



| IBM Research

Active collectives: Extending the reach of APGAS collectives

George Almási

Outline

- **The IBM APGAS library**
- **Collective communication in the APGAS library**
- **Limits of current approach**
- **Active collectives, ad-hoc teams**
- **Conclusion**

The APGAS library in a nutshell

- **Common runtime for UPC, CAF, X10**
 - Deployed since March '09
- **Multithreaded/Multinode/Heterogeneous Initialize/Finalize**
 - Provides “place” abstraction to higher levels
- **Transport for LAPI, DCMF, Myrinet, TCP/IP sockets**
 - Remote GET, PUT, AMSend – one-sided, async. Interface
- **Support for heterogeneous architectures**
 - Cell SPUs, NVidia GPUs
- **Collective communication subsystem**
- **UPC, CAF, X10 subsystem specific code**
 - UPC shared arrays, locks; X10 asyncs etc.

Collective communication primitives

- **Efficient means to schedule large scale data movement**
 - Ease of use
 - Performance portability
 - STL vs “MyQueue”

Broadcast
Scatter
Gather
Allgather
Allreduce
Alltoall

- **Barriers to adoption:**
 - Lack of good implementations
 - That is finally changing
 - Perceived lack of need
 - Small machines
 - Benefits leadership class machines more than ordinary NOW
 - But large SMPs also benefit from collectives!

The APGAS collective library today

■ The usual suspects

- broadcast, scatter/gather, reductions, alltoall (*)
- Tweaked: hybrid implementation (clusters of SMPs)
- Blocking calls today, working on non-blocking

■ Communicators/teams

- “split” primitive to create subcommunicators

■ Working with product groups

- APGAS != MPI
- Reuse efforts of Blue Gene, PERCS development teams rather than write own

Collective communication as a means to achieve productivity and performance in the APGAS library

[PERCS Milestone 7]

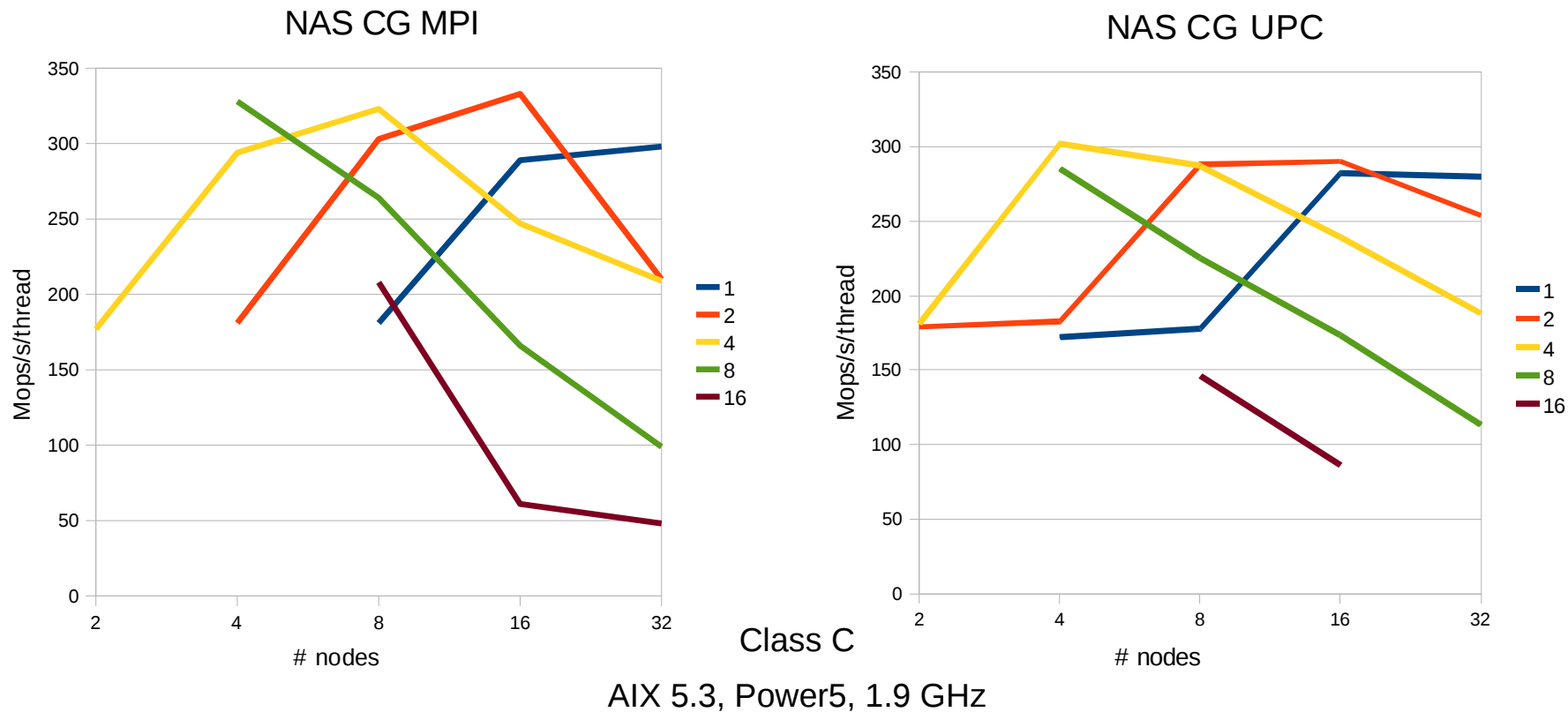
■ NAS CG benchmark:

