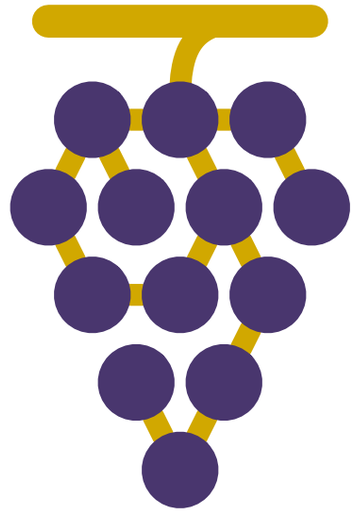


Latency-Tolerant Software Distributed Shared Memory

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

University of Washington

USENIX ATC 2015

July 9, 2015

25 years ago…

Memory Coherence in Shared Virtual Memory Systems

KAI LI
Princeton University
and
PAUL HUDAK
Yale University

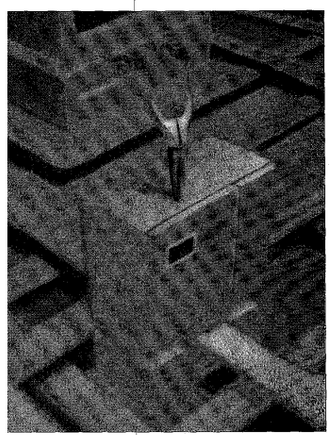
The m
couple
for so
on th
memo

Categ
chitec
Distri
ment-
Desig

Gener

Additi
progra

Feature



TreadMarks: Shared Memory Computing on Networks of Workstations

Cristiana Amza, Alan L. Cox, Sandhya Dwarkadas, Pete Keleher, Honghui Lu, Ramakrishnan Rajamony, Weimin Yu, and Willy Zwaenepoel
Rice University

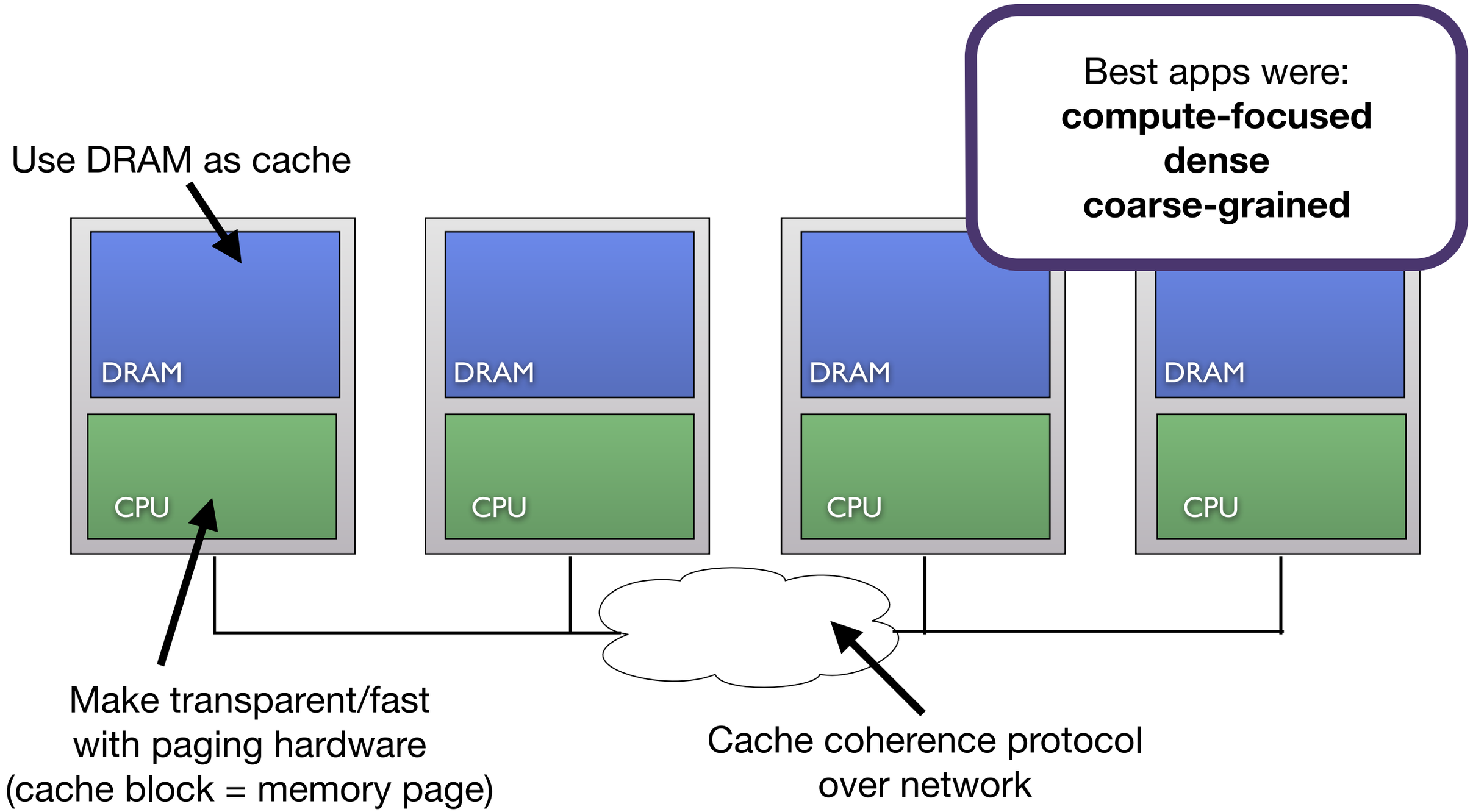
Shared memory facilitates the transition from sequential

High-speed networks and improved microprocessor performance are making networks of workstations an appealing vehicle for parallel computing. By relying solely on commodity hardware and software, networked workstations can offer parallel processing at a relatively low cost.

A network-of-workstations multiprocessor can be realized as a processor bank in which dedicated processors provide computing cycles, or it can consist of a dynamically varying set of machines that perform long-running computations during idle periods. In the latter case, the hardware cost is essentially zero, since many organizations already have extensive workstation networks.

In terms of performance, networked workstations approach or exceed supercomputer performance for some applications. These loosely coupled multiprocessors will by no means replace the more tightly coupled designs, which, because of lower latencies and higher bandwidths, are more efficient for applications with stringent synchronization and communication requirements. However, advances in networking technology and processor performance are expanding the class of applications that can be executed efficiently on networked workstations.

Distributed shared memory, then…



Distributed shared memory, now?

- New “data-intensive” applications:
  - Social network analysis
  - Machine learning
  - Bioinformatics
  - …
- Data access, not compute, is the hard part
- Locality can be hard to find
- A bad fit for DSM of 25 years ago, so community has explored other abstractions:
  - Spark, GraphLab, Naiad, etc.

S.cerevisiae
[von Mering et al.]

Grappa:
Software distributed shared memory for data-intensive apps

Your next data-intensive application or framework!

MapReduce | **GraphLab** | **Relational Query Engine** | **Irregular apps, native code, etc...** | **....**

Grappa – Distributed shared memory

Linux x86 node — Commodity network — Linux x86 node

What makes this hard?

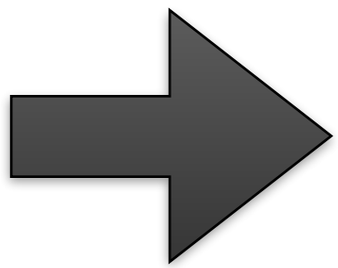
Lack of locality

Parallelism is abundant in data-intensive applications!

Small tasks

What makes this hard?

