Optimized Distributed Work-Stealing

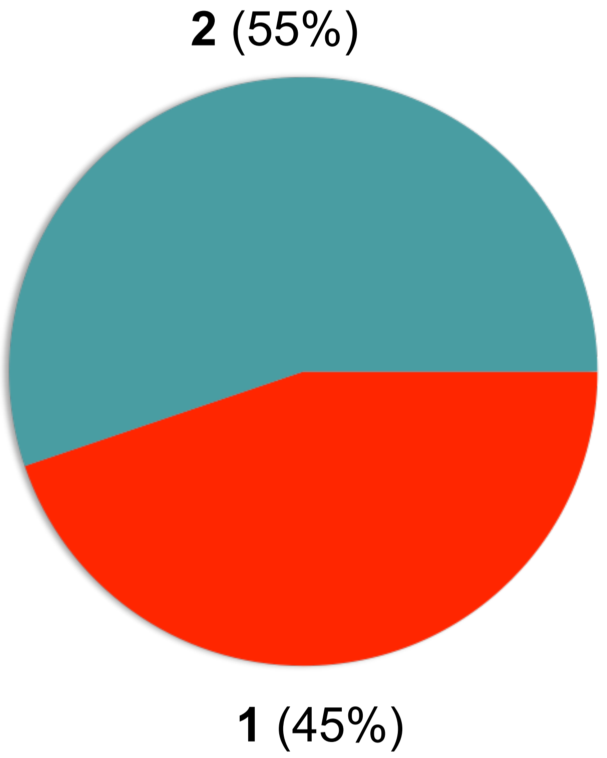
Vivek Kumar[1], Karthik Murthy[1], Vivek Sarkar[1], Yili Zheng[2]

1 Rice University

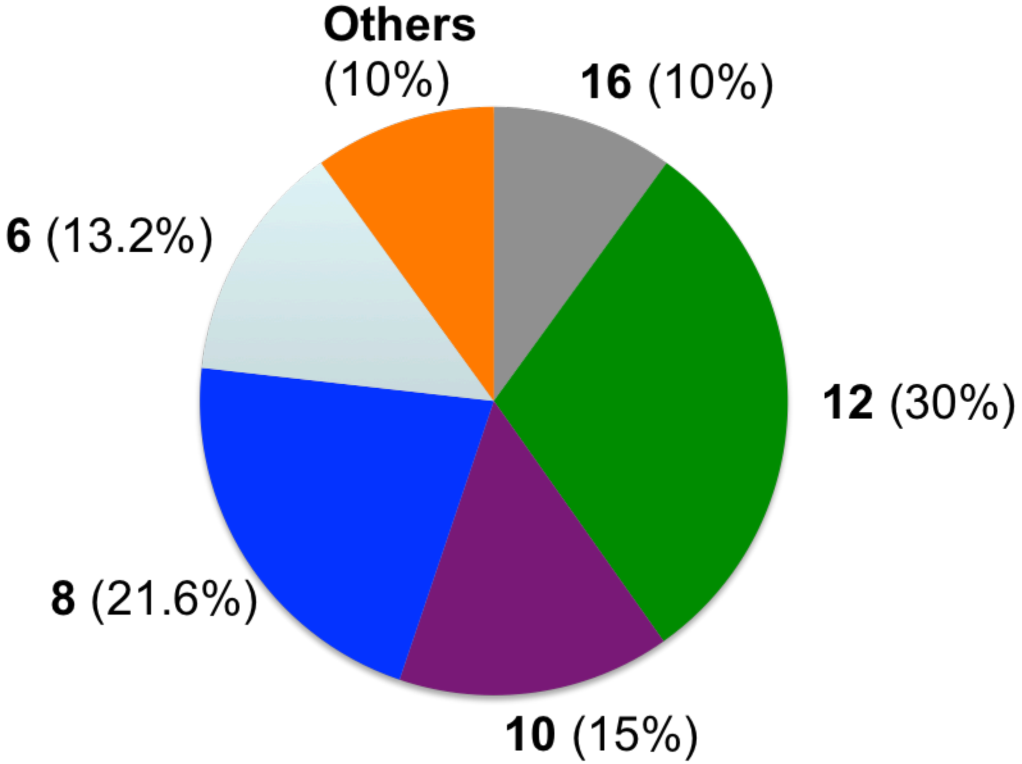
2 Lawrence Berkeley National Laboratory

Multicore Nodes in Supercomputers

Cores/Socket System Share in Top500



November **2006**



June **2016**

Graph plotted using the data obtained from https://www.top500.org/statistics/list/

