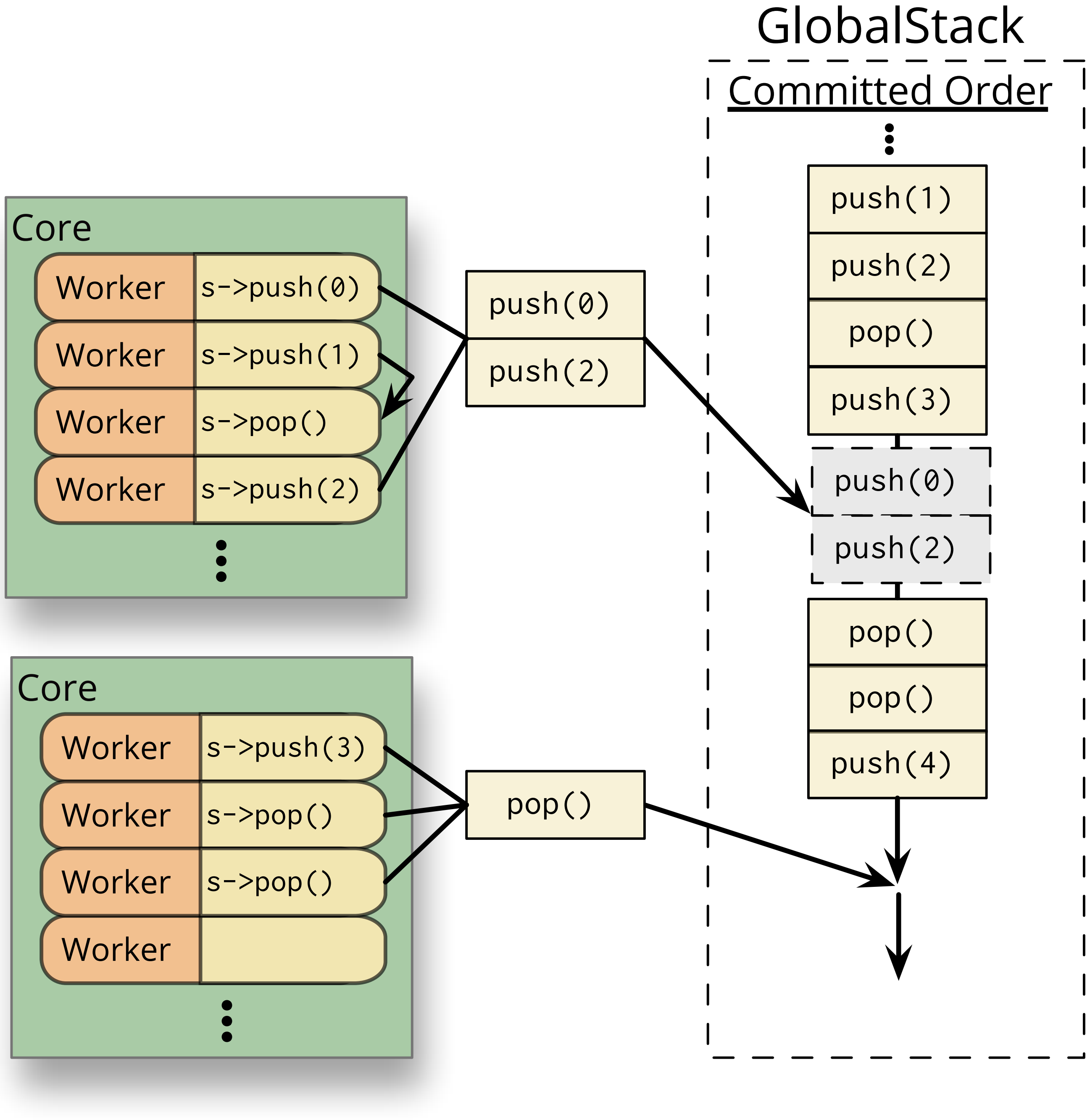
Sequential Consistency

C++ model: SC for Data-Race-Free

Enforcing **linearizability:**

– ensure program order by blocking thread until globally committed

– globally- and locally-observable order must coincide



Flat combining in PGAS

Sequential Consistency

C++ model: SC for Data-Race-Free

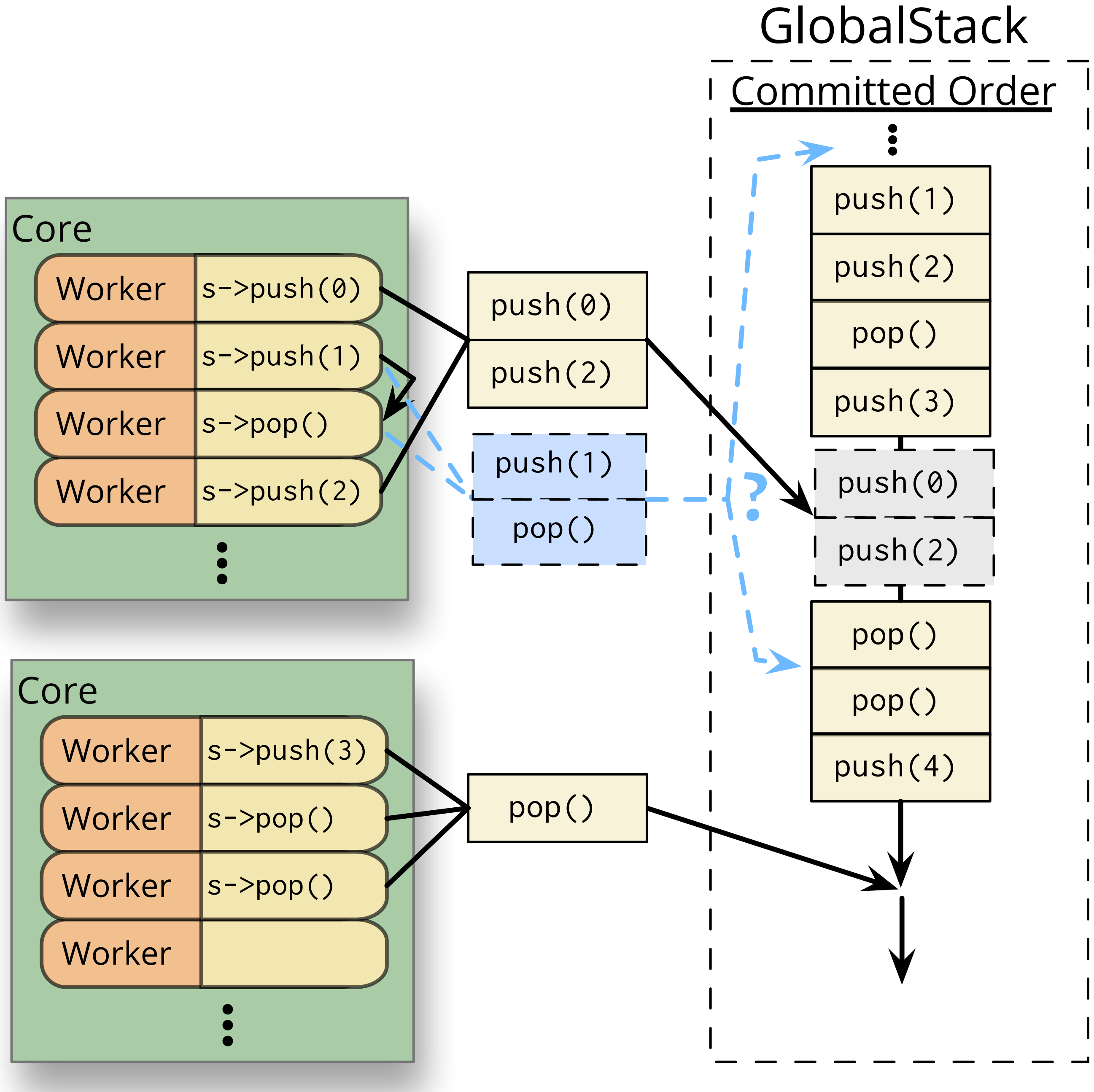
Enforcing **linearizability:**

– ensure program order by blocking thread until globally committed

– globally- and locally-observable order must coincide

GlobalStack