Lack of locality

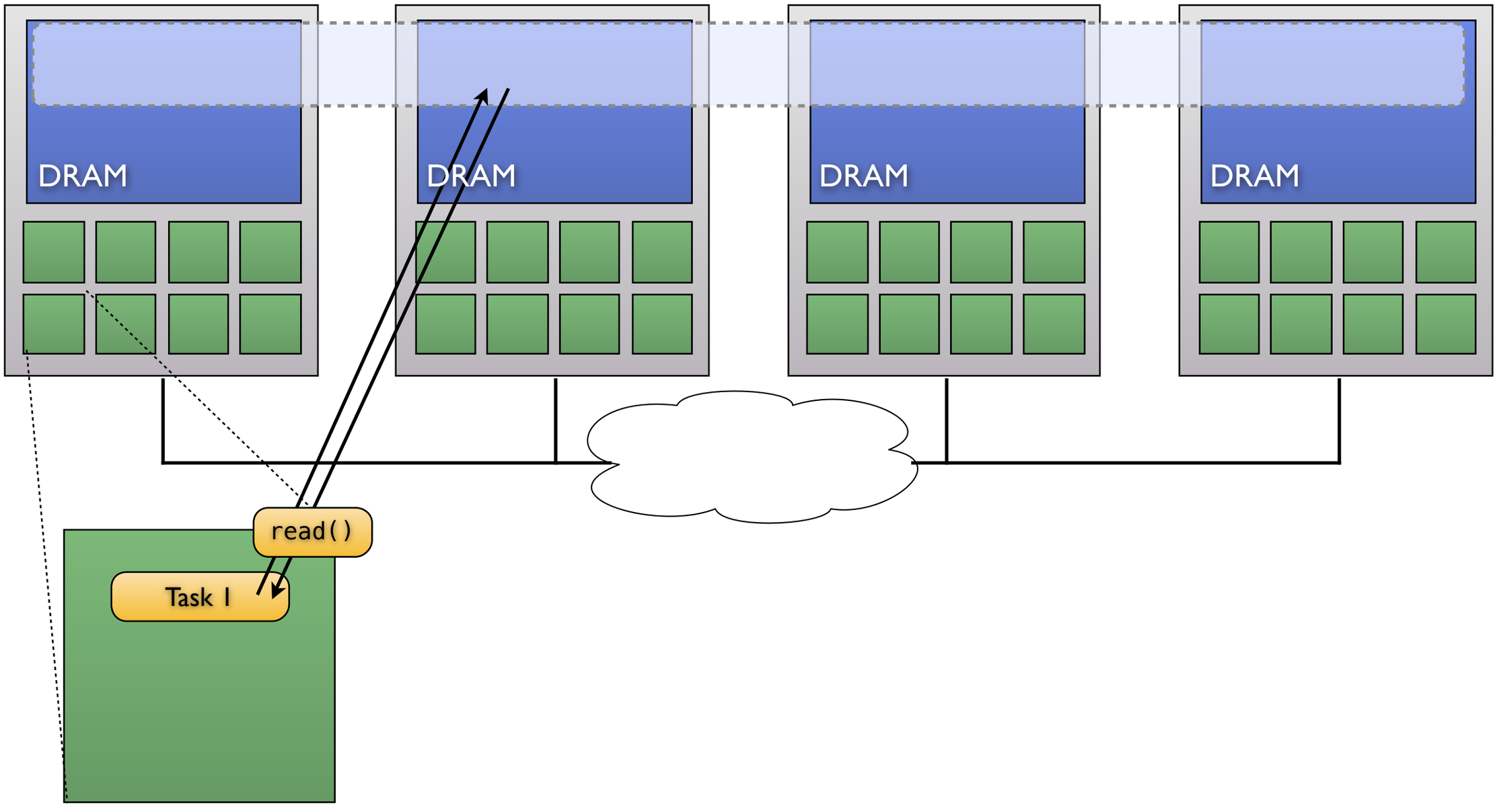


Use parallelism to hide latency!

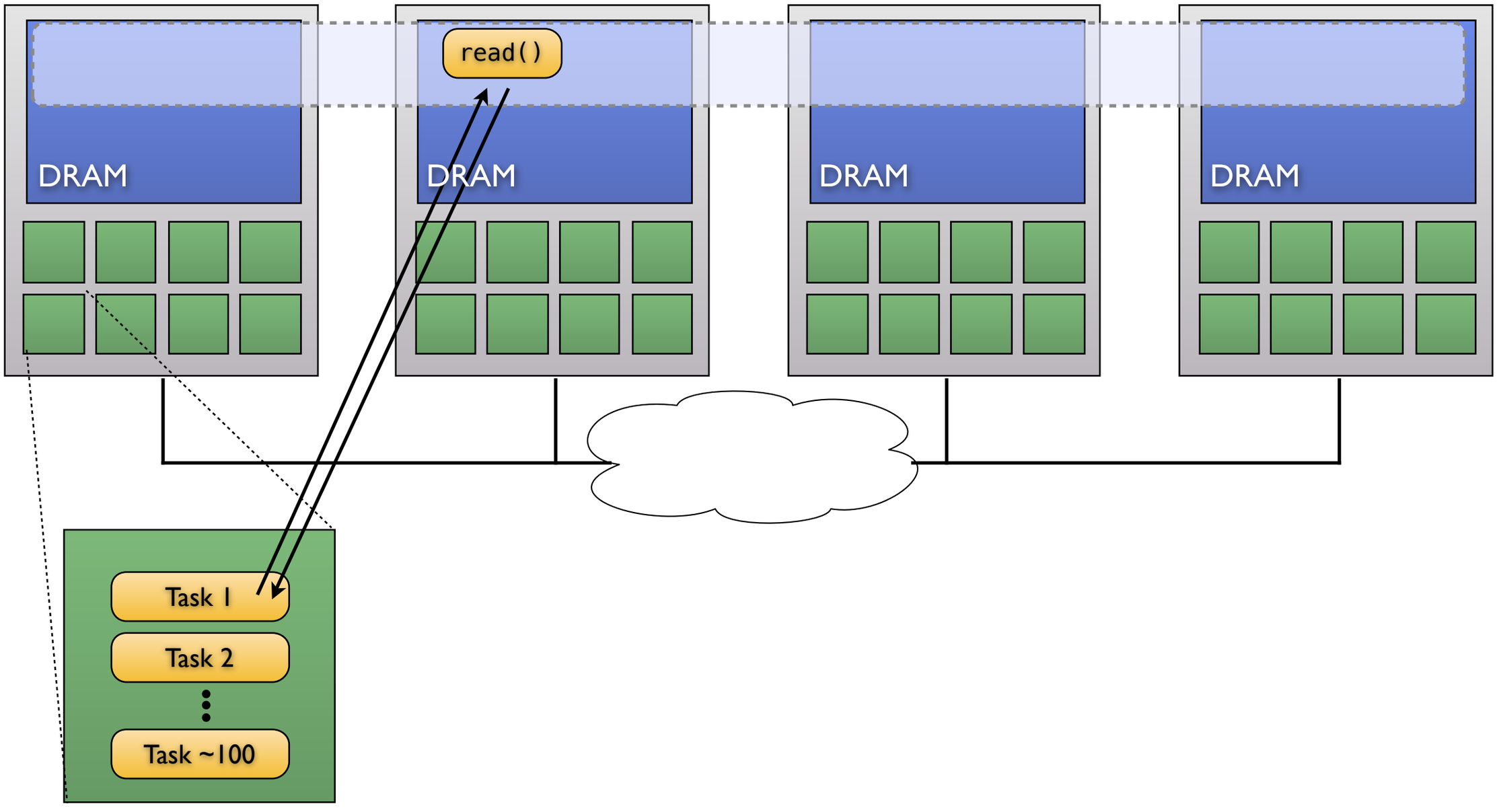
Small messages

Small tasks

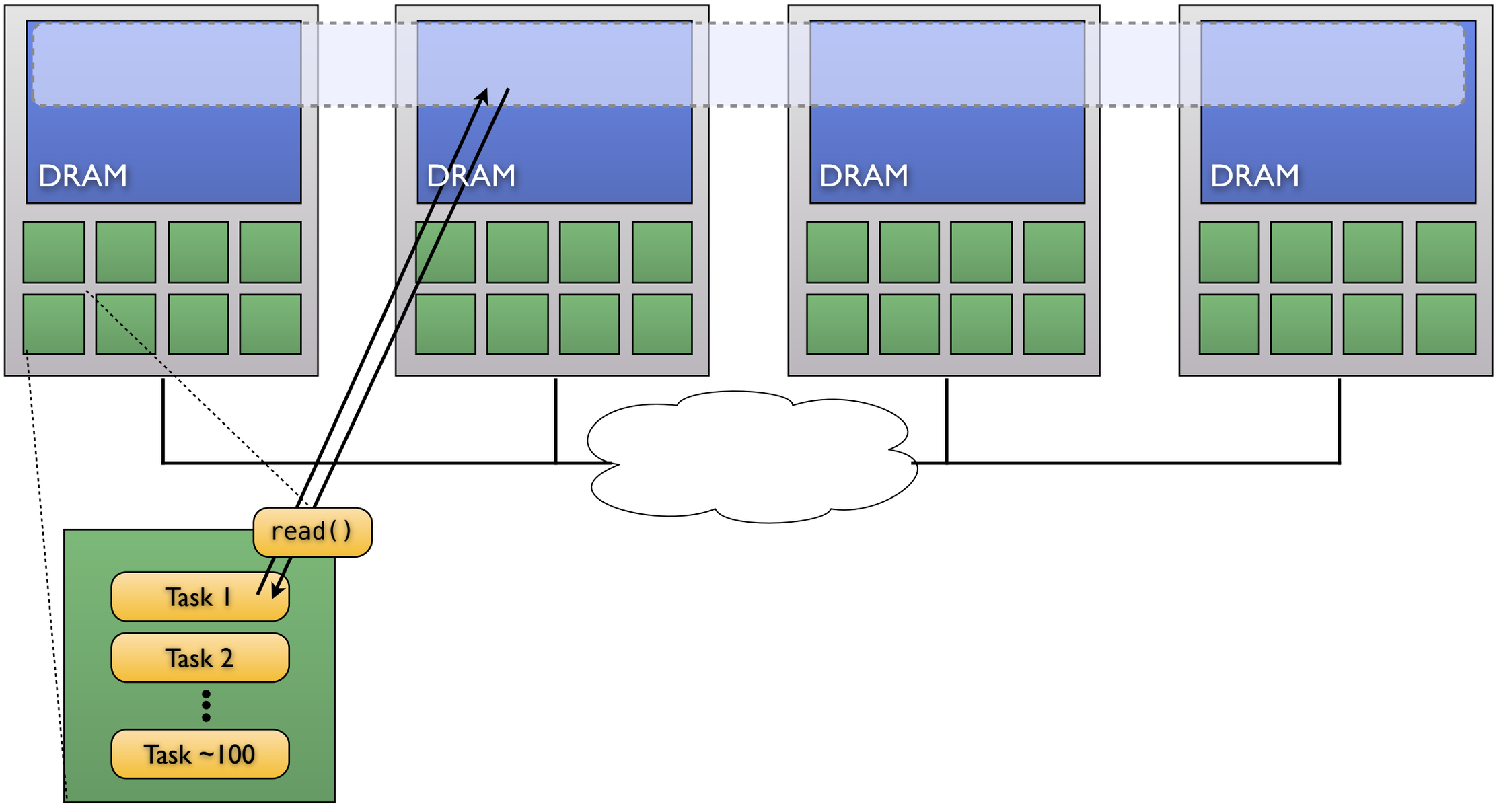
A remote read with latency tolerance



A remote read with latency tolerance

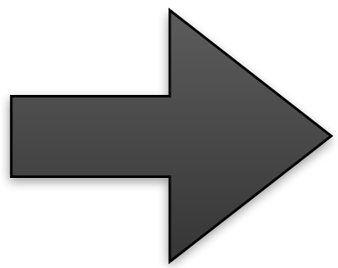


A remote read with latency tolerance



What makes this hard?