- MPI: 1825 lines of code
 - 900 lines of communication code
 - Dense, tightly embedded
 - Re-writes of:
 - > Allreduce
 - > Alltoall (matrix transpose)
- UPC+APGAS: 935 lines
 - Performance almost equivalent

■ [HPCC 08] Linpack

- MPI: 30,000 lines of code
 - 3,000 lines of communication code
 - Broadcast
 - Allreduce
- UPC+APGAS: 1430 lines
- X10+APGAS: 1391 lines
 - Performance:
 - Power cluster: trail by 10%
 - Blue Gene/L: trail by 30%

NAS 3.2 CG - MPI vs. UPC

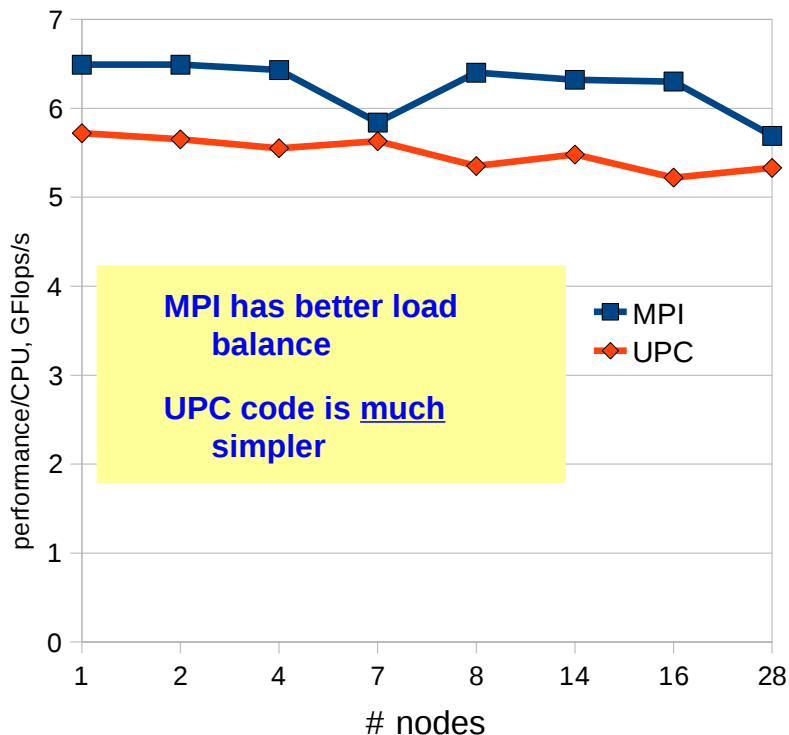


Performance: almost equivalent, UPC within 10% of MPI
 Complexity: UPC code **935** lines, MPI code **1825** lines