push/pop *annihilate* each other, can be anywhere in global order



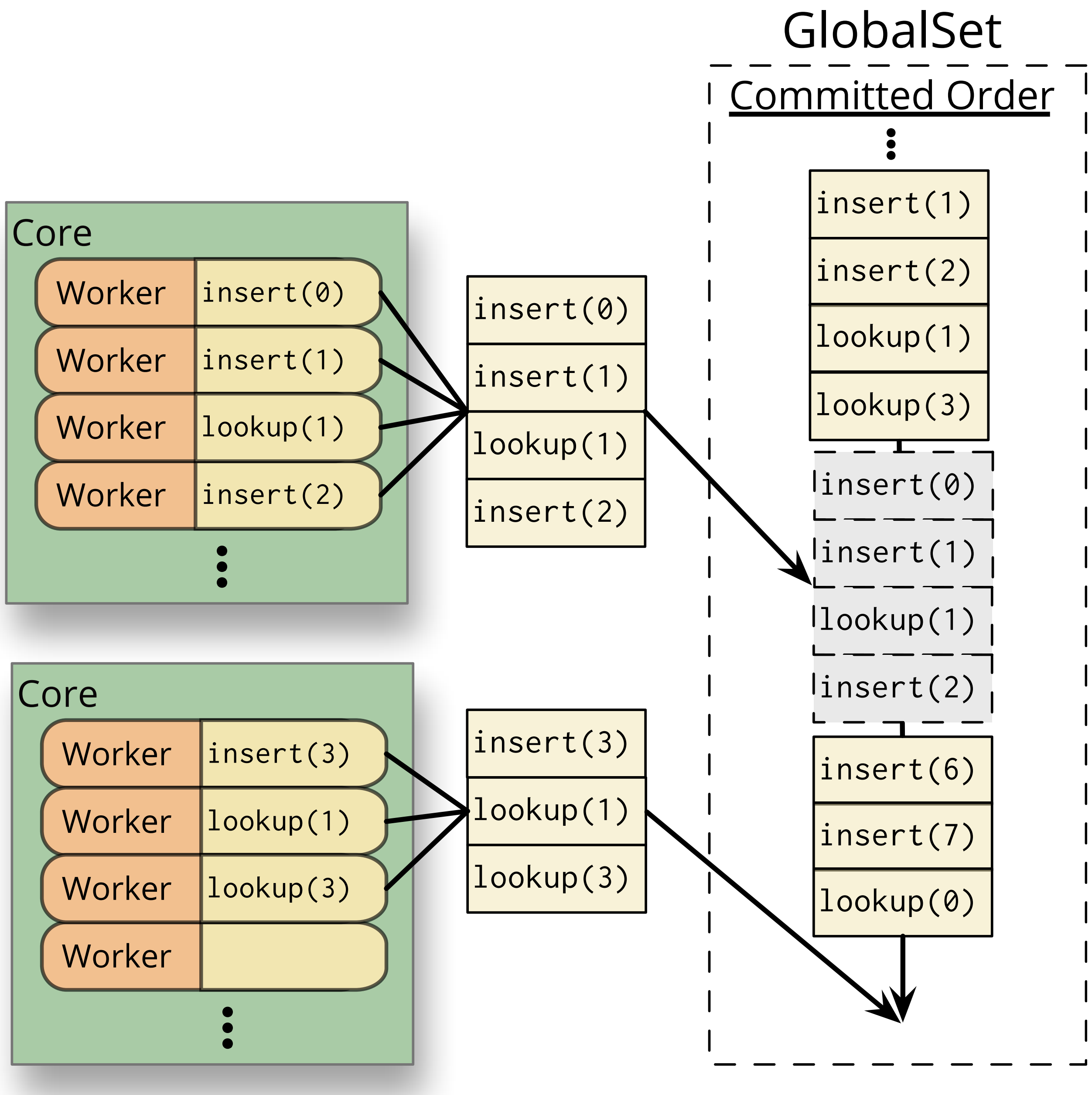
Flat combining in PGAS

Sequential Consistency

C++ model: SC for Data-Race-Free

Enforcing **linearizability:**

- ensure program order by blocking thread until globally committed
- globally- and locally-observable order must coincide



Flat combining in PGAS

Sequential Consistency

C++ model: SC for Data-Race-Free

Enforcing **linearizability:**

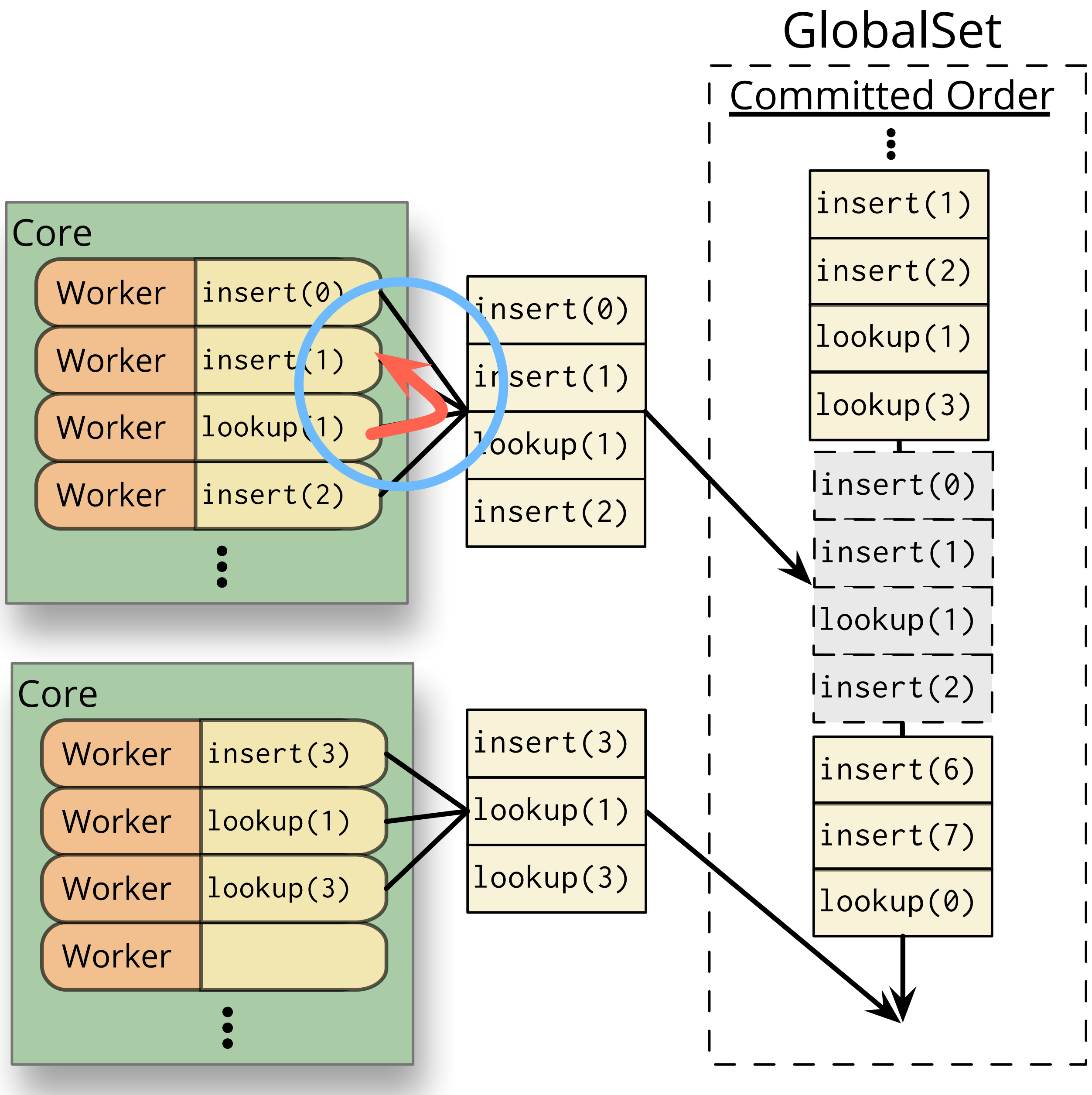
– ensure program order by blocking thread until globally committed