Lack of locality

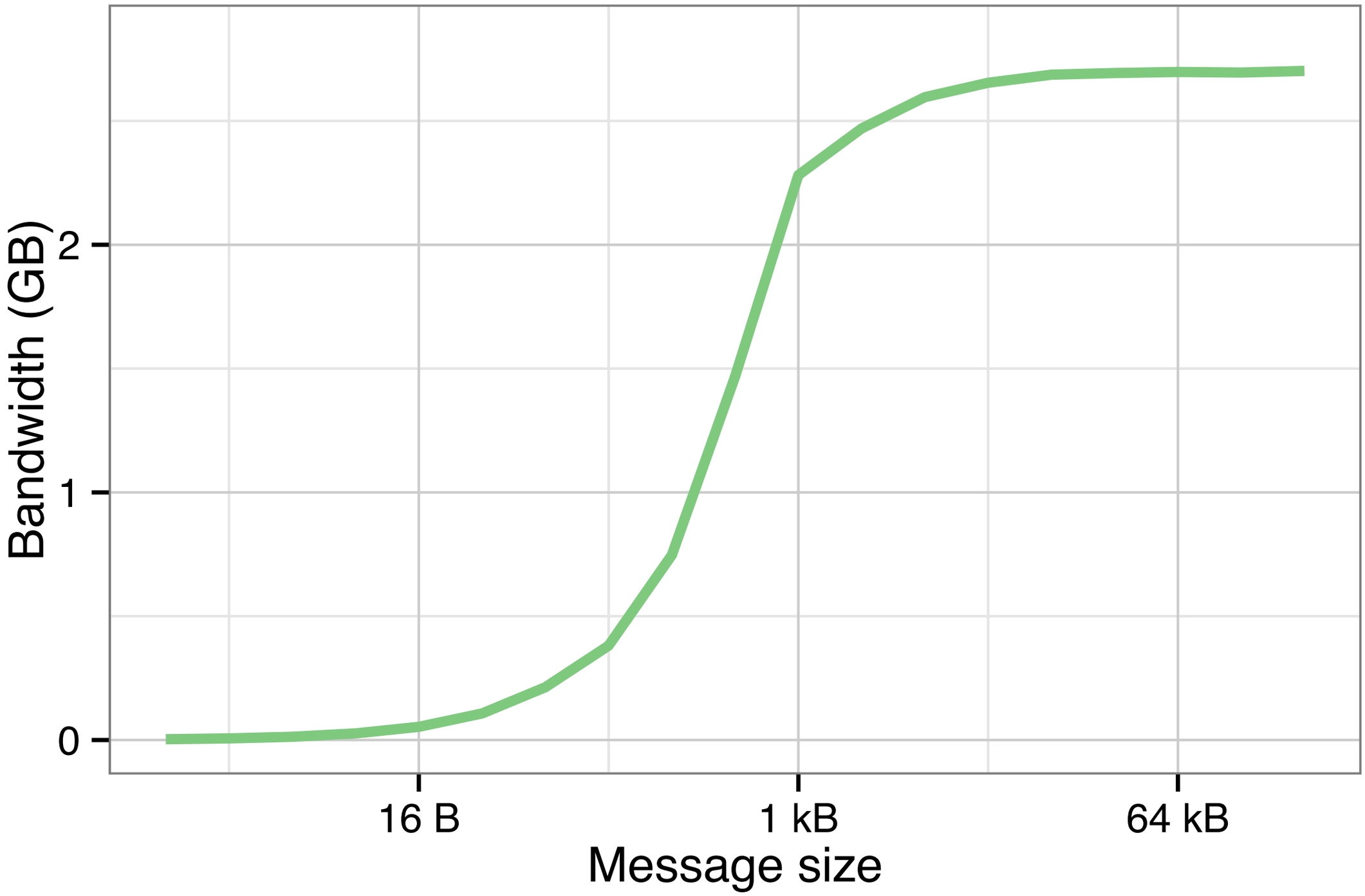


Use parallelism to hide latency!