Problem Statement

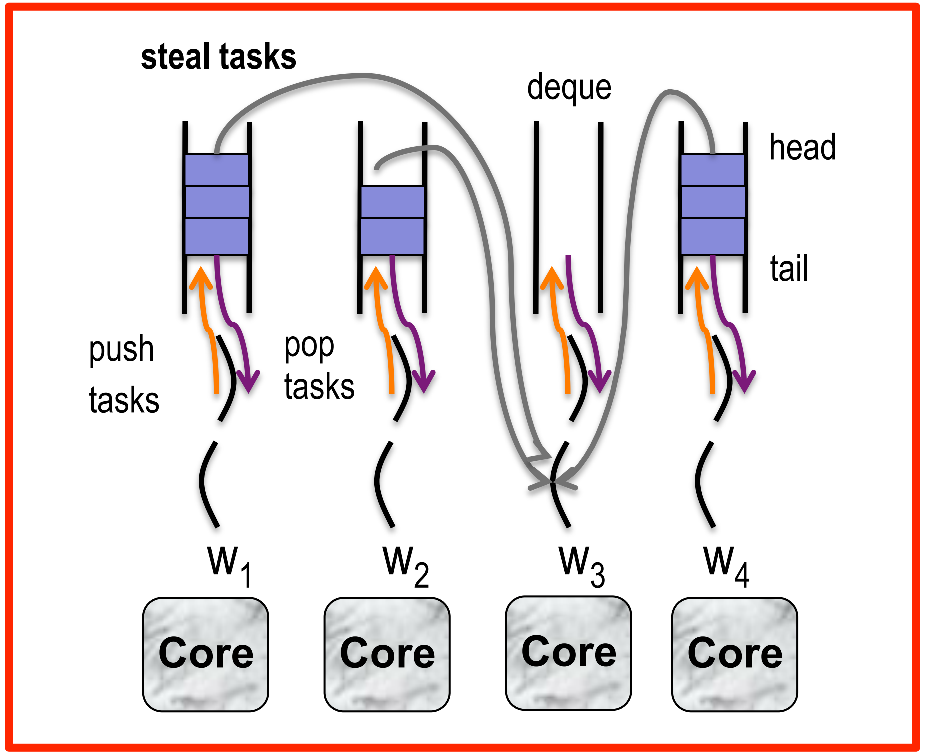
Productivity and Performance Challenge

- Productivity
  - – Several existing APIs for scientific computing
  - – Hard to parallelize complex irregular computations using existing APIs
    - Ideal candidate for runtime based global load-balancing
- Performance on multicore nodes
  - – Using a process per core (e.g., MPI everywhere) on a node not scalable
  - – Hybrid programming using thread pool per node
    - How to design a high performance implementation of global load-balancing

Contributions

- ✔ Library-based API in a PGAS library to express irregular computations
  - C++11 lambda function based API that provides serial elision
- ✔ Novel implementation of distributed work-stealing
  - That introduces a new victim selection policy that avoid all inter-node failed steals
- ✔ Detailed performance study
  - That demonstrates the benefit using scaling irregular applications up to 12k cores of Edison supercomputer
- ✔ Results
  - That shows that our approach delivers performance benefits up to 7%