– globally- and locally-observable order must coincide

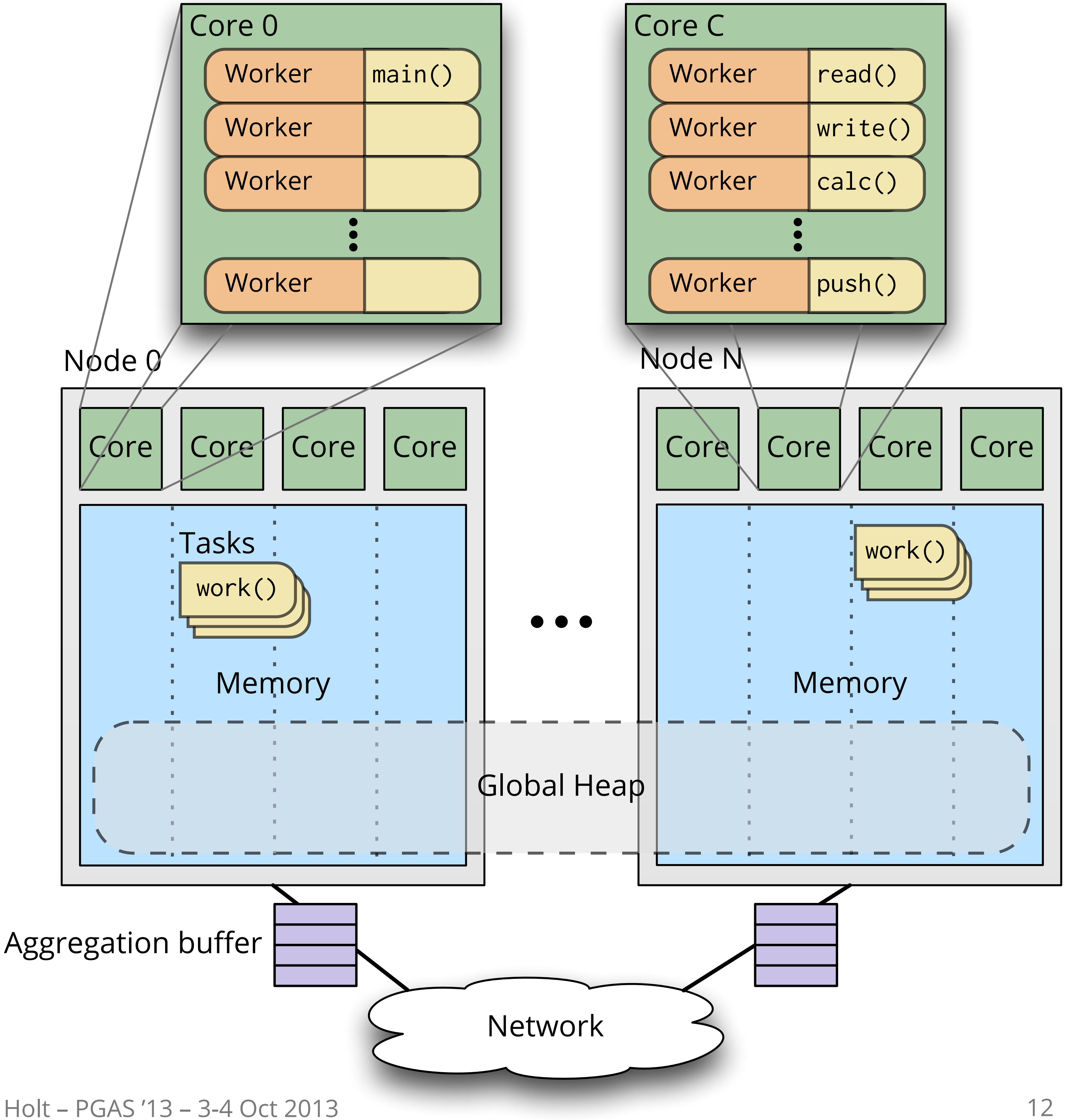
GlobalSet/GlobalMap

– insert/lookup must preserve order

– cheaper to disallow local lookups

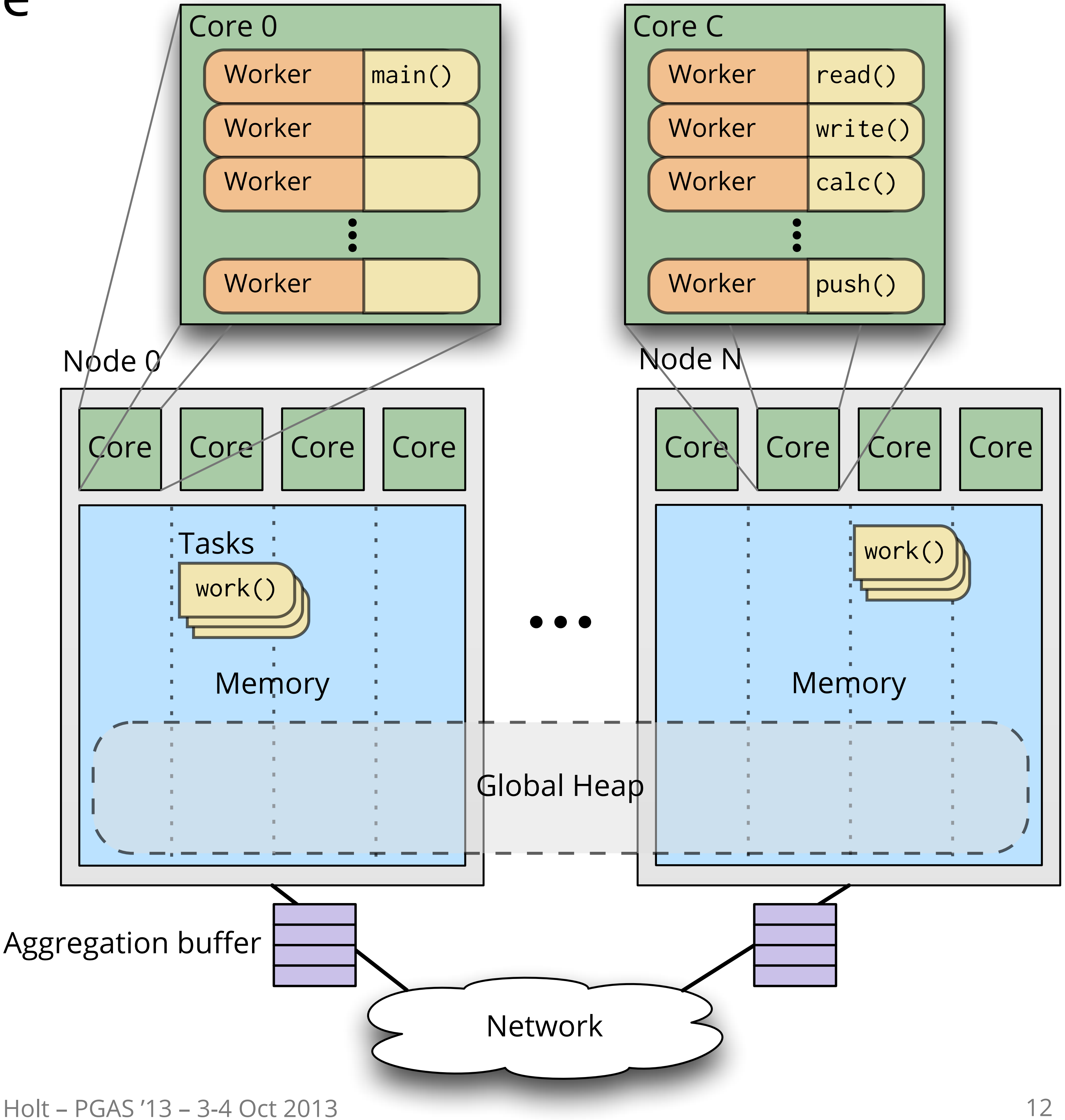


Grappa: a latency-tolerant PGAS runtime



Grappa: a latency-tolerant PGAS runtime

Global-view, single system image



Grappa: a latency-tolerant PGAS runtime

Global-view, single system image

C++11 library interface

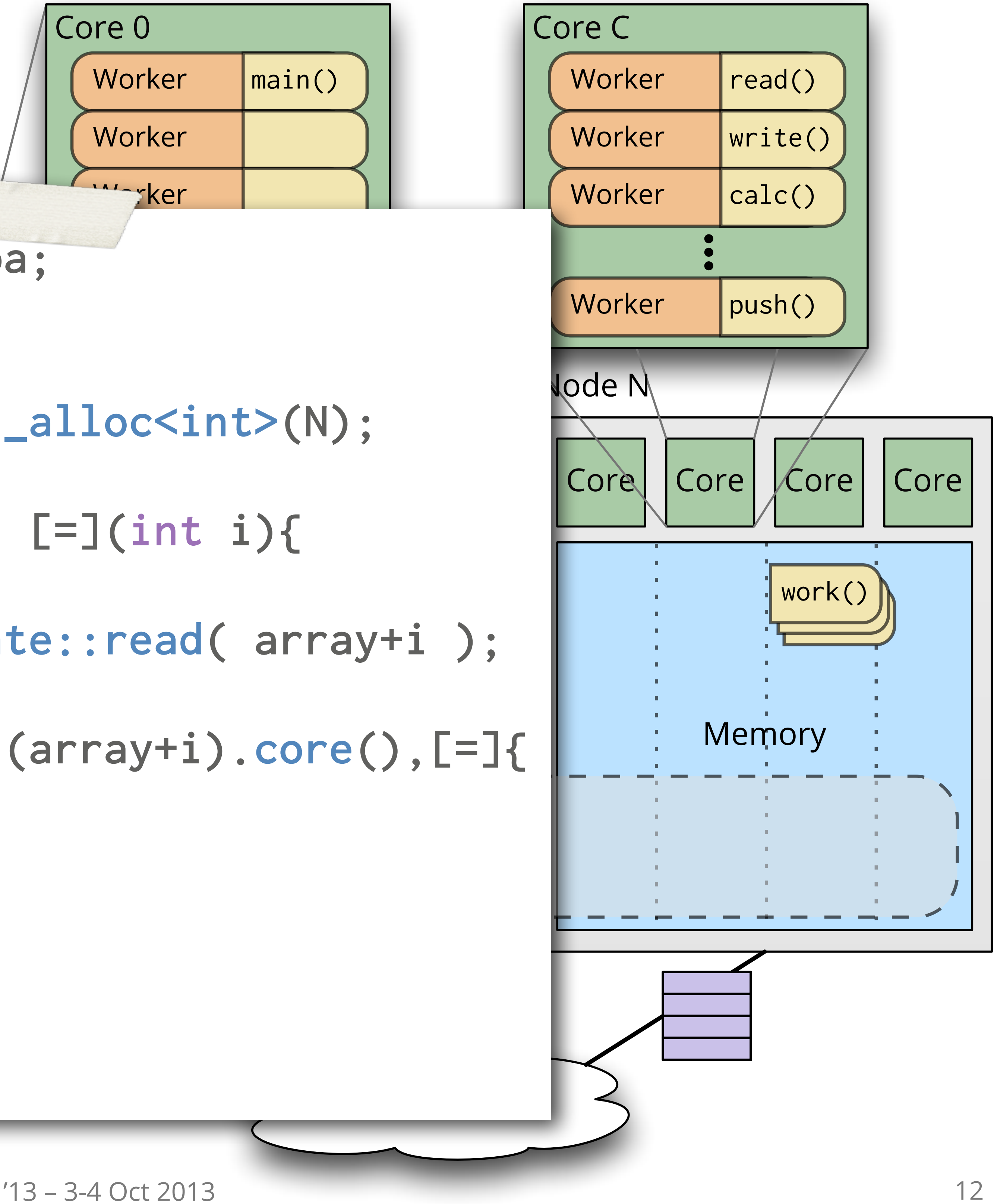
```
using namespace Grappa;

void grappa_main() {
  auto array = global_alloc<int>(N);

  forall_global(0, N, [=](int i){

    auto val = delegate::read( array+i );
    if (val == 0) {
      delegate::call((array+i).core(),[=]{
        // ...
      });
    }

  });
}
```