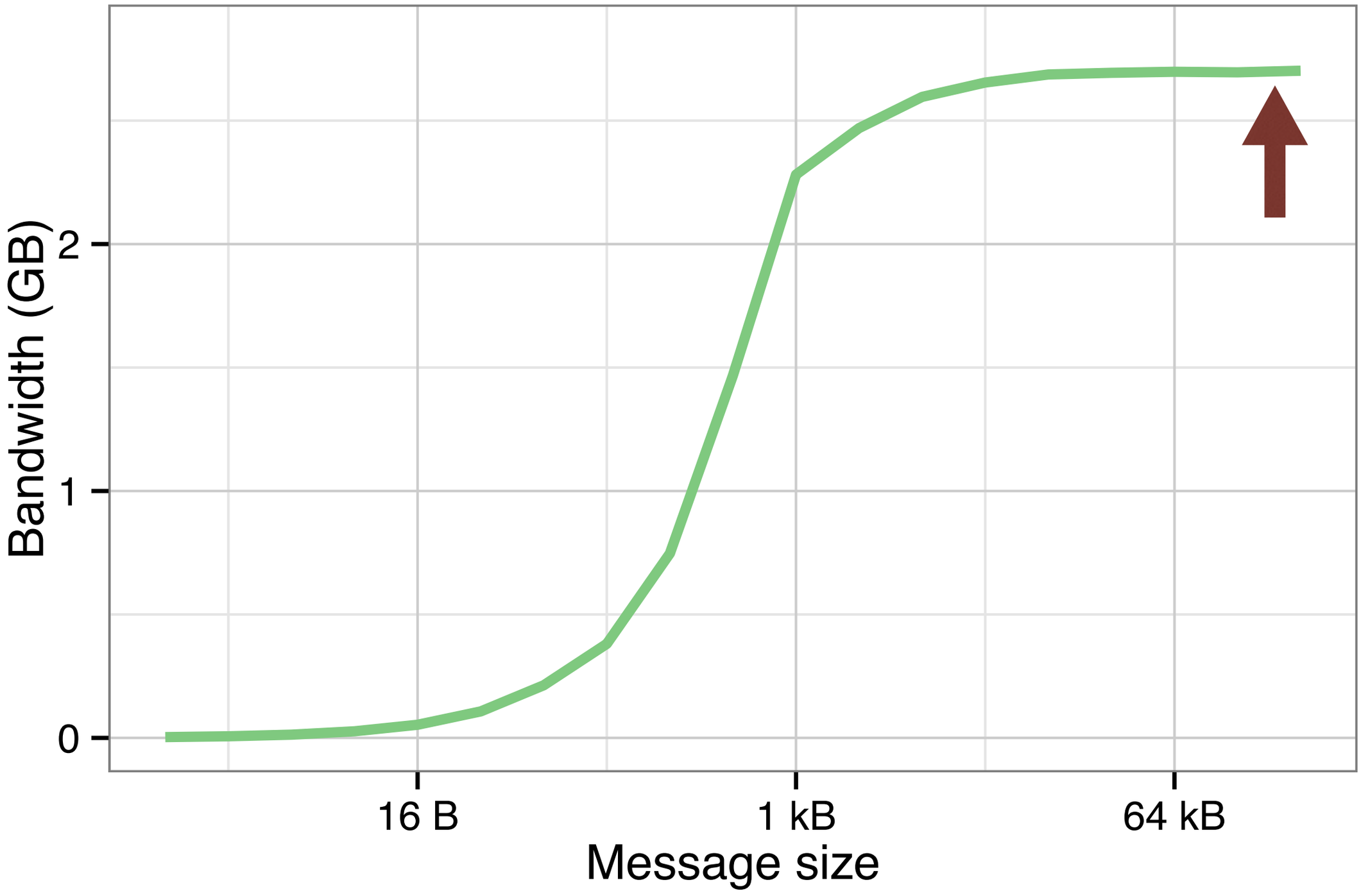
Small messages

Small tasks

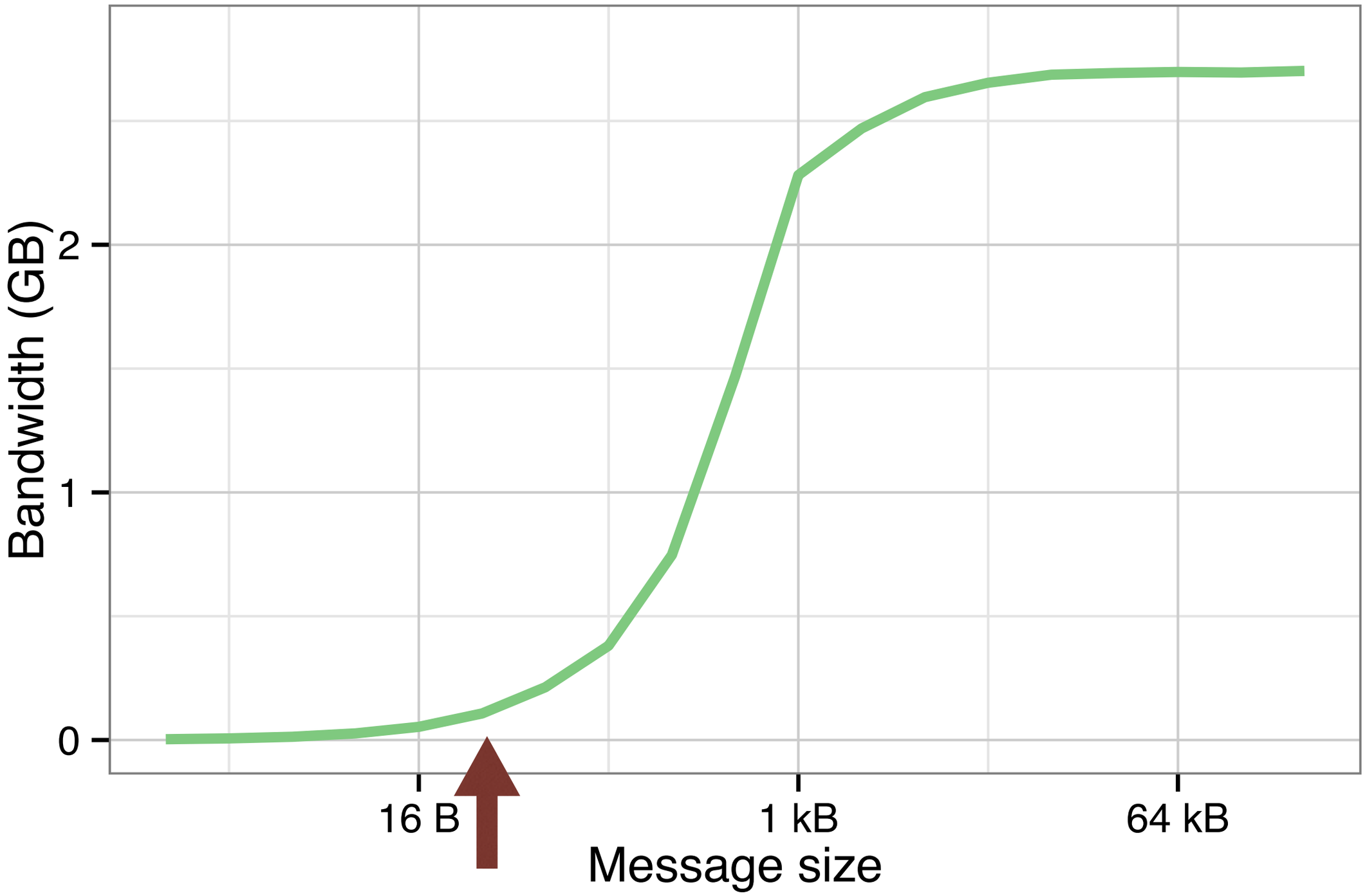
The small message problem



The small message problem

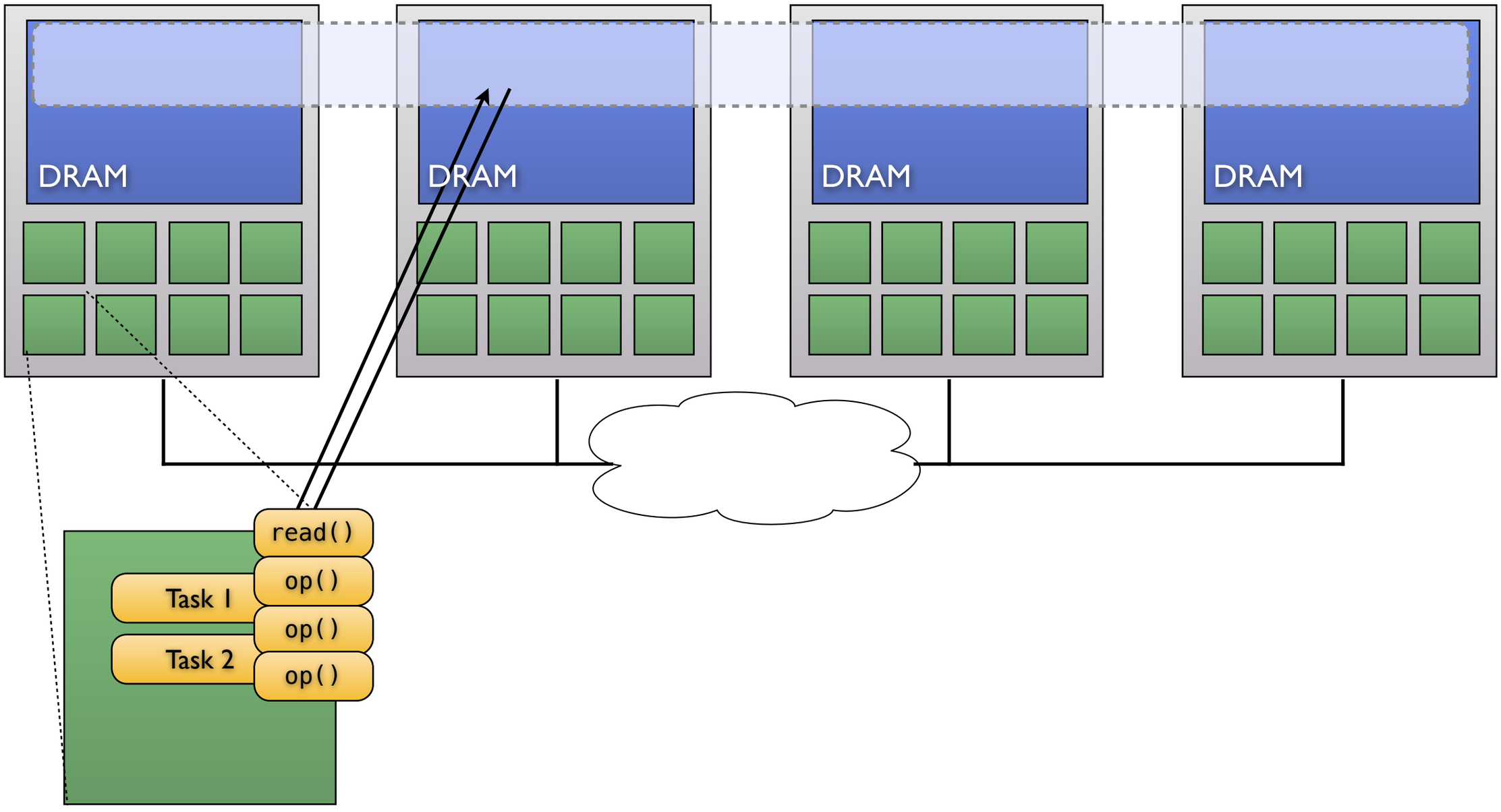


The small message problem

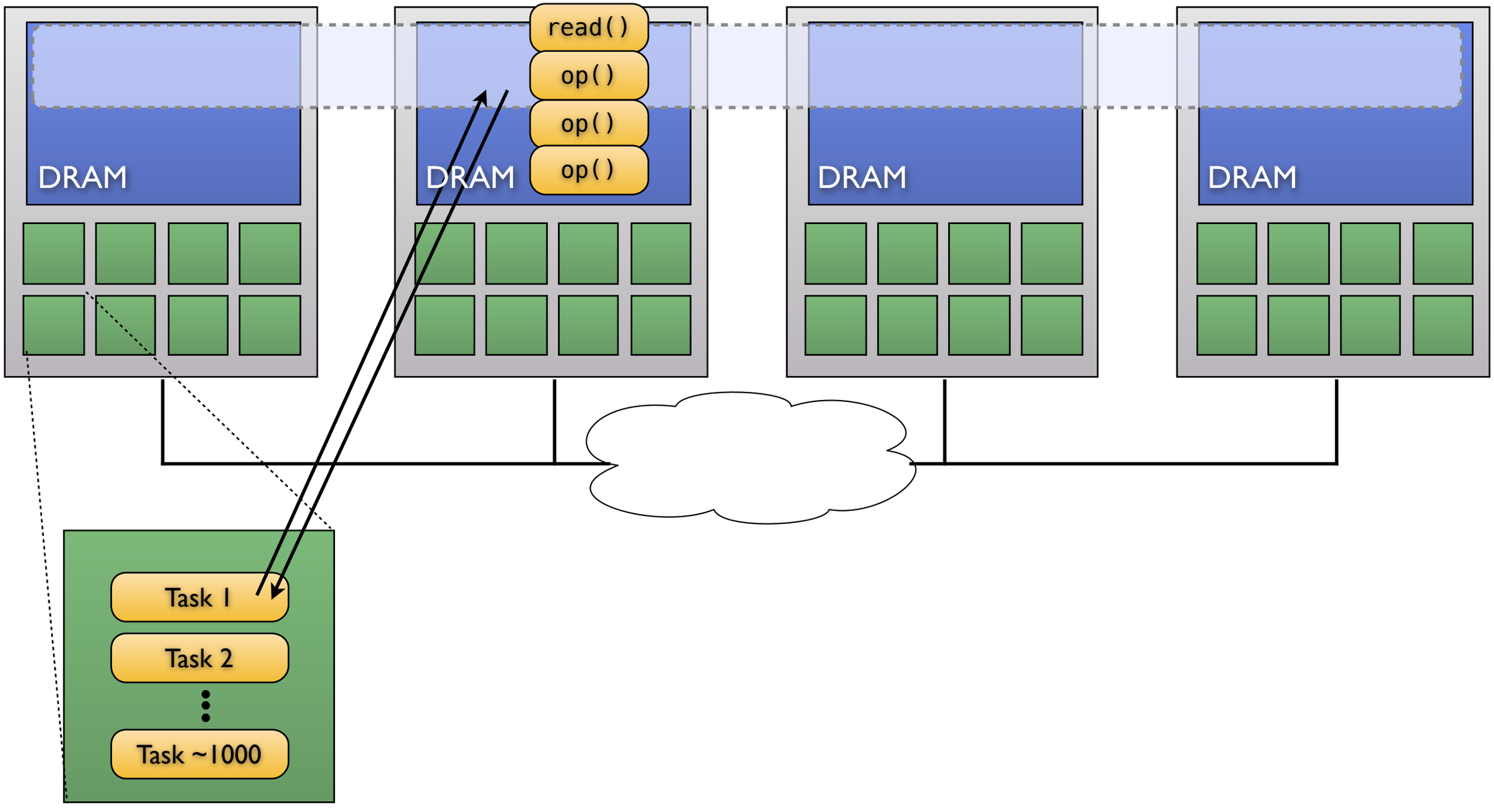


Our goal is throughput!
We can trade additional latency for increased throughput.

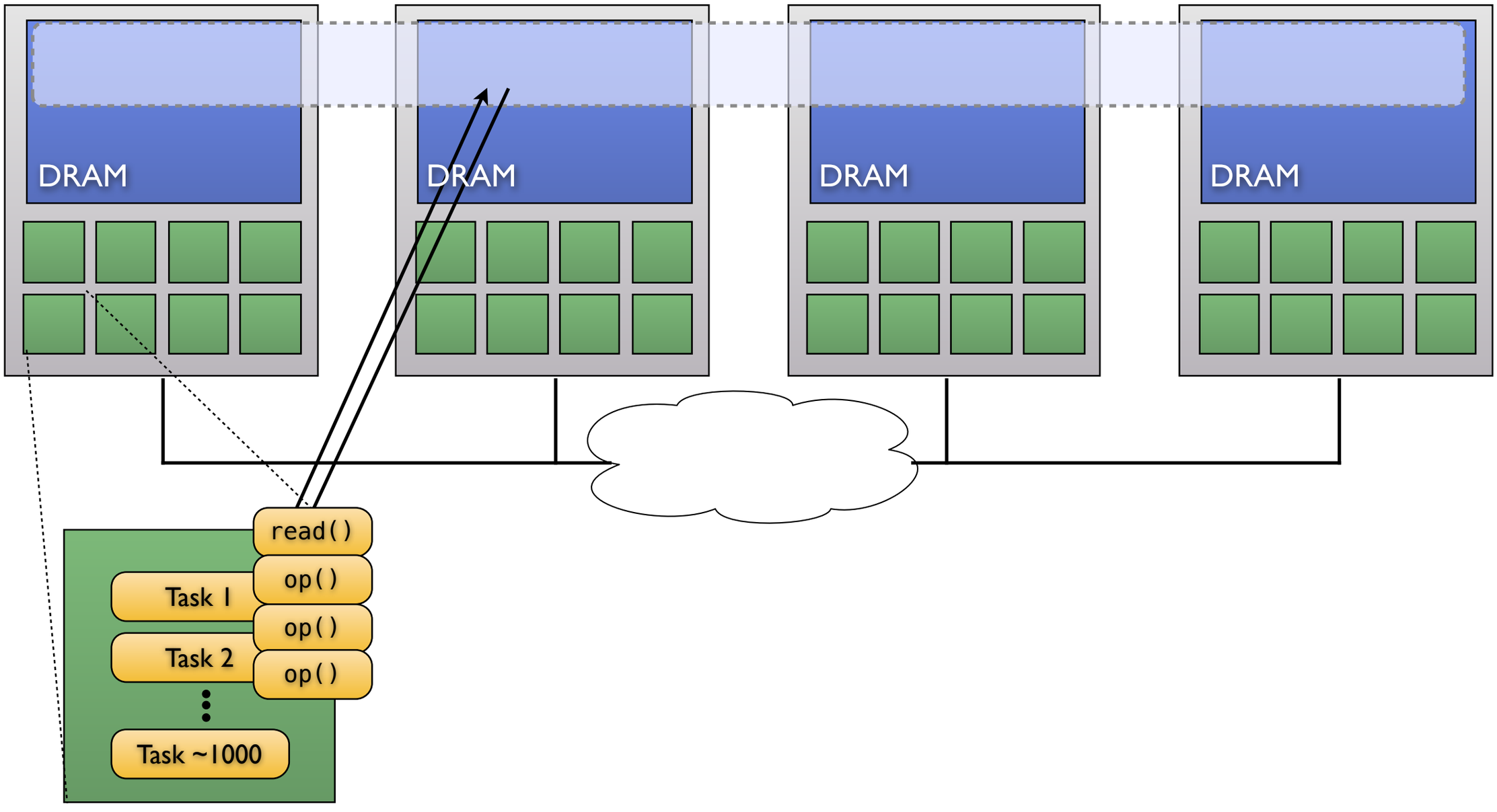
Aggregating remote operations



Aggregating remote operations

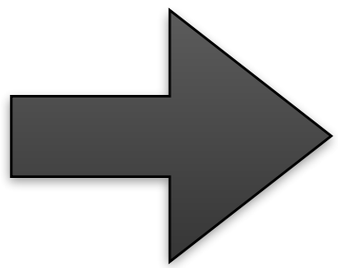