Load Balancing using Work-Stealing

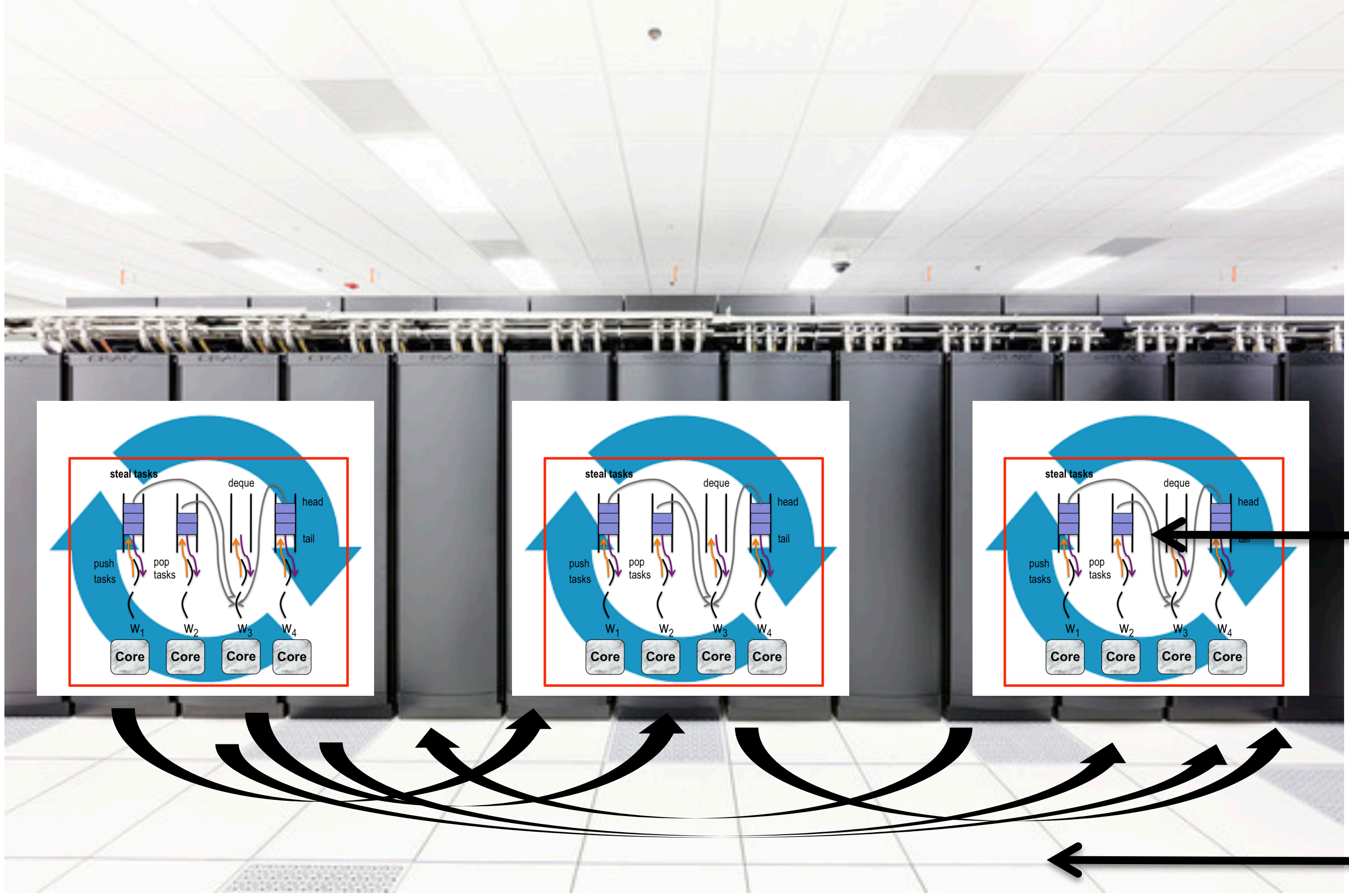


Work-stealing in a thread pool

- Thread pool **(intra-node)** based implementations perform stealing using low overhead CAS operations