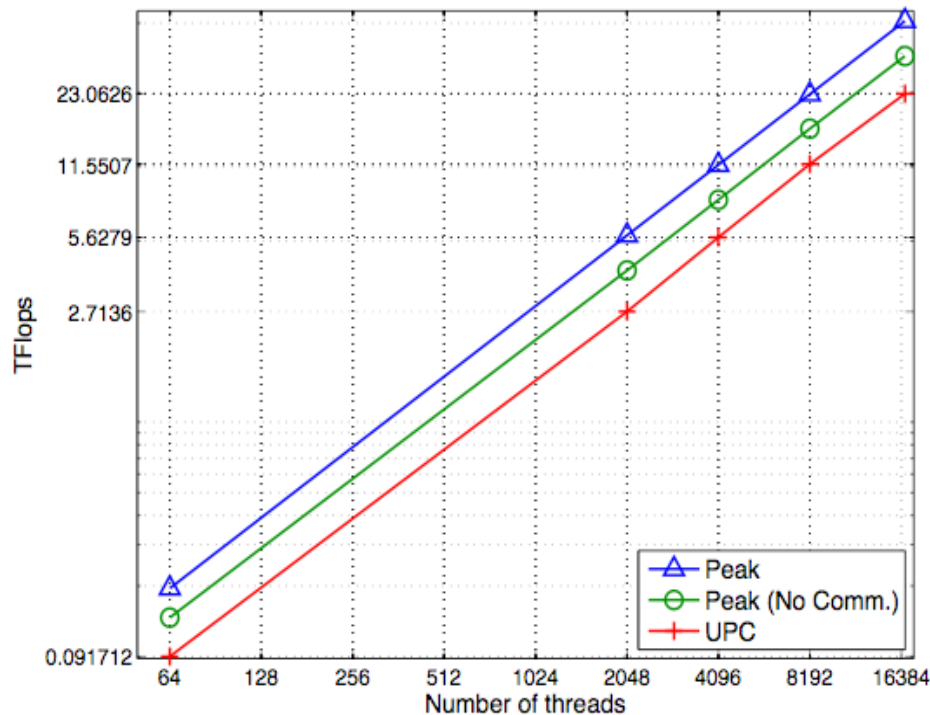
HPCC Linpack – MPI vs UPC

MPI vs UPC HPL

Power5 + HPS, 28 nodes, 16 processes/node



UPC HPL (Blue Gene/L, 16k nodes)



Performance: almost equivalent, lags 10% behind MPI
Complexity: UPC code 1,430 lines, MPI code 30,744 lines

APGAS Collectives and UPC

- **UPC standard 1.2 includes collective communication**
 - Provided by APGAS collectives library
- **Current standard is inadequate**
 - Collectives allowed only on shared arrays
 - Extension proposal in the works: value based semantics
 - Unnecessarily complicated API
 - Tied to availability of remote shared data (*)
 - No support for teams
 - Berkeley extension: team collectives (*)
- **In process: implementation & evaluation of extended standard**

Non-blocking collectives

■ MPI 2 standard:

blocking collectives

- Collective returns when complete

■ Difficult to program for:

- Comm./comp. overlap
- Comm./comm. overlap

■ MPI 3 forum:

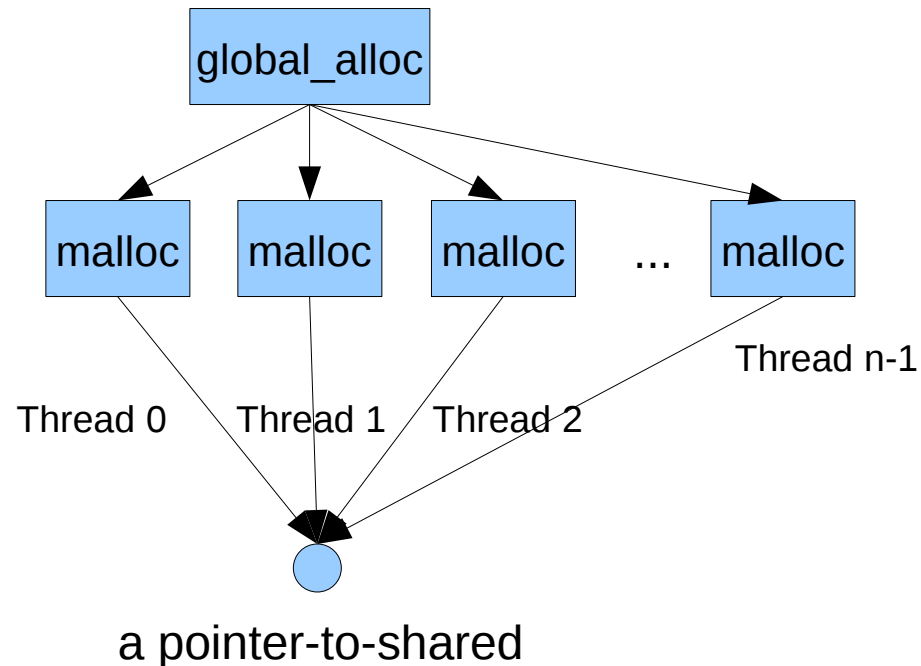
non-blocking collectives

■ IBM Research & product groups involved with MPI forum activities

- Will provide compliant implementation
- APGAS runtime will benefit

The upc_global_alloc() lesson

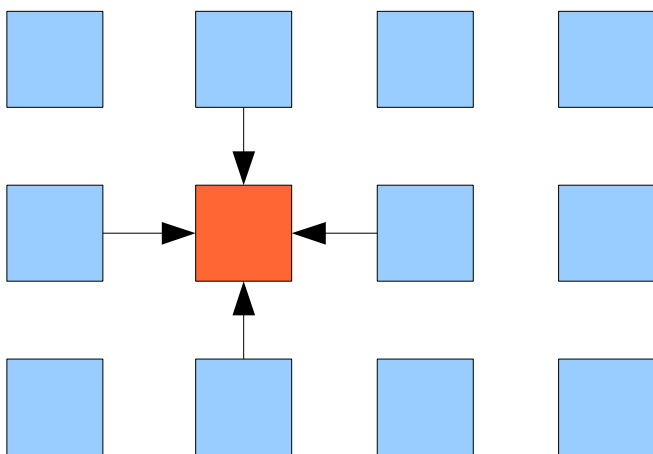
- Allocate “blks” times “nbytes” space across whole machine
- Results in a pointer-to-shared valid anywhere on the system
 - Invoked on 1 thread
 - Result on the same thread
 - But all other threads required to contribute
 - One-sided scatter/gather



Chaotic relaxation, “Monte Pi” & friends

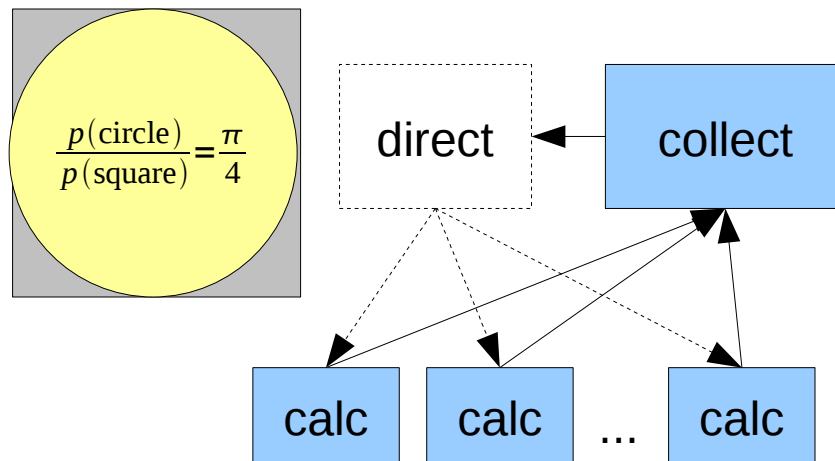
■ Chaotic stencil code:

- Gather neighbor data; timing unimportant
- Can fix load balance problems
- Scatter or gather



■ Chaotic Monte-Carlo, simulated annealing

- Periodically gather partial results
- Coalesced result determines convergence

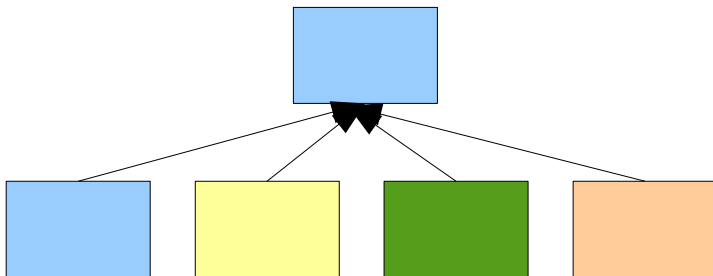


Distributed tree traversal

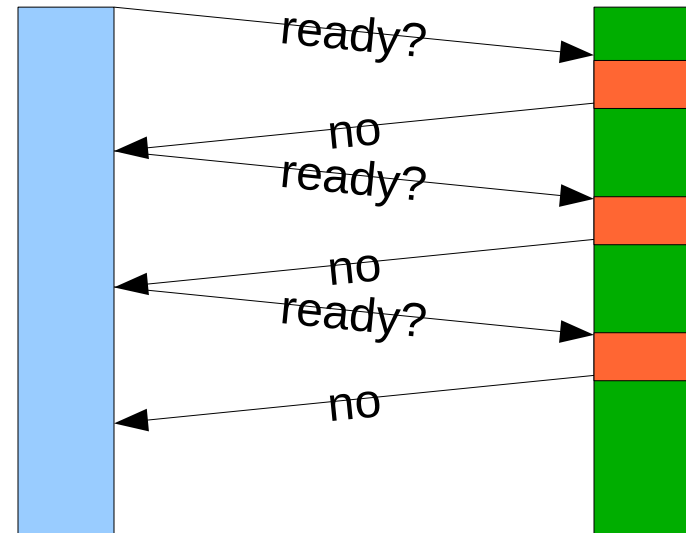
■ Barnes-Hut: C.G. calc.

– Distributed Octree traversal

- Wait for all subtrees to be ready before computing node CG
- Busy-waiting not scalable
 - Livelock!