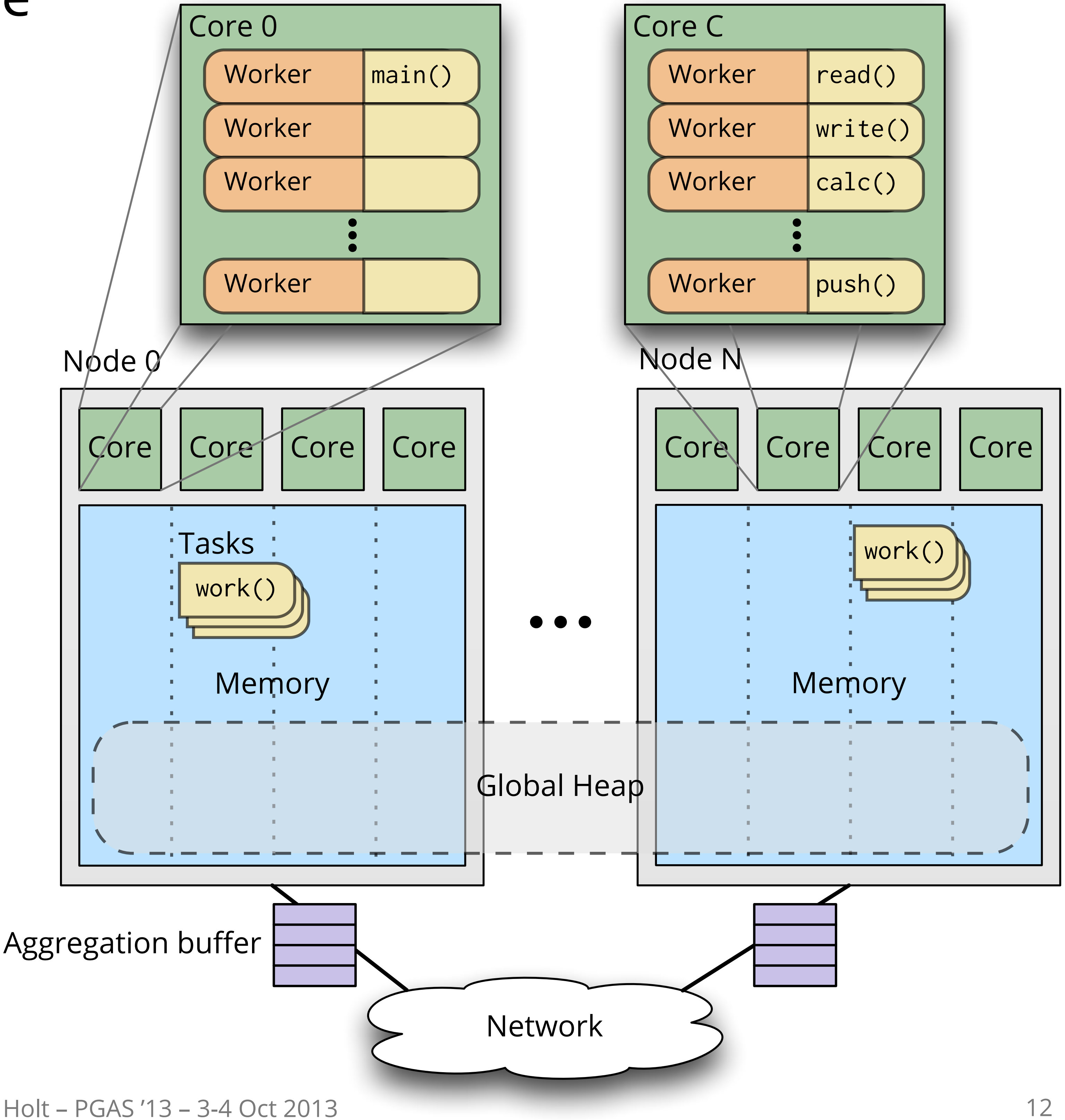


Grappa: a latency-tolerant PGAS runtime

Global-view, single system image

C++11 library interface

Aggregated **communication**



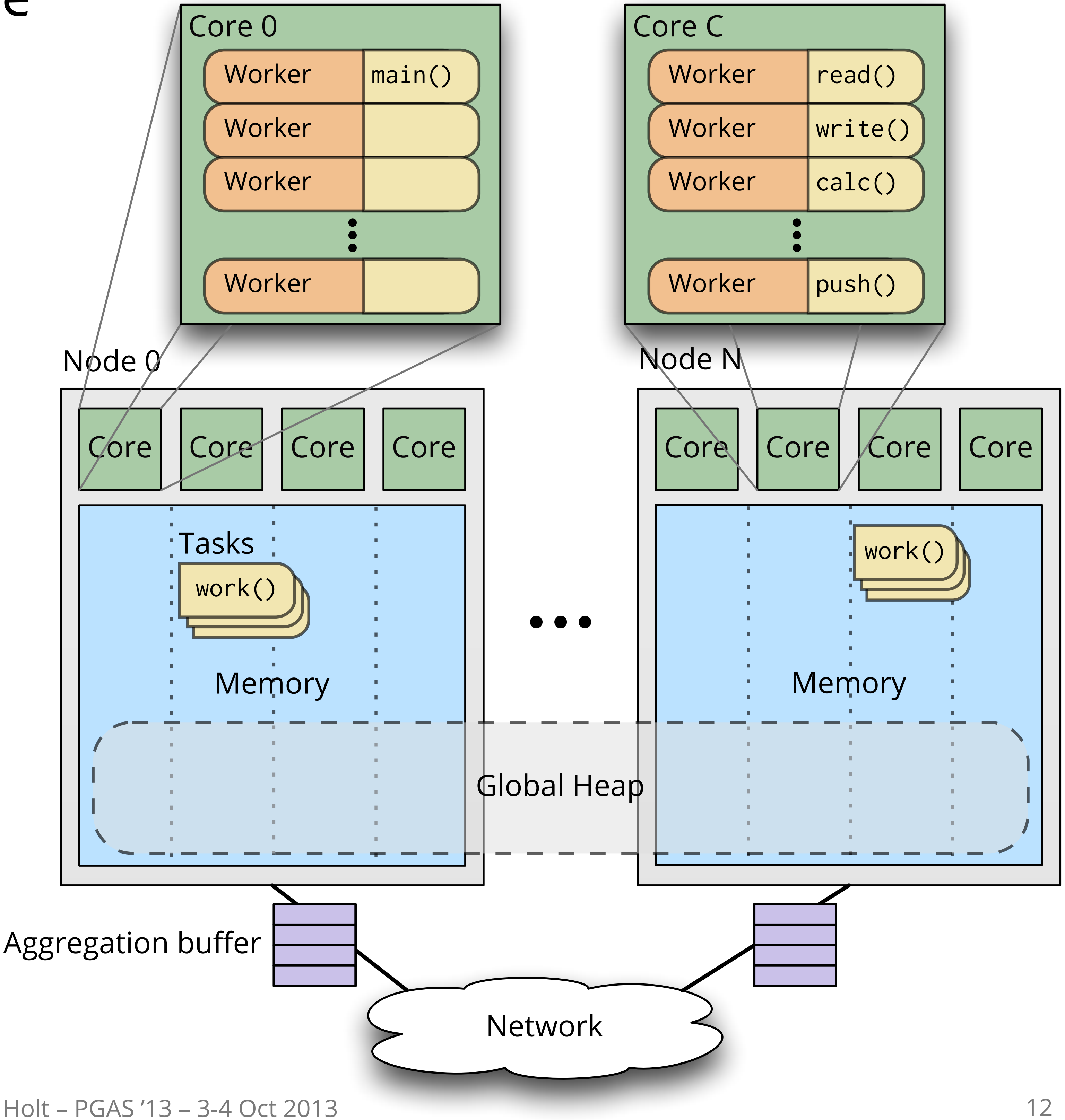
Grappa: a latency-tolerant PGAS runtime

Global-view, single system image

C++11 library interface

Aggregated communication

Lightweight user-level threads for ***latency tolerance***



Grappa: a latency-tolerant PGAS runtime

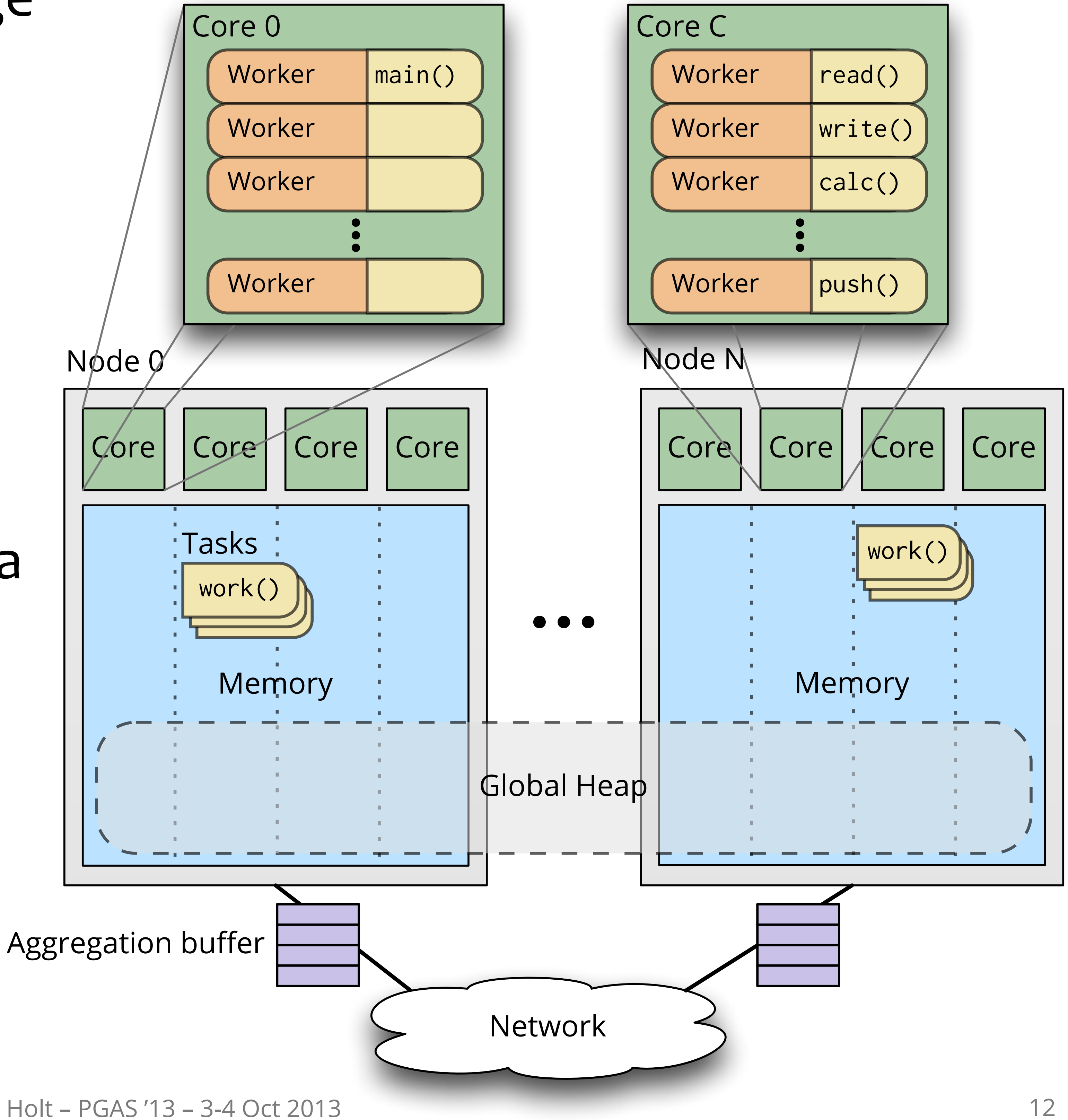
Global-view, single system image

C++11 library interface

Aggregated communication

Lightweight user-level threads for ***latency tolerance***

