Do You Know What Your I/O Is Doing?

William Gropp
www.cs.illinois.edu/~wgropp

Messages

- Current I/O performance is poor
  - Even relative to what current systems can achieve
  - Part of the problem is the I/O interface semantics
- Big data is more than just I/O
  - HPC has relevant insights

Just How Bad Is Current I/O Performance?

- Much of the data (and some slides) taken from “A Multiplatform Study of I/O Behavior on Petascale Supercomputers,” Huong Luu, Marianne Winslett, William Gropp, Robert Ross, Philip Carns, Kevin Harms, Prabhat, Suren Byna, and Yushu Yao, presented at HPDC’15.
  - This paper has lots more data – consider this presentation a sampling
- Thanks to Luu, Behzad, and the Blue Waters staff and project for Blue Waters results
  - Analysis part of PAID program at Blue Waters

I/O Logs Captured By Darshan, A Lightweight I/O Characterization Tool

- Instruments I/O functions at multiple levels
- Reports key I/O characteristics
- Does not capture text I/O functions
- Low overhead → Automatically deployed on multiple platforms.

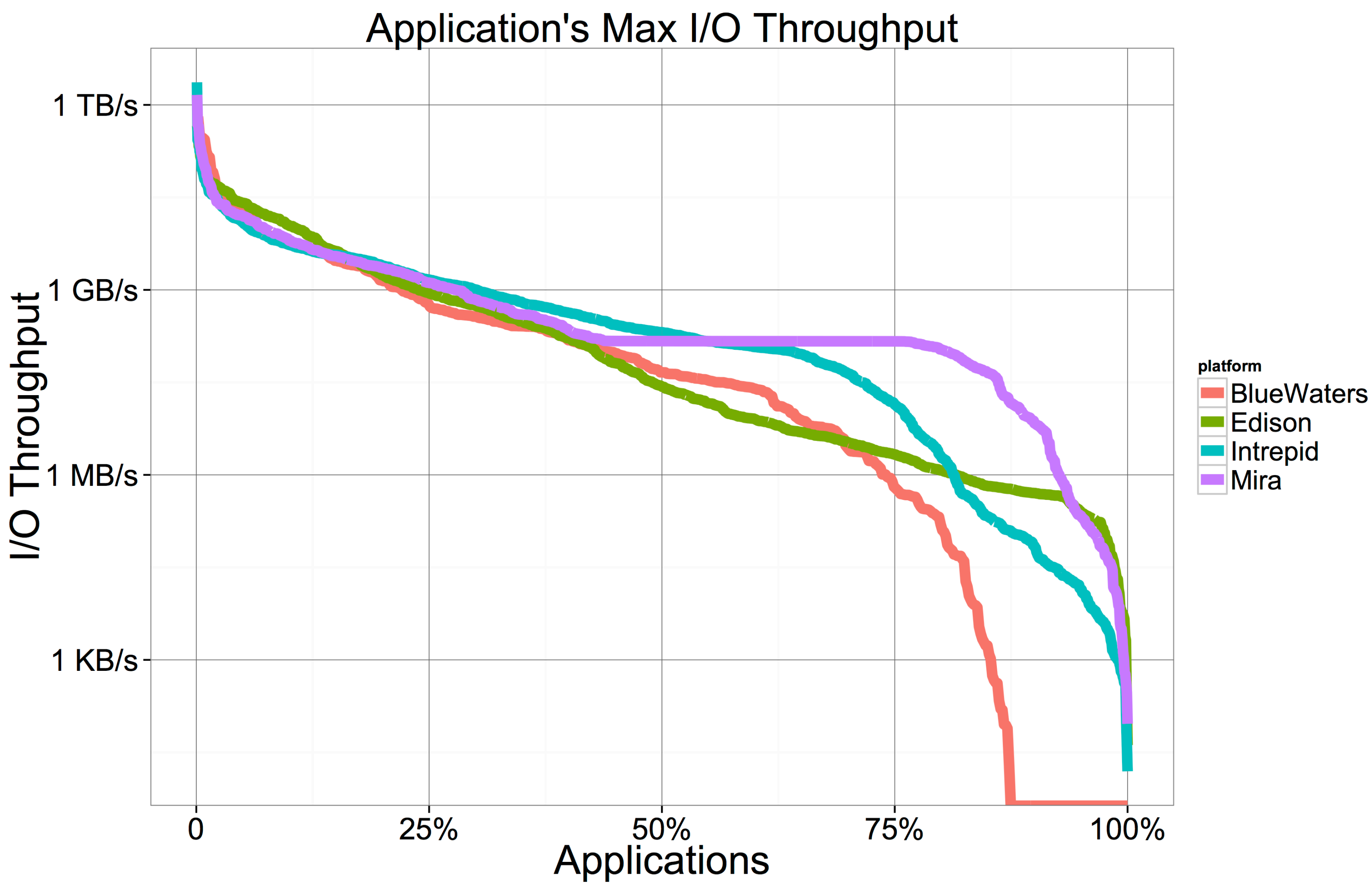
Caveats on Darshan Data

- Users can opt out
  - Not all applications recorded; typically about ½ on DOE systems
- Data saved at MPI_Finalize
  - Applications that don't call MPI_Finalize, e.g., run until time is expired and then restart from the last checkpoint, aren't covered
- About ½ of Blue Waters Darshan data not included in analysis
  - Expect to be fixed soon