Motivating Analysis

Distributed Work-Stealing



- **Inter-node** steals are much costlier than **intra-node** steals

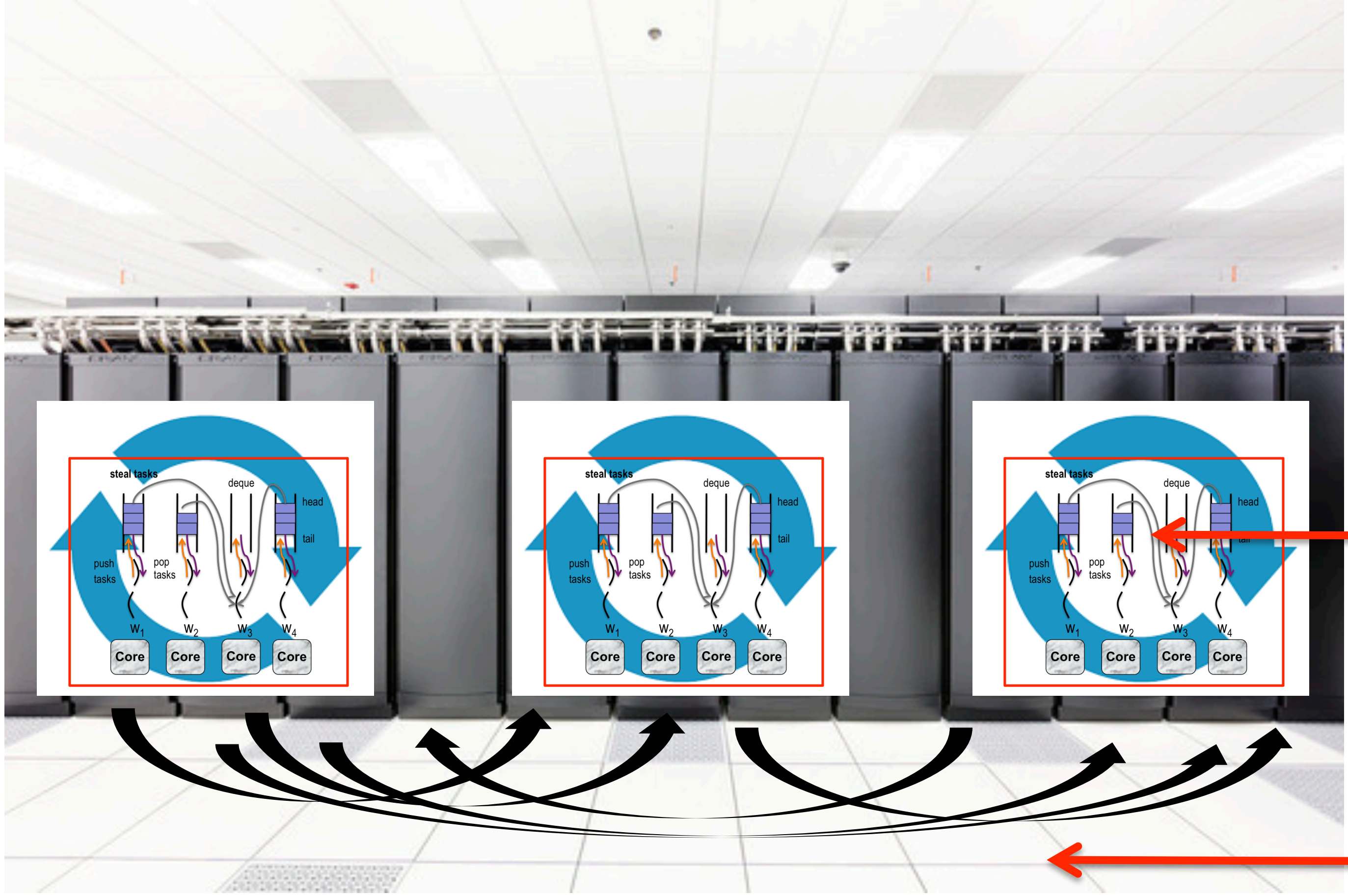
Intra-node steals

Inter-node steals

Motivating Analysis

Failed Steal Attempts

- Thief fails to steal a task from victim



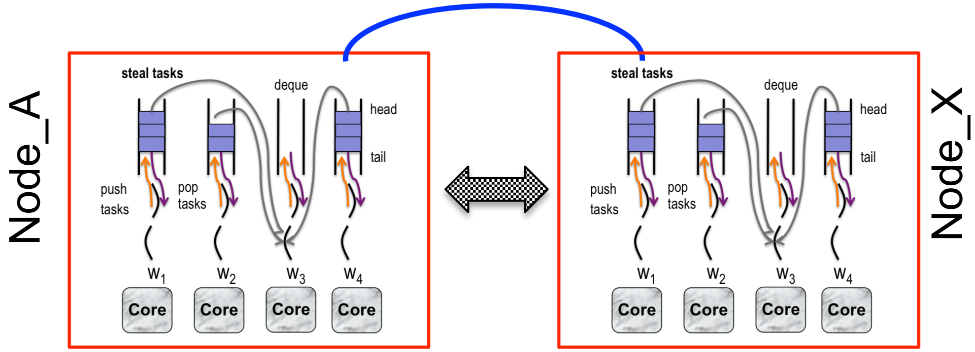
Inter-node failed steals are more costly than **intra-node** steals

Chances to fail with same victim multiple times

Intra-node failed steals

Inter-node failed steals

Our Approach



One process with a thread pool at each node

- Use HabaneroUPC++ PGAS library for multicore cluster [Kumar et. al., PGAS 2014]
  - Several asynchronous tasking APIs
- Provide a programming model to express irregular computation
- Implement a high performance distributed work-stealing runtime that **completely removes** all inter-node failed steal attempts

Implementation

HabaneroUPC++ Programming Model

```
asyncAny ( [=] {

        irregular_computation();

}); //distributed work-stealing
```

- C++11 lambda-function based API
- Provides **serial elision** and improves productivity