Access other cores' data *only* via ***delegate operations***



Grappa: a latency-tolerant PGAS runtime

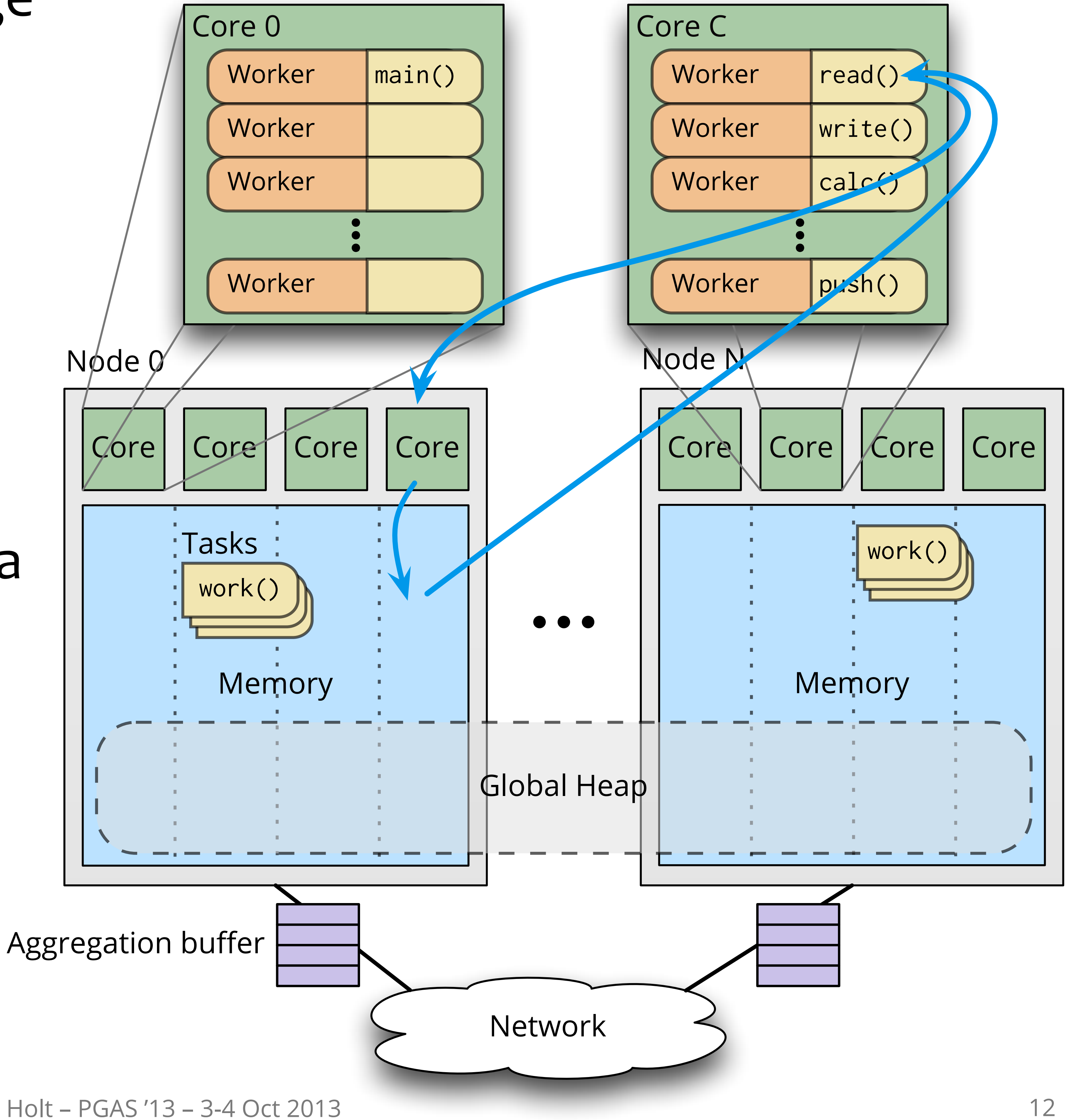
Global-view, single system image

C++11 library interface

Aggregated communication

Lightweight user-level threads for ***latency tolerance***

Access other cores' data *only* via ***delegate operations***



Grappa: a latency-tolerant PGAS runtime

Global-view, single system image

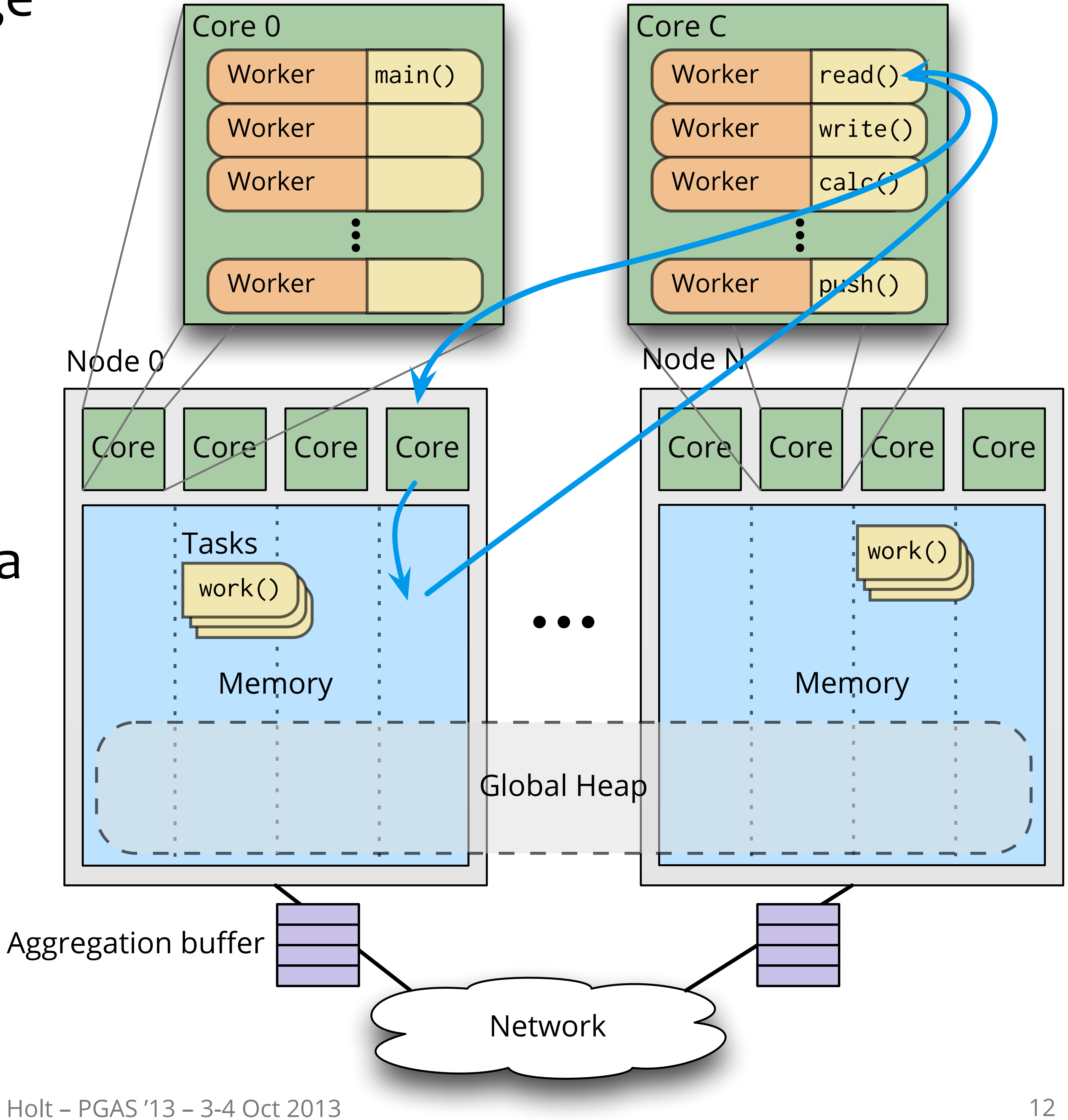
C++11 library interface

Aggregated communication

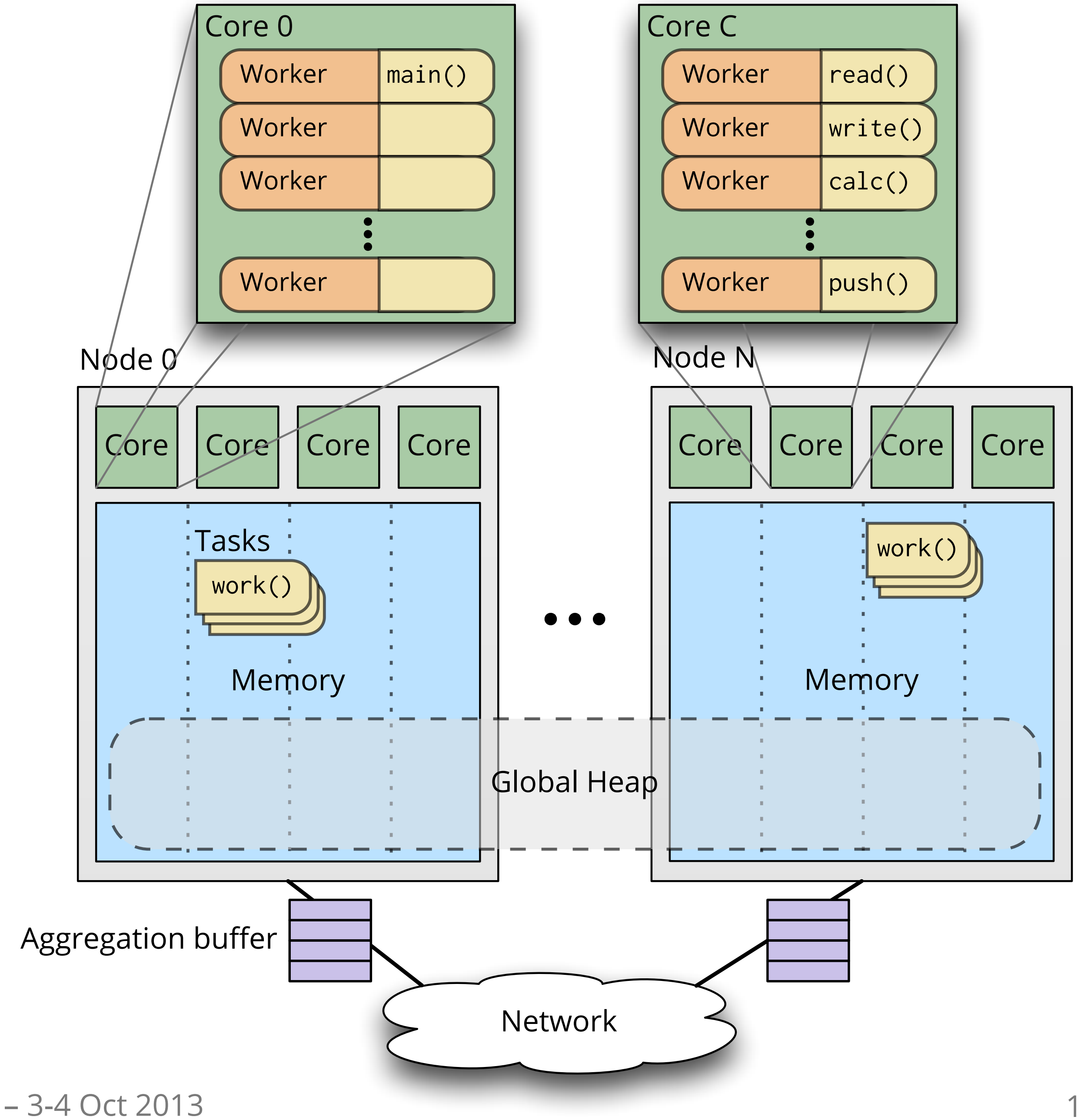
Lightweight user-level threads for ***latency tolerance***