I/O log dataset: 4 platforms, >1M jobs, almost 7 years combined

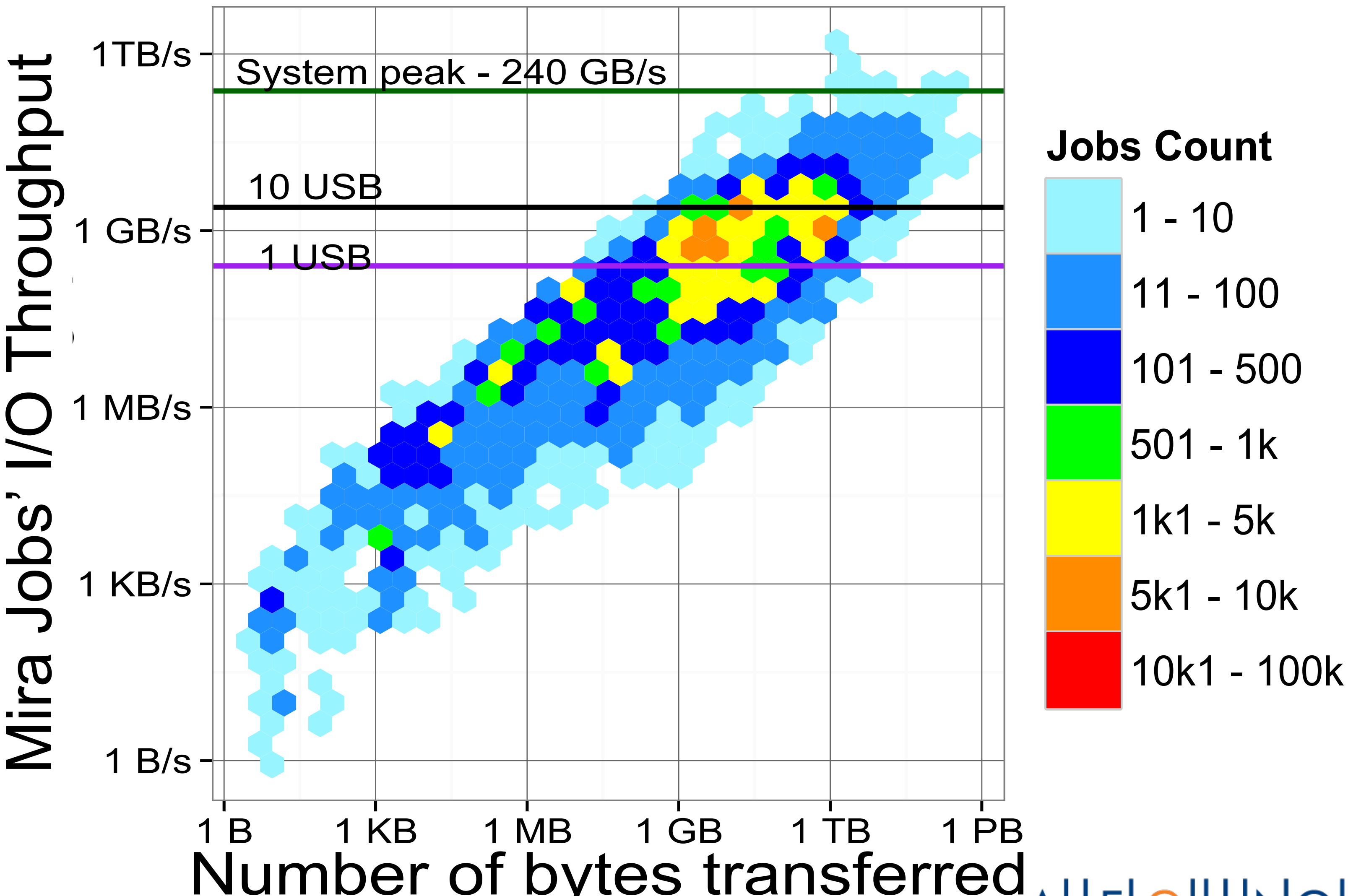
	Intrepid	Mira	Edison	Blue Waters
Architecture	BG/P	BG/Q	Cray XC30	Cray XE6/ XK7
Peak Flops	0.557 PF	10 PF	2.57 PF	13.34 PF
Cores	160K	768K	130K	792K+59K smx
Total Storage	6 PB	24 PB	7.56 PB	26.4 PB
Peak I/O Throughput	88 GB/s	240 GB/s	168 GB/s	963 GB/s
File System	GPFS	GPFS	Lustre	Lustre
# of jobs	**239K**	**137K**	**703K**	**300K**
Time period	**4 years**	**18 months**	**9 months**	**6 months**