Aggregating remote operations



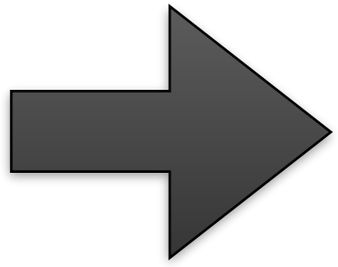
What makes this hard?

Lack of locality



Use parallelism to hide latency!

Small messages

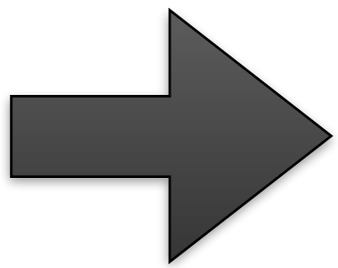


Trade latency for more throughput!

Small tasks

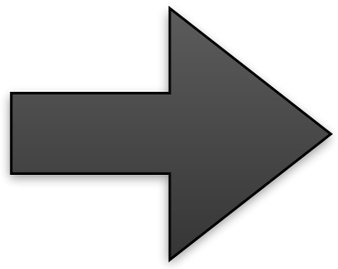
What makes this hard?

Lack of locality



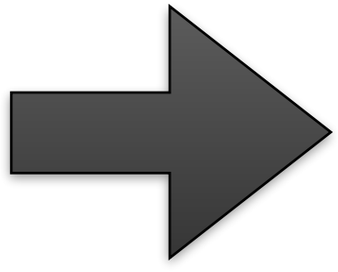
Use parallelism to hide latency!

Small messages



Trade latency for more throughput!

Small tasks



Make context switching fast.