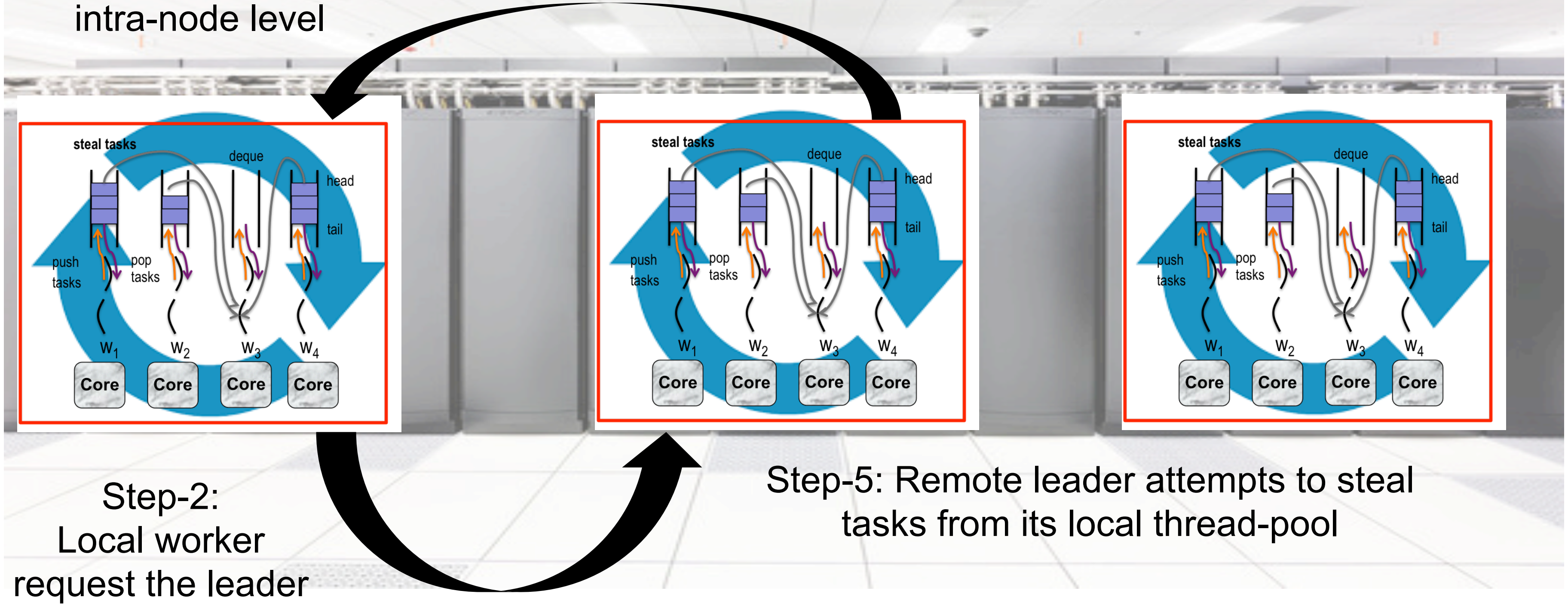
Implementation

Distributed Work-Stealing Runtime

- Two different implementations in HabaneroUPC++
- BaselineWS
  - Uses prior work
- SuccessOnlyWS
  - Extends BaselineWS by using a novel victim selection policy that complete removes all inter-node failed steals

Implementation