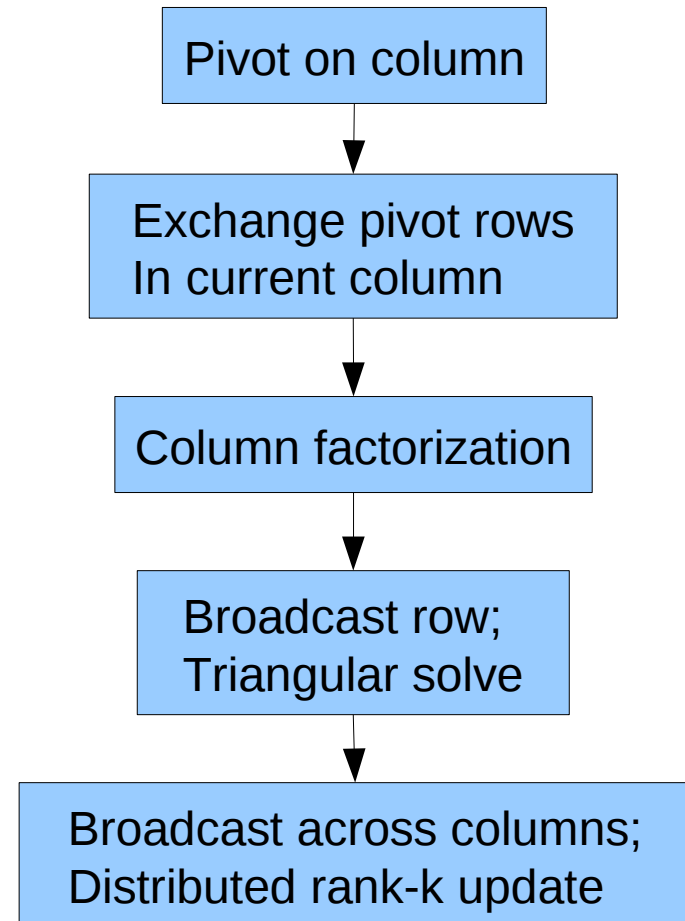


```
void comp_cg (node) {
  node->Cg = 0
  for (i=0; i<node->nchld; i++) {
    child = node->children[i];
    while (!child->ready) ;
    node->cg += child->cg;
  }
  node->cg /= node->children;
  node->ready = true;
}
```