User-level cooperative multithreading

- With ~1000 threads per core, contexts often don't fit in L1
  Our scheduler prefetches contexts into cache
  Limited by DRAM bandwidth, not miss latency

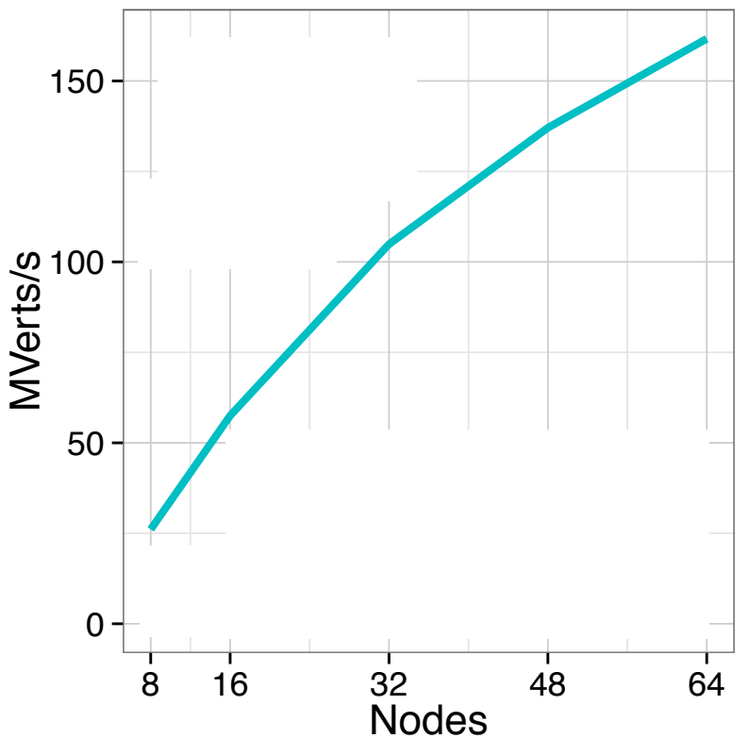
- Context switch moves 1 cacheline of thread state, 3 cachelines of working set

- ~50 ns

DSM implementation

Grappa Code: Expose DSM abstraction at language level using C++11 library

Searching a large, unbalanced tree



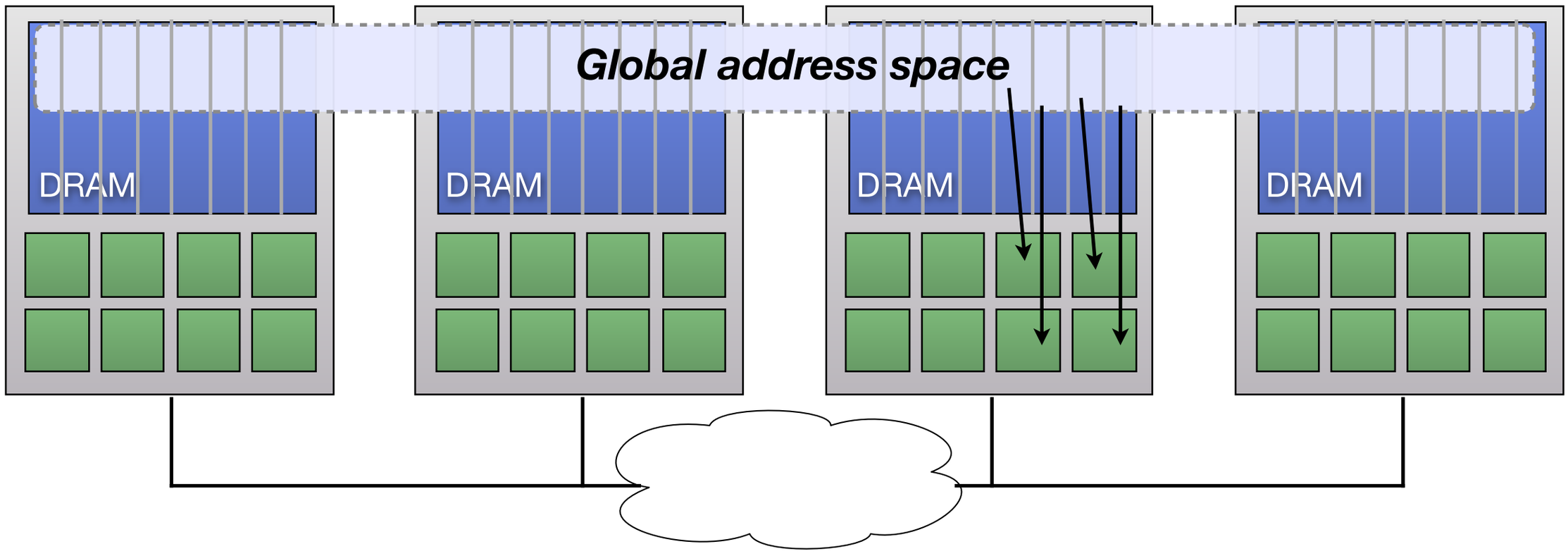
Standard single-core version

```
void search(Vertex * vertex_addr) {
    Vertex v = *vertex_addr;

    Vertex * child0 = v.children;
    for( int i = 0; i < v.num_children; ++i ) {
        search(child0+i);
    }
}
```

Grappa multi-node version

```
void search(GlobalAddress<Vertex> vertex_addr) {
    Vertex v = delegate::read(vertex_addr);

    GlobalAddress<Vertex> child0 = v.children;
    forall( 0, v.num_children, [child0](int64_t i) {
        search(child0+i);
    }
}
```

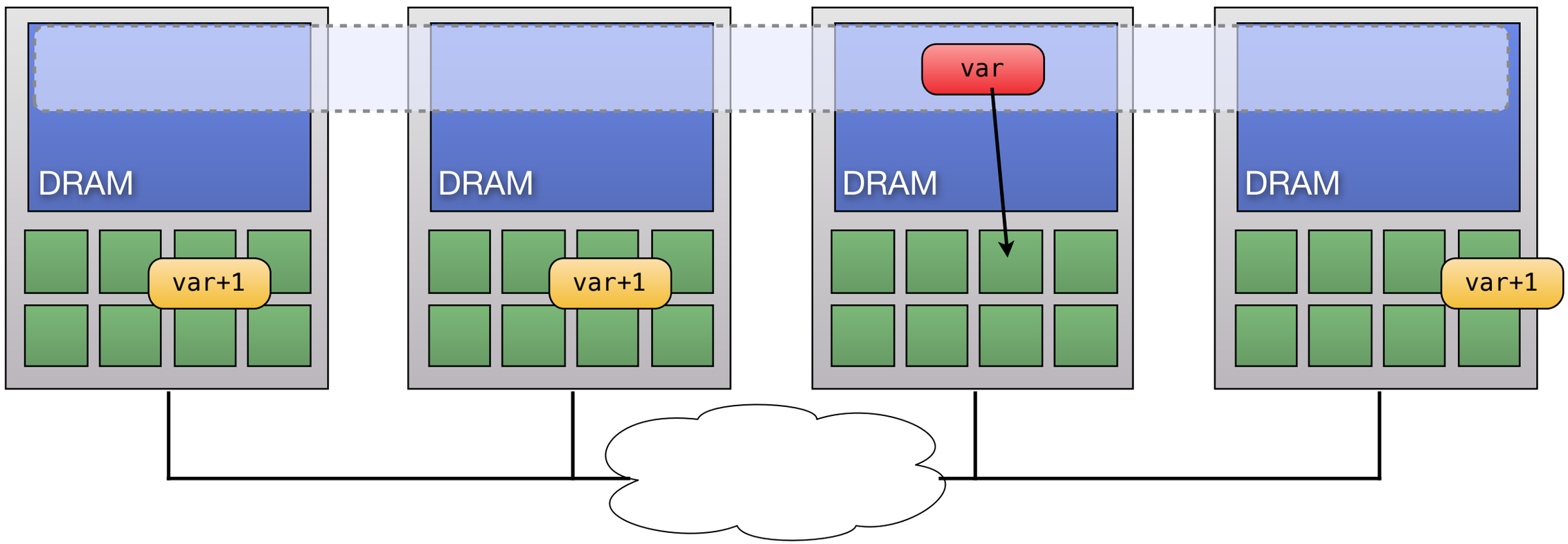
Accessing data in the global address space