BaselineWS in HabaneroUPC++



Step-1:
Failed steal at
intra-node level

Step-6: Remote victim can send tasks,
else it's a inter-node failed steal

Step-2:
Local worker
request the leader
for inter-node steal

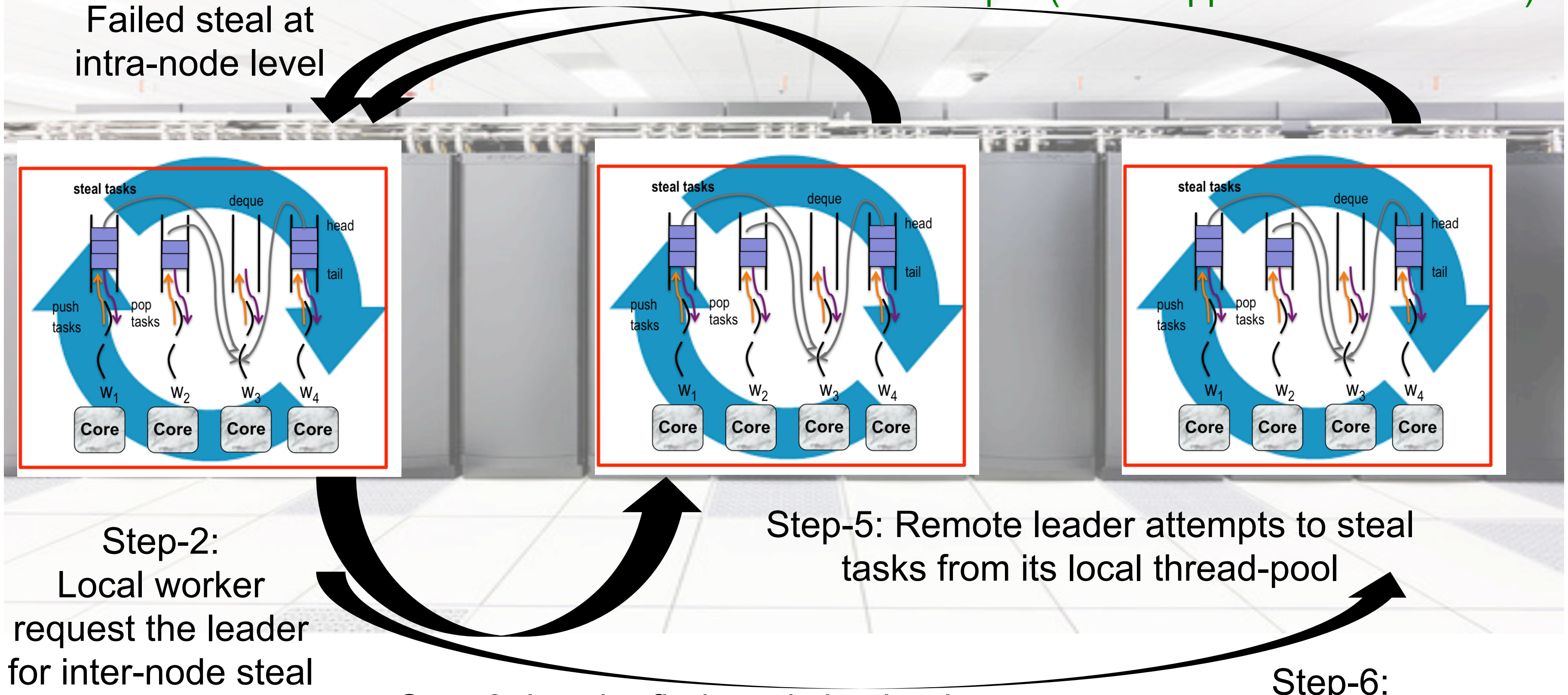
Step-5: Remote leader attempts to steal
tasks from its local thread-pool