Access other cores' data *only* via ***delegate operations***

Atomicity due to cooperative scheduling & delegates



Flat combining in Grappa



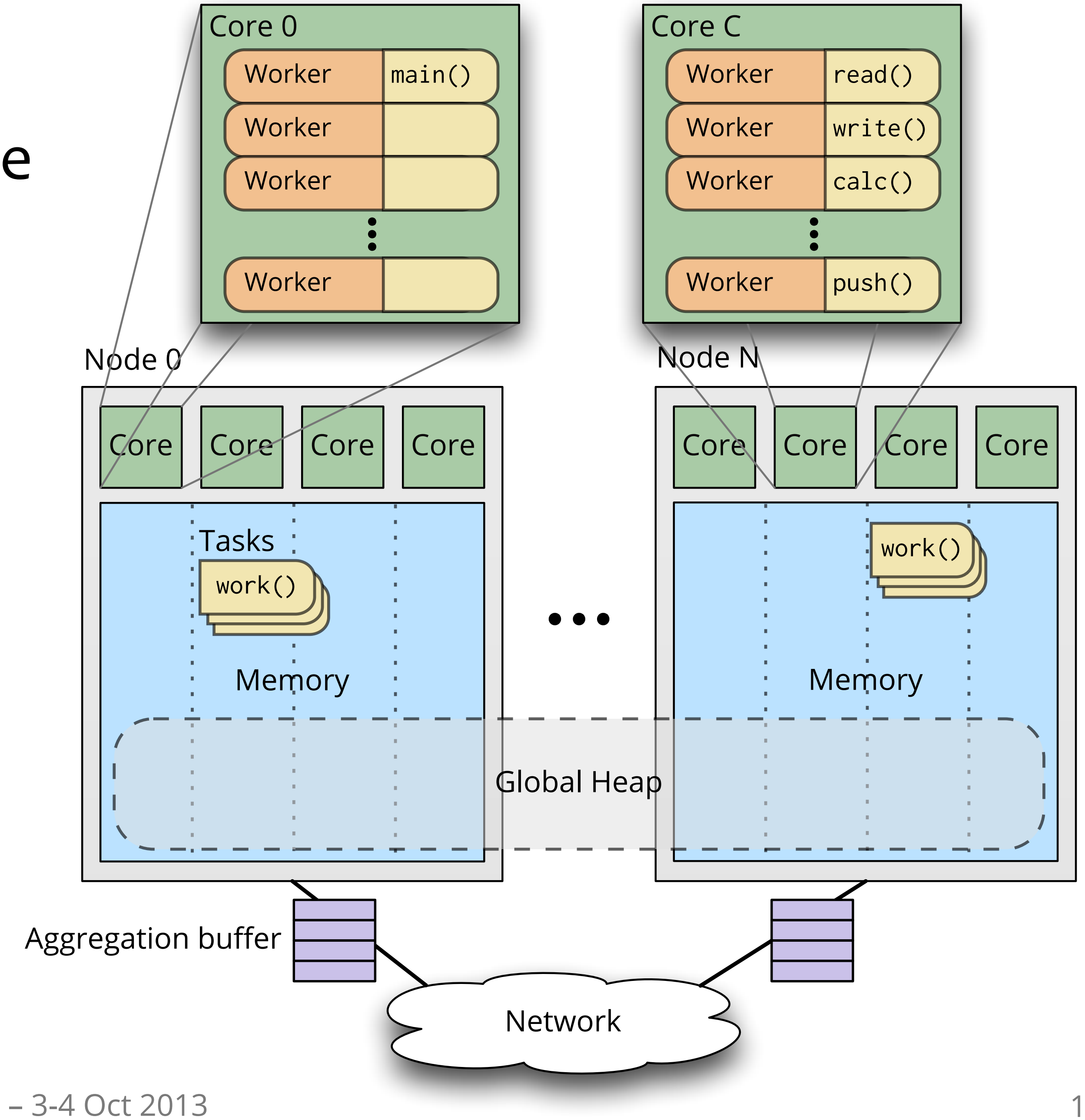
Flat combining in Grappa

Massive multithreading

- many workers, lots of combining
- lightweight suspend/wake

Synchronizing with Proxy is free

- **cooperative multithreading** within core
- only access other cores' memory via **delegate ops**



Flat combining

framework

Flat combining
framework

Reusable logic:

- manage freezing and creating fresh `Proxys`
- ensure always one combiner
- handle delivering results to blocked workers and waking them

Flat combining framework

Reusable logic:

- manage freezing and creating fresh Proxys
- ensure always one combiner
- handle delivering results to blocked workers and waking them

Each data structure defines:

- Proxy structure
- ops that combine into local proxy
- sync op that globally commits proxy's state

```
template< class T >
class GlobalStack {

  class Proxy: FCProxy< T > {
    // Local state for tracking requests
    T  pushed_values[1024];
    T* popped_results[1024];
    int npush, npop;

    // combining ops
    void push(T val);
    T pop();

    void sync() override; // commit globally
  };
};
```

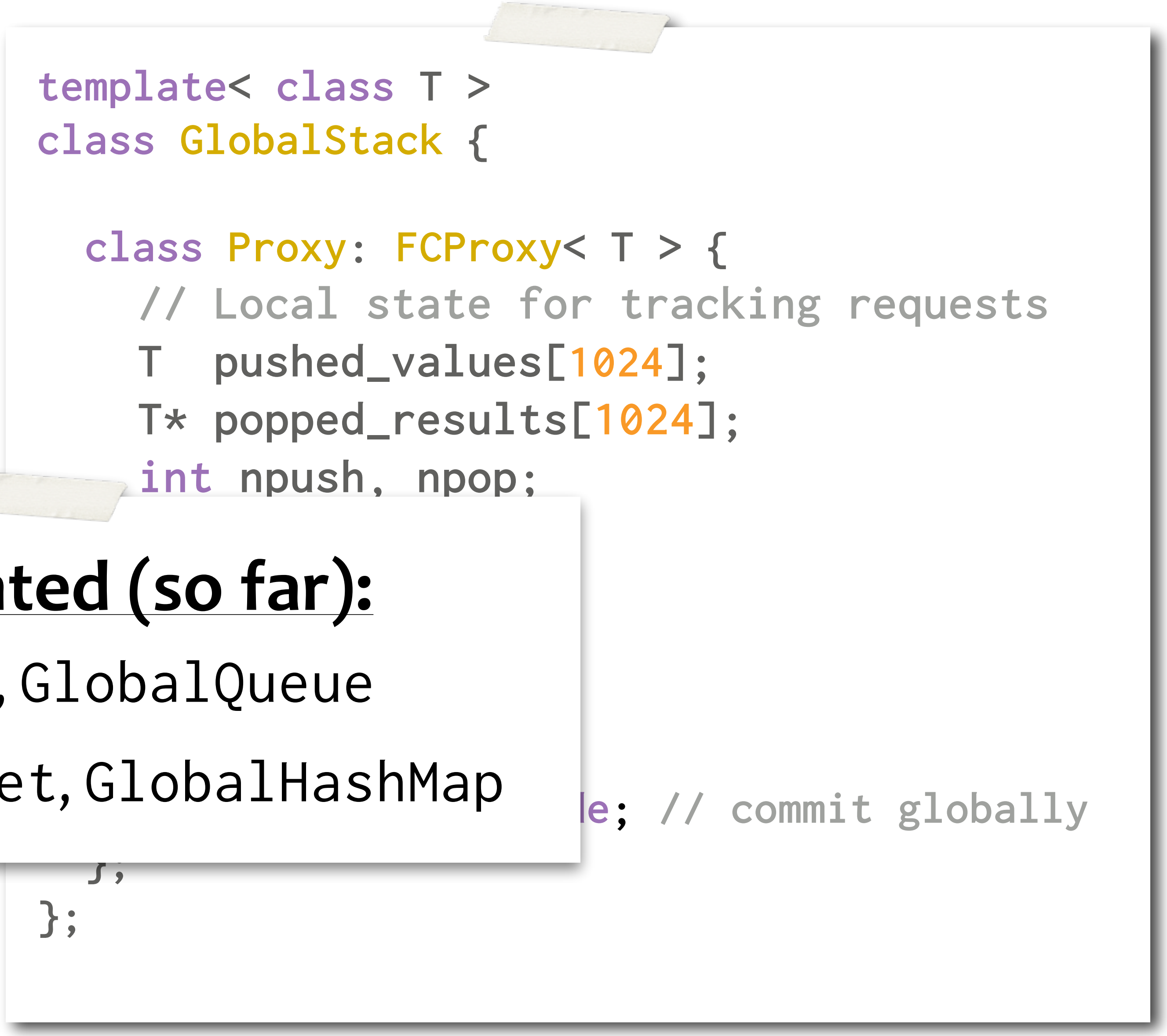
Flat combining framework

Reusable logic:

- manage freezing and creating fresh Proxys
- ensure always one combiner
- handle delivering blocked workers a

Each data structure

- Proxy structure
- ops that combine into local proxy
- sync op that globally commits proxy's state



```
template< class T >
class GlobalStack {

  class Proxy: FCProxy< T > {
    // Local state for tracking requests
    T  pushed_values[1024];
    T* popped_results[1024];
    int npush, npop;

                      le; // commit globally
  };
};
```

Implemented (so far):

- GlobalStack, GlobalQueue
- GlobalHashSet, GlobalHashMap

Flat combining
performance evaluation

Experimental setup

- Run on the PIC cluster at Pacific Northwest National Lab (PNNL)
- AMD Interlagos 2.1 GHz, 40 Gb Infiniband (Mellanox Connect-X 2, with QLogic switch)
- 16 cores per node, 2048 workers per core

Flat combining
performance evaluation

Methodology

Random throughput workload

- With/without flat combining
- Varied operation mix (push/pop, lookup/insert)