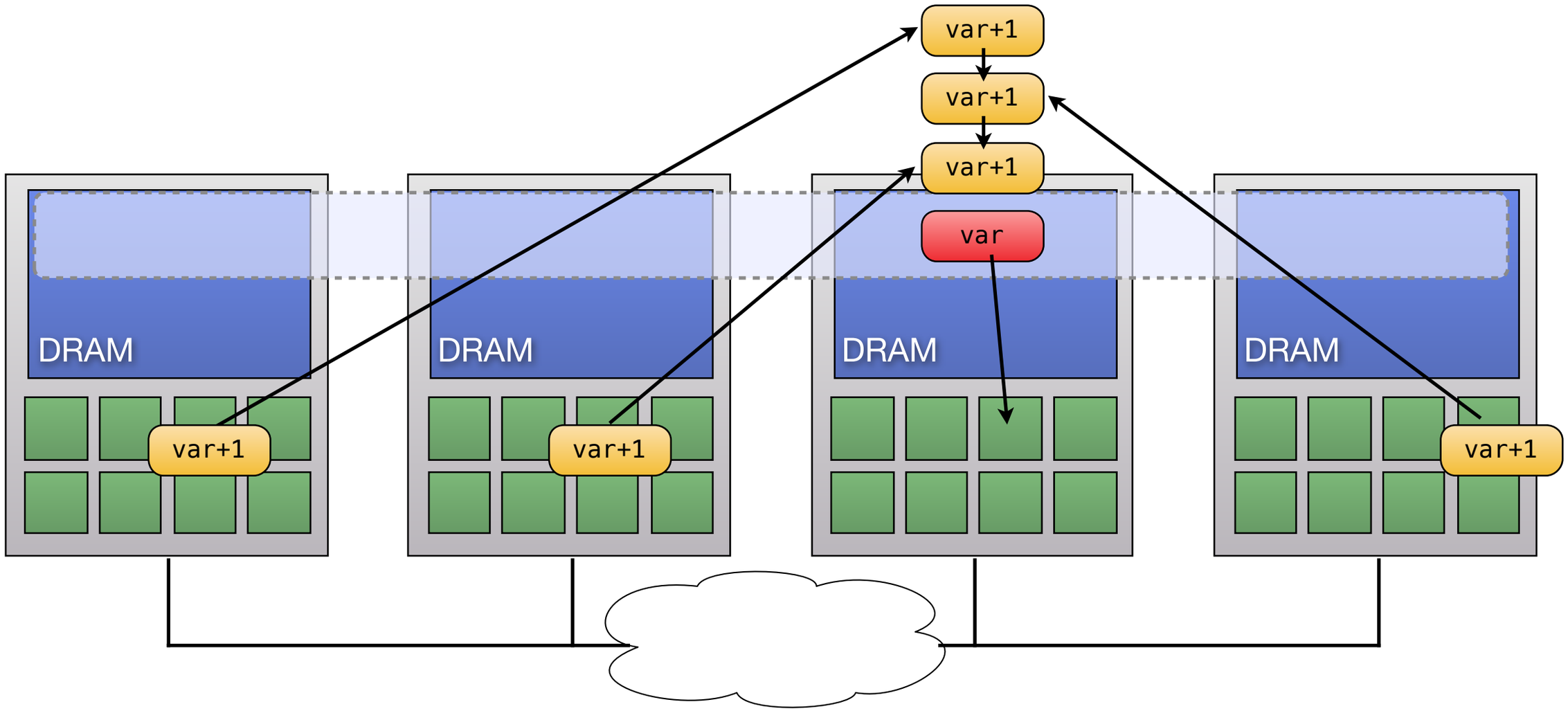
Memory is partitioned by core

All sharing is done using communication, so synchronization == scheduling

Accessing memory through delegates



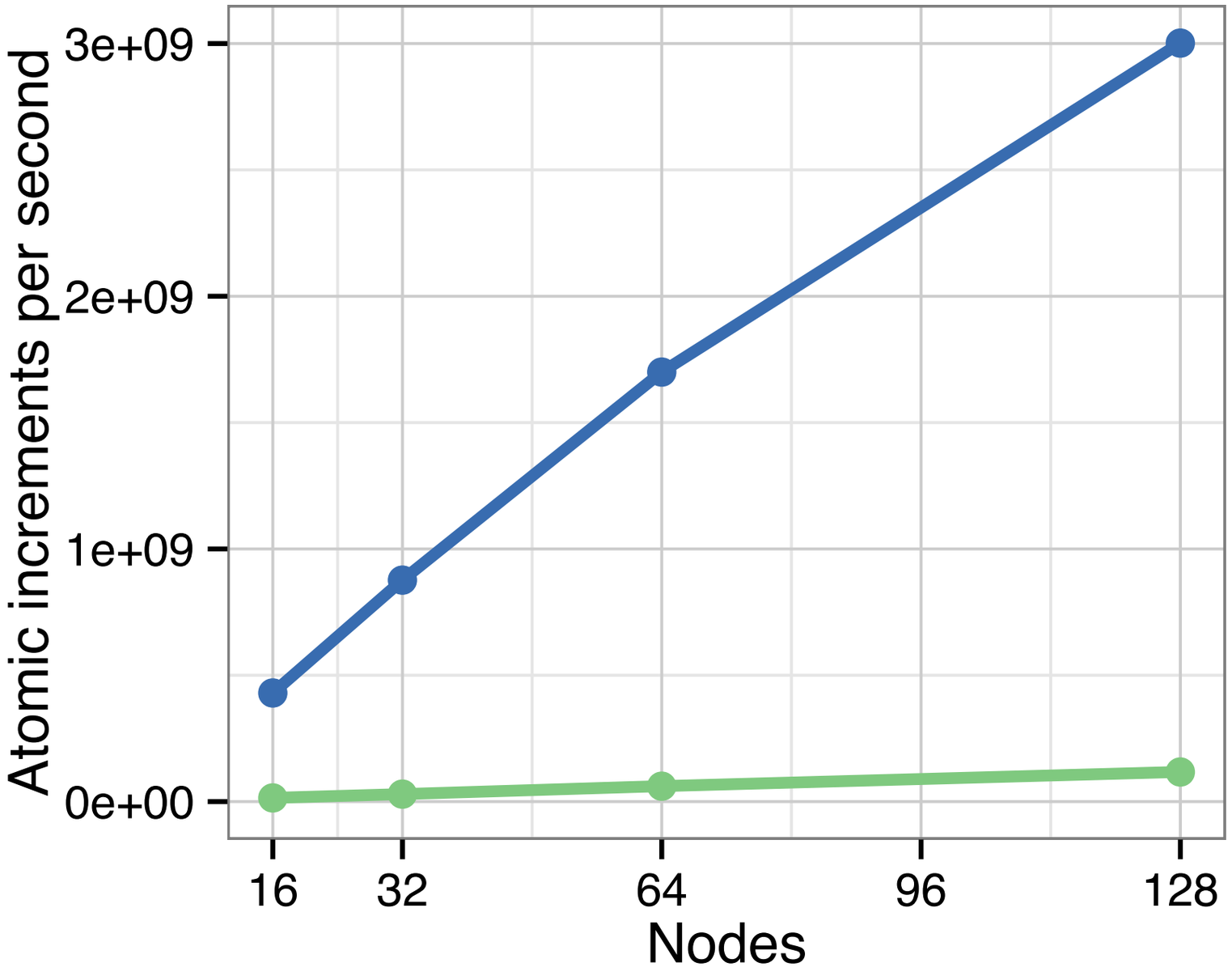
Accessing memory through delegates



Move computation to data:
All accesses to a word run on its home core

Results

Delegation + aggregation makes random access fast



GUPS pseudocode:

```
int a[BIG];
int b[n] = {rand()};

for (i=0; i<n; i++)
  a[ b[i] ]++;
```

32-core AMD Interlagos nodes,
Mellanox ConnectX-2
40 Gb InfiniBand

Building application frameworks on Grappa

```
void mapper(x) {
  k, v = compute(x)
  reducers[hash(k)].append(k,v)
}
void reducer(k, vals) {
  results.append(k,sum(vals))
}

forall (e : inputs)
  mapper(e)
forall ((k,vals) : reducers.groups)
  reducer(k, vals)
```

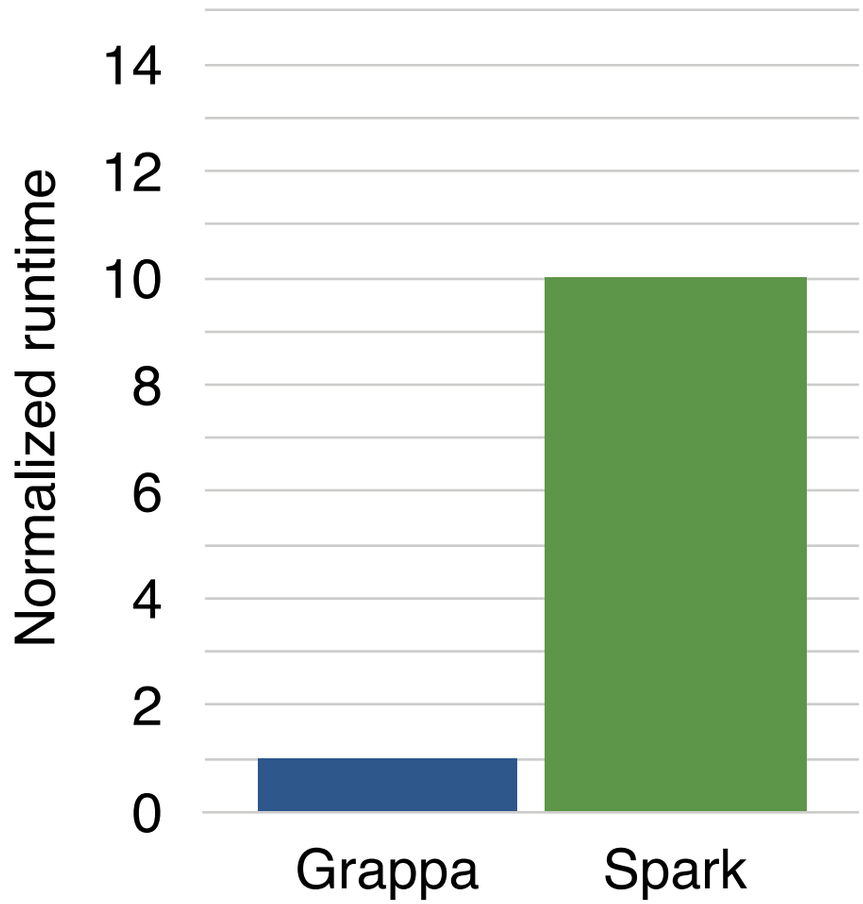
```
while (graph.active_verts.size > 0) {
  // gather phase
  forall (Vertex v : graph.active_verts)
    forall (Edge e : v.in_edges)
      v.prog.gather(v, e);
  // apply phase
  forall (Vertex v : graph.active_verts)
    v.prog.apply(v);
  // scatter phase
  forall (Vertex v : graph.active_verts)
    forall (Edge e : v.out_edges)
      v.prog.scatter(v, e);
}
```

```
// FriendsOfFollowers(a,b,c) :-
//     FollowedBy(a,b),
//     Friends(b,c),
//     a > 10
forall(Tuple t : Friends)
  hash0.insert(t.get(0), t);

forall(Tuple t : FollowedBy) {
  if (t.get(0) > 10) {
    e = hash0.lookup(t.get(1))
    results.append(e)
  }
}
```