Step-3: Leader finds a victim that has
sufficient number of tasks (RDMA)

Step-4: Lock and wait for tasks from victim

Implementation

SuccessOnlyWS in HabaneroUPC++



Step-1:
Failed steal at
intra-node level

Step-7: One or more remote victim will send tasks. Break out of Step-5 (also if application terminates)

Step-2:
Local worker
request the leader
for inter-node steal

Step-5: Remote leader attempts to steal tasks from its local thread-pool

Step-3: Leader finds a victim that has sufficient number of tasks (RDMA)

Step-6:
Repeat Step-3 and Step-4

Step-4: Asynchronous task request

Experimental Evaluation

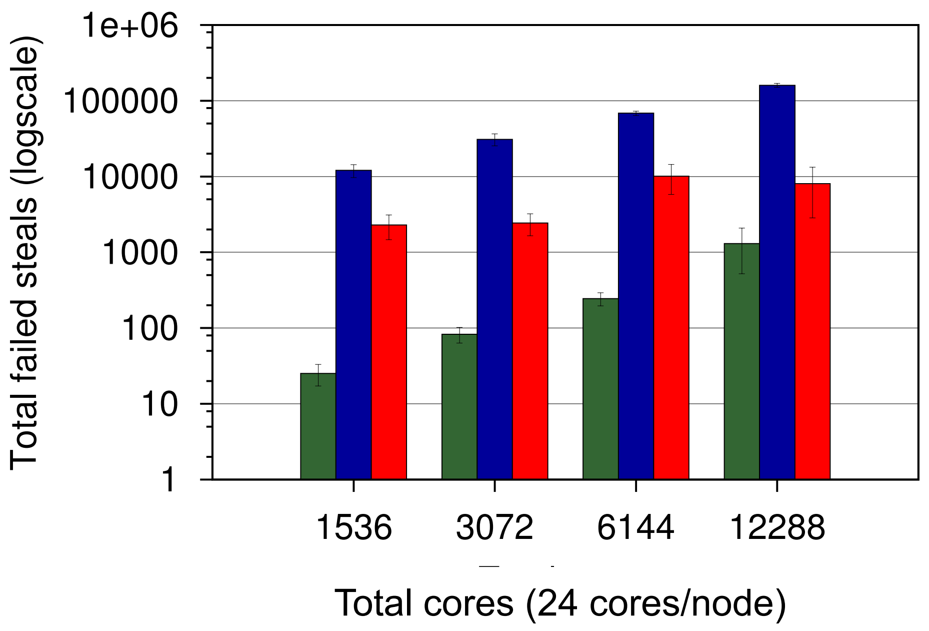
Methodology

- Benchmarks
  - Two UTS trees T1WL and T3WL
  - NQueens
- Computing infrastructure
  - Edison supercomputer at NERSC
    - 2x12 cores per node