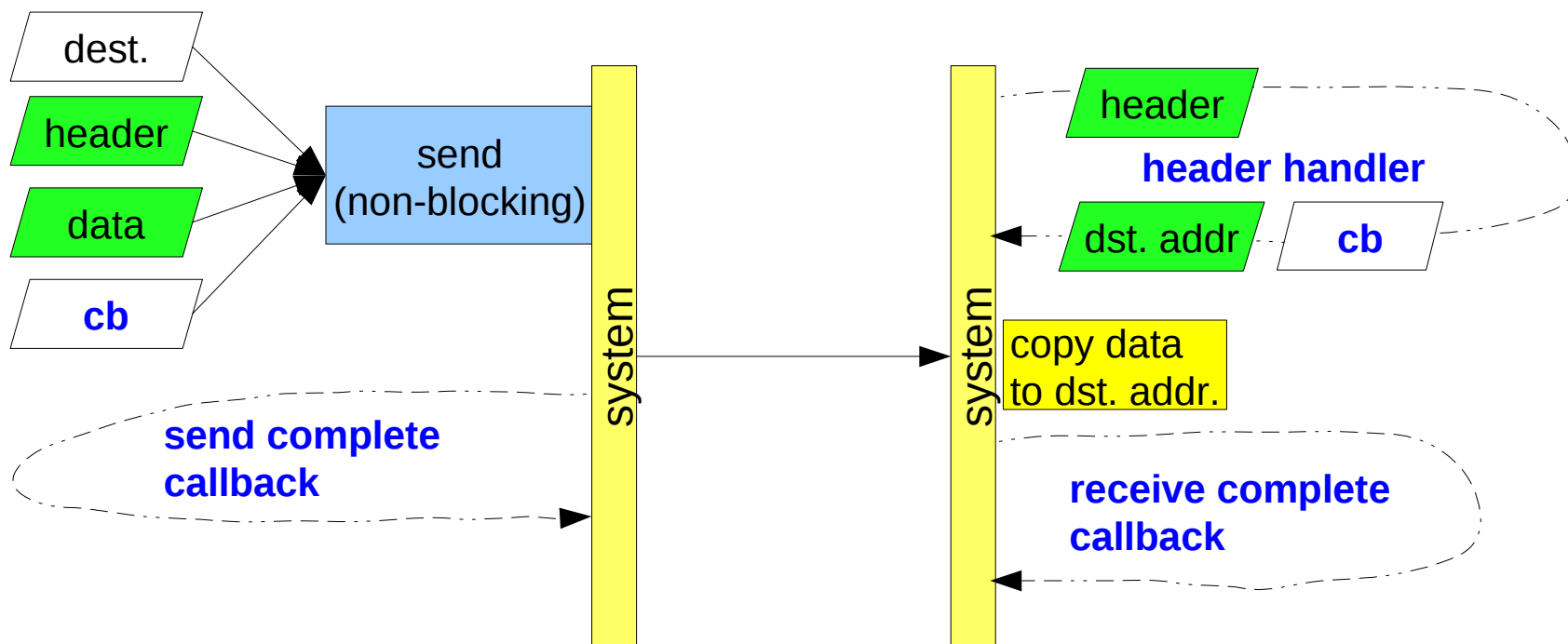


Data flow HPL

- **HPC data dependency chain across groups of tasks**
 - Data dependent formulation eases load imbalance
- **Simple idea, nasty code**
 - SPMD cannot express
 - Async formulation: no collectives
- **What is needed:**
 - One-sided broadcast + active message sent to receivers
 - One-sided reduction with passive notification



Review: the structure of a LAPI active message



- **Send complete:** as soon as send buffers reusable
- **Header handler:** when data arrives
- **Recv complete:** when data transfer complete
- **How does sender know reception is complete? (fences!)**

Proposal: active collectives in the APGAS library

■ Natural extension of active messages

- Similar (familiar) API
 - callbacks
- Same completion semantics
- Addressee:
 - explicit list
 - communicator

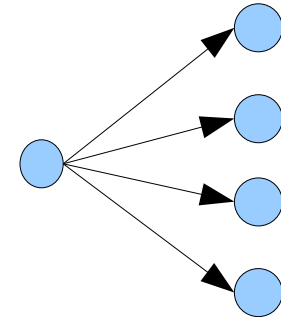
■ Take w/ 2 lumps of syntax sugar!

- People don't like to program callbacks, but:
 - Earlier proposals for UPC one-sided collectives
 - Chaotic codes
 - X10 clocks
 - Java concurrency utils
 - Register/notify patterns
 - Syntactic variations of non-blocking collectives

Active collectives: syntax

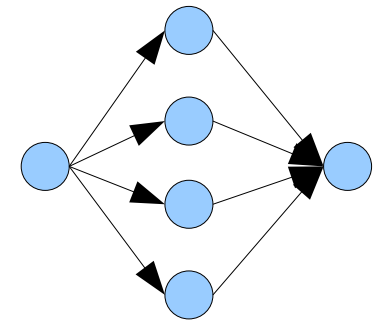
“Multi-send” initiated by 1 place

```
am_bcast (dest, hdr, data, cb)
am_scatter (dest, hdr, data, cb)
```



Data gather operation initiated by 1 place;
Has scatter/gather functionality

```
am_gather (dest, hdr, data, cb)
```



One-sided reduction:

Like am_gather, but data processing instead of relocation

```
am_reduce (dest, hdr, data, cb, op, dtype)
```

Data flow graphs are conceptual;
implementations optimize latency, bandwidth, fan-in, fan-out

Implementation issues

- **There is no better time to push active collectives**
 - MPI 3 Forum specification for non-blocking collectives
 - Like non-blocking point-to-point MPI messages
 - Overlap computation/communication
 - Internal implementation of non-blocking collectives exposes state-machine based implementations
 - Can be trivially adapted to execute active collectives
- **Jury is out on ad-hoc teams/communicators**
 - Implementations today:
 - Rely on heavy-weight communicator creation primitives
 - Have limits on # communicators, memory use
 - Major rethinking needed

Conclusions

- **Collectives useful in APGAS context**
- **Current collectives implementation SPMD limited**
- **One-sided “active” collectives needed**
 - Will help scale non-SPMD PGAS codes
 - Clear implementation path: ride on MPI Forum non-blocking collectives initiative
- **Ad-hoc teams/communicatore needed**
 - Implementation dicey, requires thought