In-memory MapReduce	GraphLab API	Relational query execution engine

In-memory MapReduce

- Simple implementation of MapReduce model for iterative applications (no fault-tolerance)
- Compared with Spark, with fault-tolerance disabled
- Benchmark: K-Means on SeaFlow ocean cytometry dataset (8.9GB)
- 64 AMD Interlagos nodes, Mellanox 40Gb ConnectX-2 InfiniBand

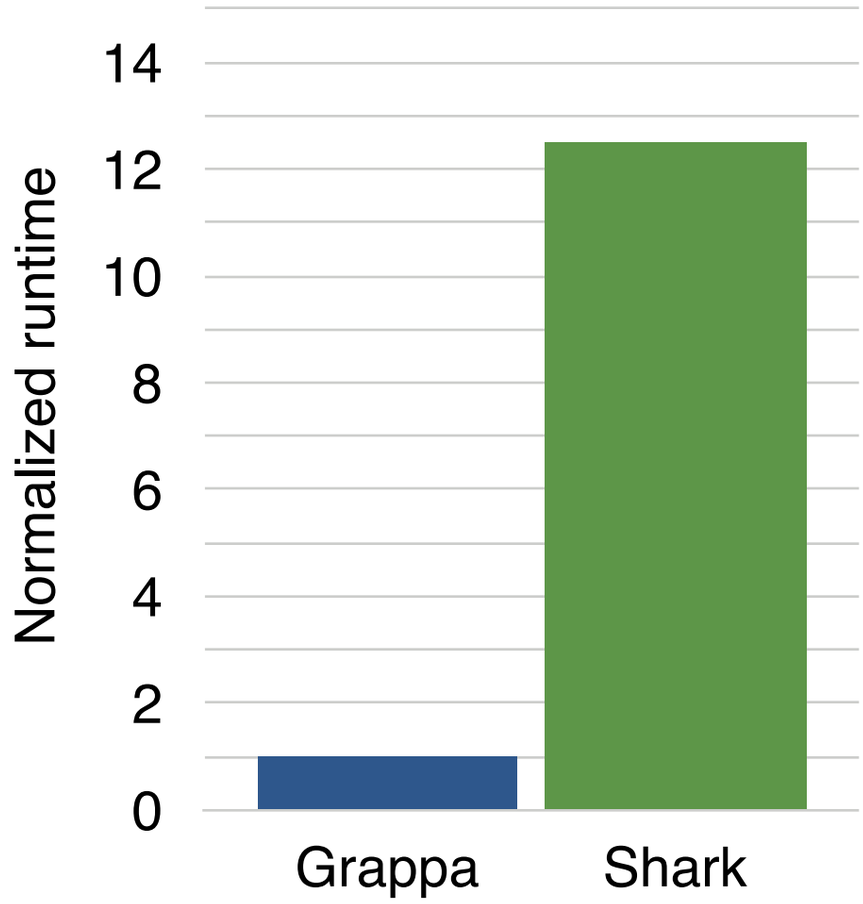


Relational query execution

- Built a backend for the Raco relational algebra compiler/optimizer: github.com/uwescience/raco

  - Queries are compiled into Grappa for() loops

- Compare with Shark, a Hive/SQL-like query system built on Spark using SP2Bench benchmark



(16 nodes)

GraphLab on Grappa

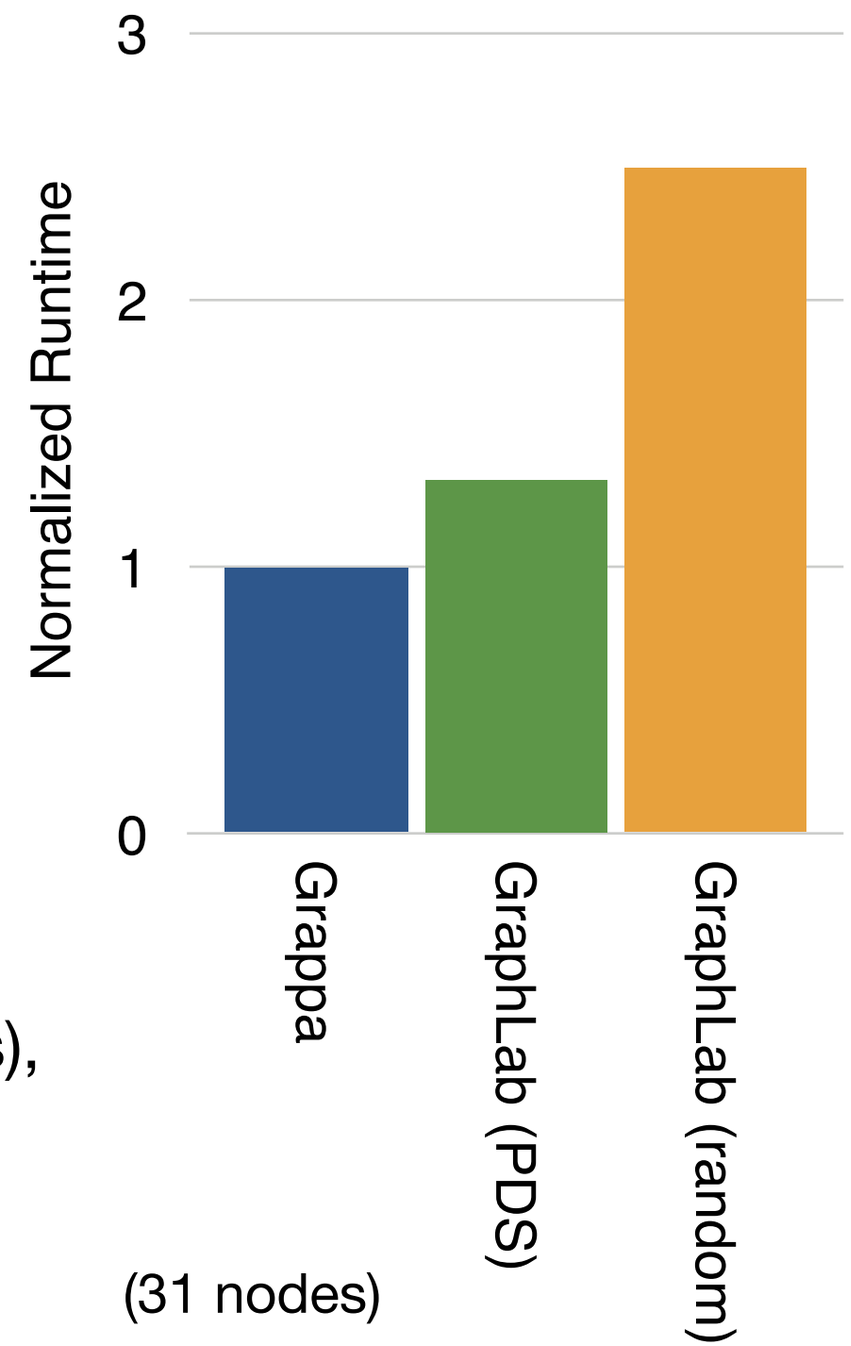
- Subset of the GraphLab API described in PowerGraph paper

- GraphLab: replicated graph representation, complex partitioning strategy;

  Grappa: simple adjacency list, random partitioning

- Four benchmarks from GraphBench.org:
  PageRank, conn. components, SSSP, BFS

- Graphs: Friendster (65M vertices, 1.8B edges), Twitter (41M vertices, 1B edges)



Why is Grappa fast?

- Much higher message rates

- Built to enable use of RDMA
  (but is still fast over TCP)

- Faster serialization

- Efficient fine-grained synchronization and scheduling

Not in the talk

- Also in paper:
  Deeper dive on performance
  Results from programming against Grappa directly

- Related projects:

Alembic: Automatic Locality Extraction via Migration.
B. Holt, P. Briggs, L. Ceze, M. Oskin
OOPSLA 2014

Radish: Compiling Efficient Query Plans for Distributed Shared Memory.
B. Myers, D. Halperin, J. Nelson, M. Oskin, L. Ceze, B. Howe
Tech report, October 2014

Flat Combining Synchronized Global Data Structures.
B. Holt, J. Nelson, B. Myers, P. Briggs, L. Ceze, S. Kahan, and M. Oskin
International Conference on PGAS Programming Models (PGAS), October 2013

Conclusion

- Grappa is a platform for building new data-intensive analytics frameworks
- **Latency tolerance** enables fast distributed shared memory for analytics
- BSD-licensed source, more info:

http://grappa.io