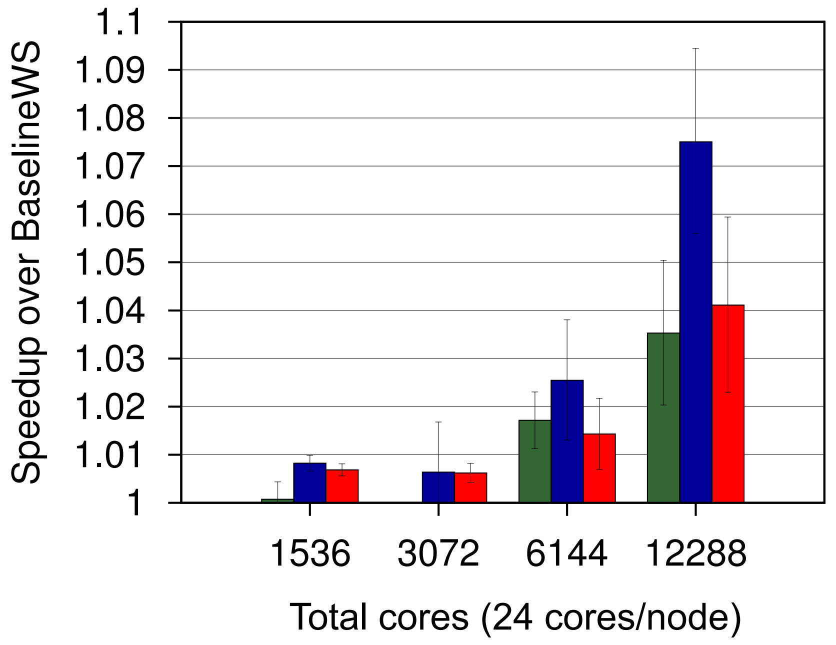
Experimental Evaluation

Results

Higher inter-node failed steals in BaselineWS **=>** Better performance in SuccessOnlyWS



(a) Total inter-node failed steals in BaselineWS



(b) SuccessOnlyWS speedup over BaselineWS

🟩 T1WL 🟦 T3WL 🟥 NQueens

Note: More results are available in the paper

Summary and Conclusion

- Inter-node steals are costlier than intra-node steals
- Failed inter-node steals could hamper performance
- C++11 lambda function based API to in HabaneroUPC++ to express complex irregular computation that can participate in distributed work-stealing
- A novel implementation of distributed work-stealing runtime in HabaneroUPC++ PGAS library that completely removes all inter-node failed steals
- Our novel runtime delivers performance benefits up to 7%

Backup Slides

Related Work

Existing Techniques for Inter-node Stealing

- Thread pool based hybrid runtimes [Lifflander et. al., HPDC’12, Paudel et. al., ICPP’13]
- Communication worker maintain ready queue of tasks even before a remote request arrives [Paudel et. al., ICPP’13]
- Load-aware steal attempts to *reduce* chances of failure [Dinan et. al., ICPP’08]
- First try random victims and on failing contact set of victims (lifelines) that promises to send tasks whenever they have it ready [Saraswat et. al., PPoPP’11]

Inter-node Steal Request from Thief

BaselineWS Runtime

SuccessOnlyWS Runtime

```
procedure Steal_AsyncAny
    while (global termination is not detected)
        V = get a random remote rank
        if (V has declared task availability in PGAS space) // RDMA
            if (I did not try to steal from V)
                queue my rank at V
            if TryLock (V) is success
                save my rank at V
                wait until V send tasks or decline
                Unlock (V)
        break from while loop if I just received asyncAny tasks
    if I receive asyncAny tasks from any victim
        forget that I contacted this victim
        reset my task receiving status
```

Inter-node Task Transfer from Victim

```
procedure Send_AsyncAny
    while (there are pending inter-node steal requests)
        T = get rank of one of the queued remote thief
        steal tasks from my local workers and send to T
        forget that T contacted me
        break out of the while loop if local steal failed
    T = get rank of the only waiting remote thief
    steal tasks from local workers and send to T
    declare that now I don’t have any waiting remote thief
    publish in PGAS space asyncAny count at my place
```

🟥 BaselineWS Runtime

🟩 SuccessOnlyWS Runtime