Experimental setup

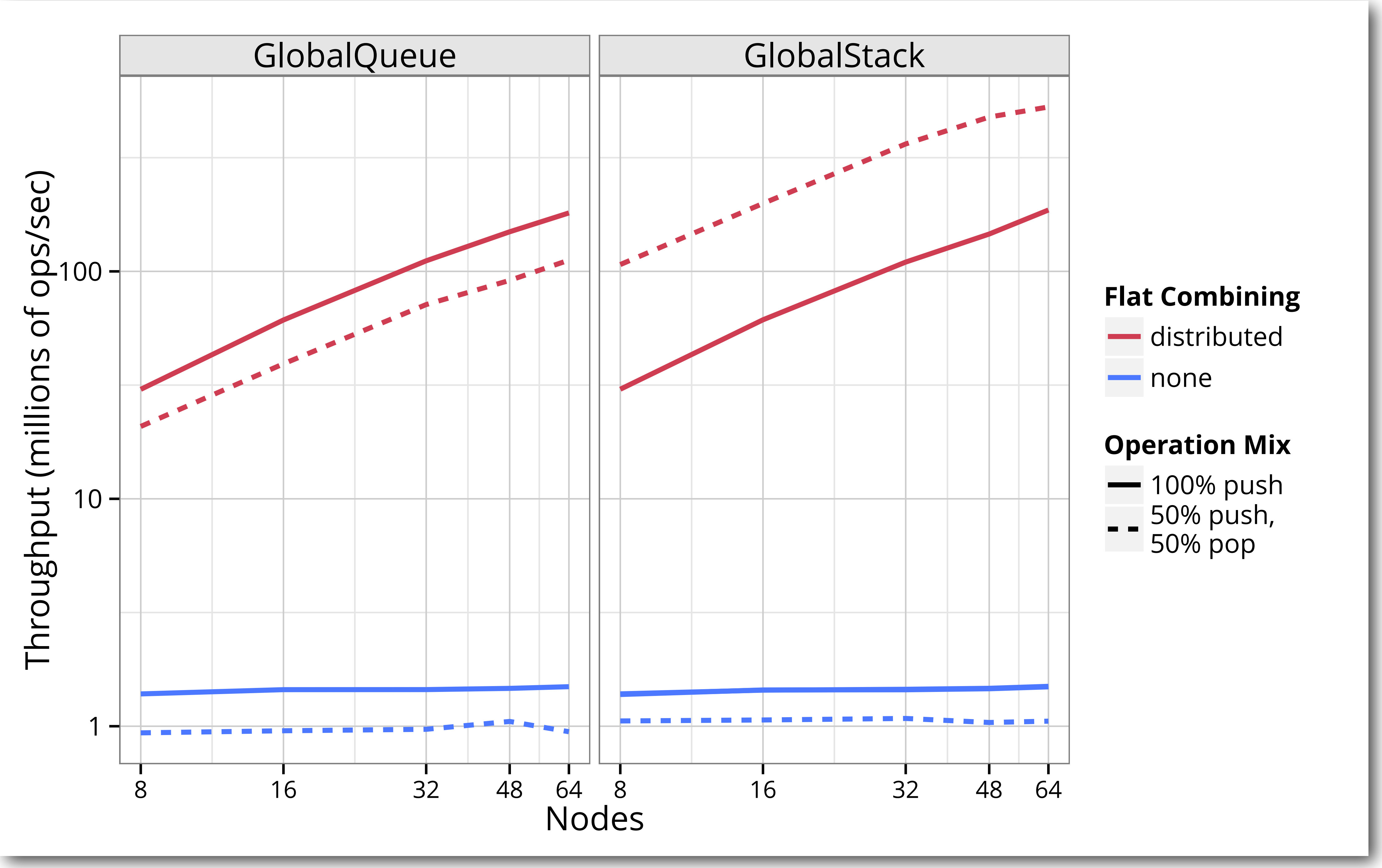
- Run on the PIC cluster at Pacific Northwest National Lab (PNNL)
- AMD Interlagos 2.1 GHz, 40 Gb Infiniband (Mellanox Connect-X 2, with QLogic switch)
- 16 cores per node, 2048 workers per core

```
void test(GlobalAddress<GlobalStack<long>> stack) {
  forall_global(0, 1<<28, [=](long i){
    if (choose_random(push_mix)) {
      stack->push(next_random<long>());
    } else {
      stack->pop();
    }
  });
}
```

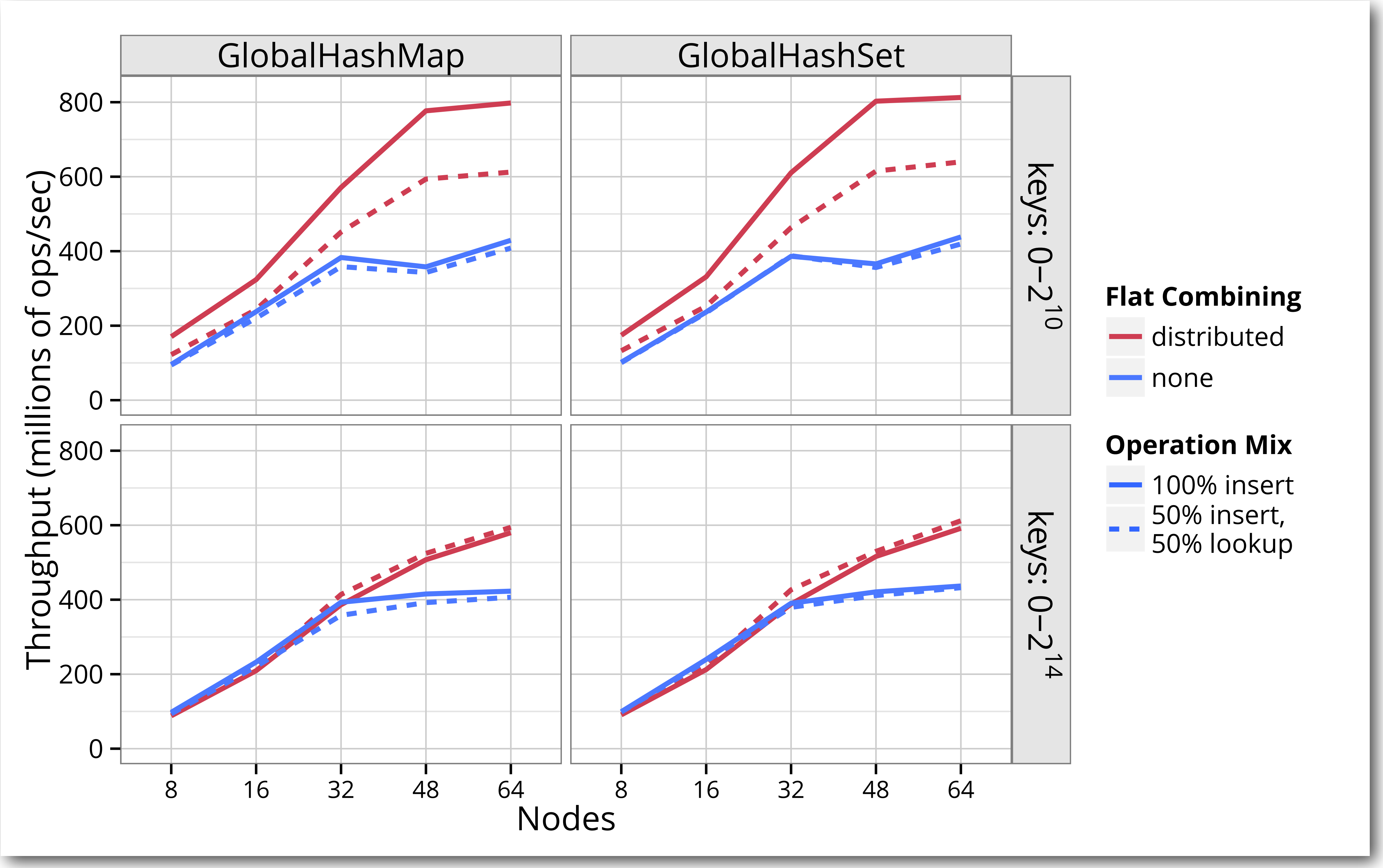
Flat combining

performance evaluation

Flat combining
performance evaluation



Flat combining
performance evaluation



Flat combining
performance evaluation

Experimental setup (same)

- Run on the PIC cluster at Pacific Northwest National Lab (PNNL) (AMD Interlagos 2.1 GHz, 40 Gb Mellanox InfiniBand)
- 16 cores per node, 2048 workers per core

Flat combining
performance evaluation

Experimental setup (same)

- Run on the PIC cluster at Pacific Northwest National Lab (PNNL) (AMD Interlagos 2.1 GHz, 40 Gb Mellanox InfiniBand)
- 16 cores per node, 2048 workers per core

Methodology: Apps

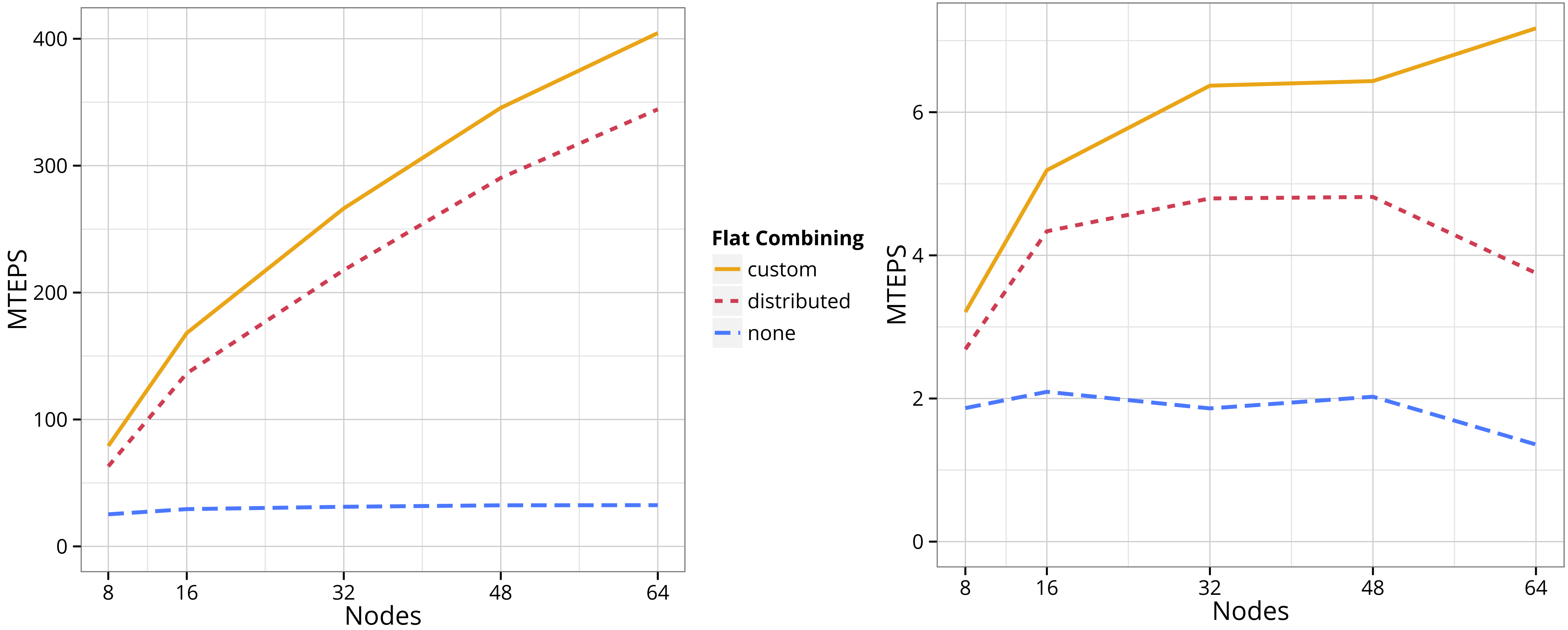
- Scale 26 Graph500-spec graph (64 M vertices, 1 B edges)
- **Breadth First Search** benchmark (find parent tree from random root)
- **Connected Components** (using 3-phase algorithm)

Flat combining

performance evaluation

Flat combining

performance evaluation



Breadth First Search

Connected Components

Flat Combining Synchronized Global Data Structures

take-aways

Brandon Holt, Jacob Nelson, Brandon Myers,
Preston Briggs, Luis Ceze, Simon Kahan, Mark Oskin

Flat Combining Synchronized Global Data Structures

take-aways

lesson learned from multicore:
cooperation beats **contention**

massive concurrency enables **sequential consistency at scale** thanks to Grappa’s latency tolerance

add new data structures easily with flat-combining framework

Brandon Holt, Jacob Nelson, Brandon Myers,
Preston Briggs, Luis Ceze, Simon Kahan, Mark Oskin

Flat Combining Synchronized Global Data Structures

take-aways

lesson learned from multicore:
cooperation beats **contention**

massive concurrency enables **sequential consistency at scale** thanks to Grappa's latency tolerance

add new data structures easily with flat-combining framework

questions?

Brandon Holt, Jacob Nelson, Brandon Myers,
Preston Briggs, Luis Ceze, Simon Kahan, Mark Oskin

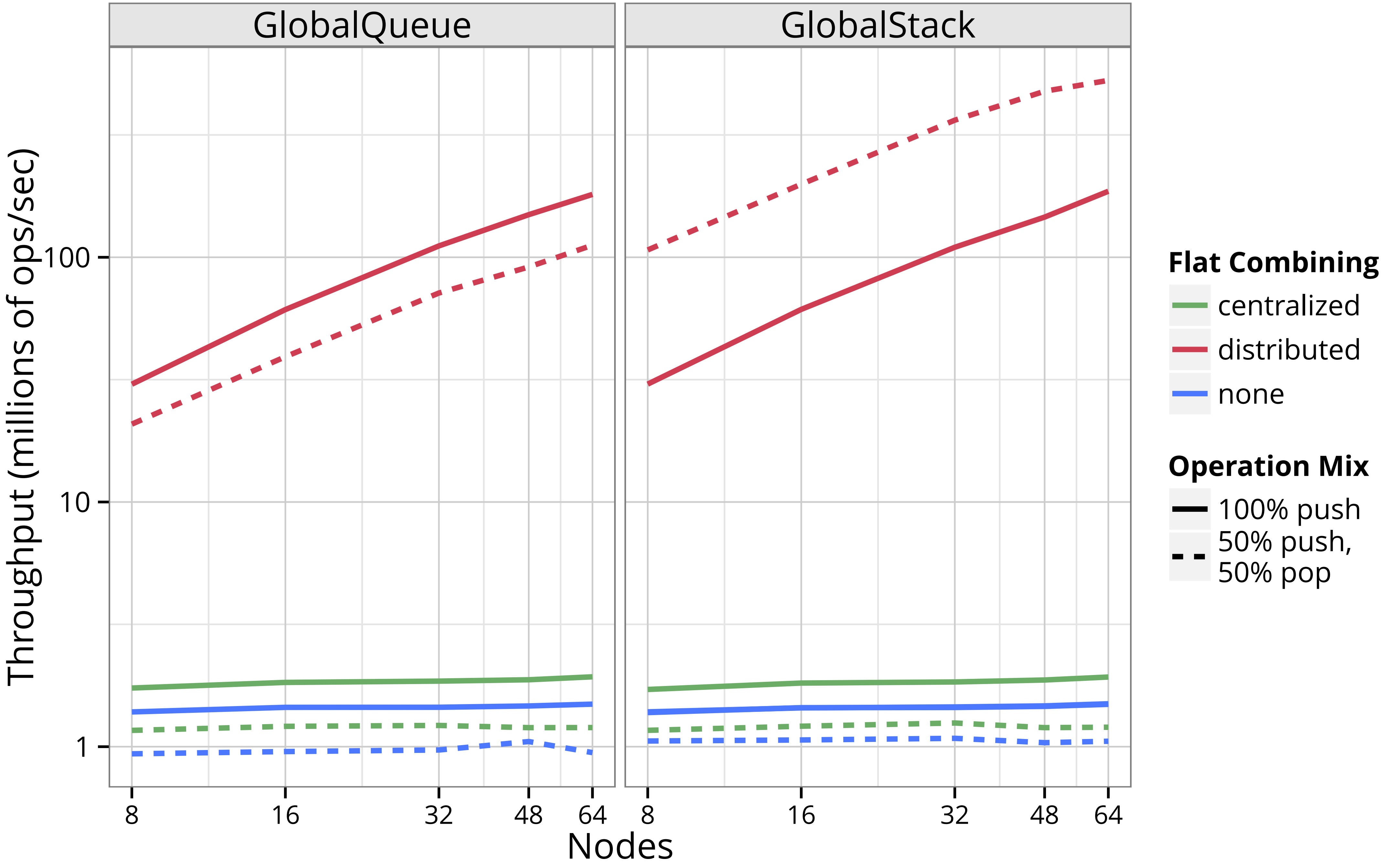
© Disney, Inc. *Fantasia (The Pastoral Symphony)*

Thank you!

© Disney, Inc. *Fantasia (The Pastoral Symphony)*

Flat combining

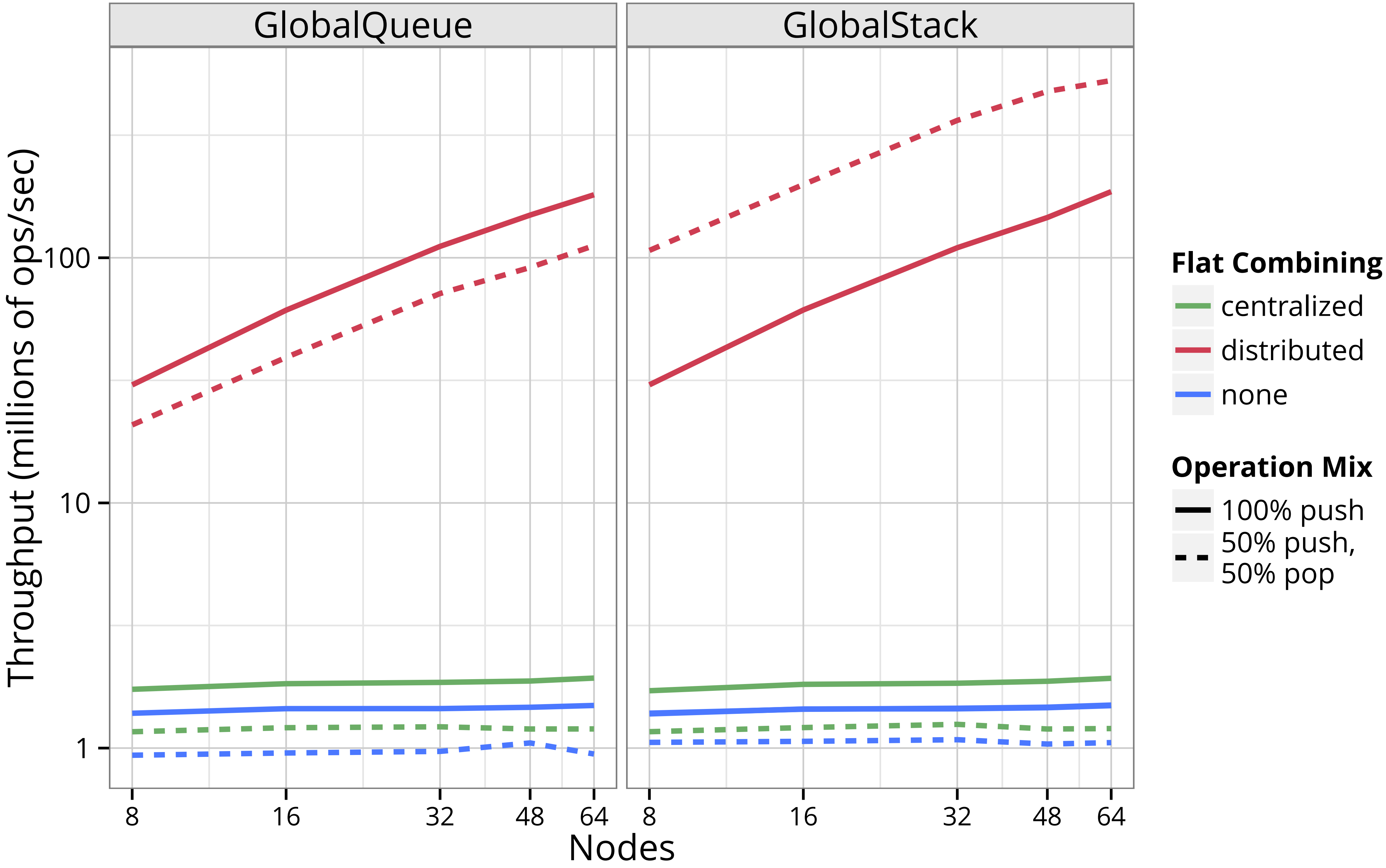
performance



Flat combining

performance

random throughput workload
– 2048 workers per core
– 16 cores per node


GlobalQueue
GlobalStack
Throughput (millions of ops/sec)
Nodes
Flat Combining
centralized
distributed
none
Operation Mix
100% push
50% push, 50% pop