Very Low I/O Throughput Is The Norm

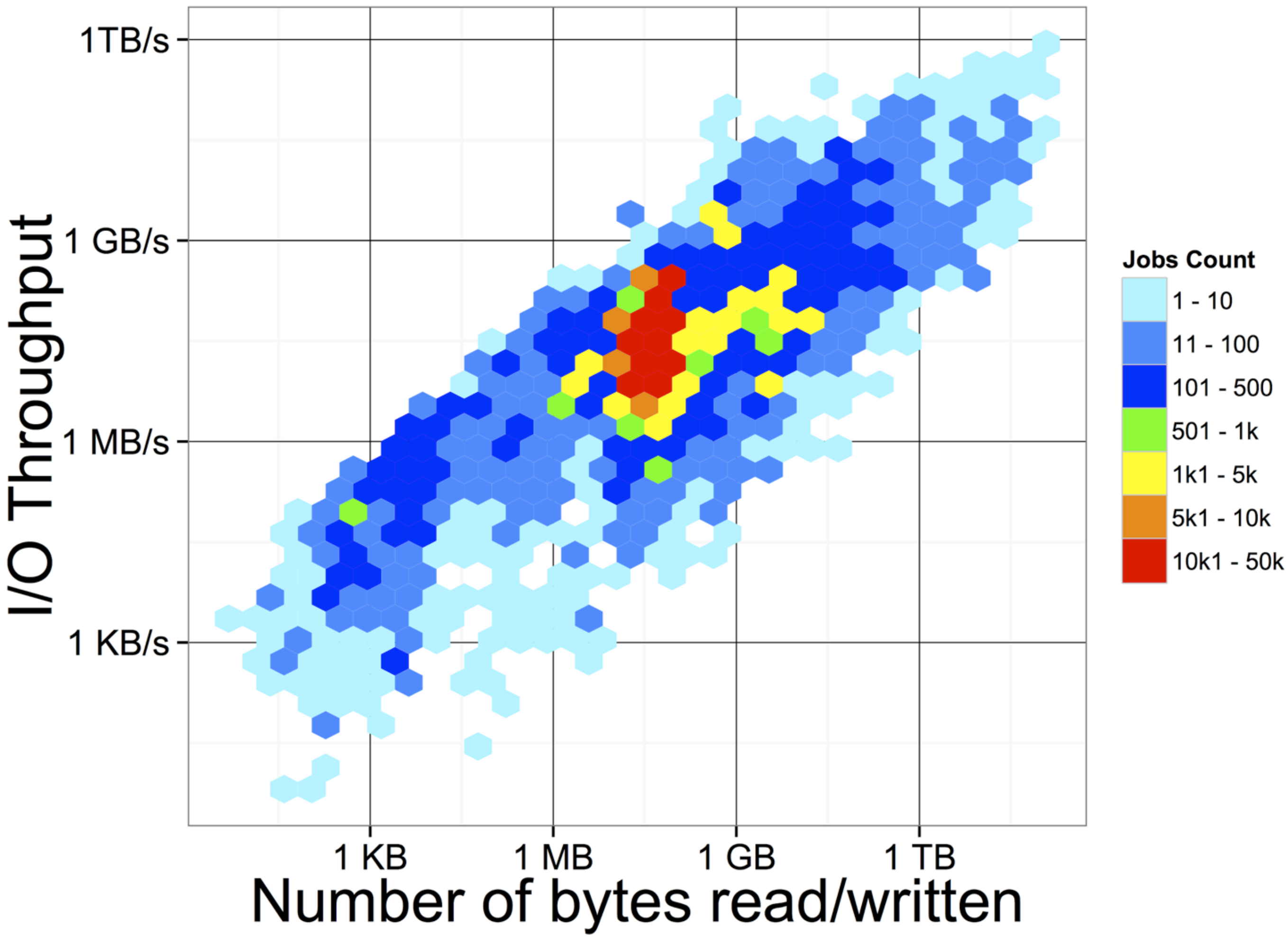


PARALLEL@ILLINOIS

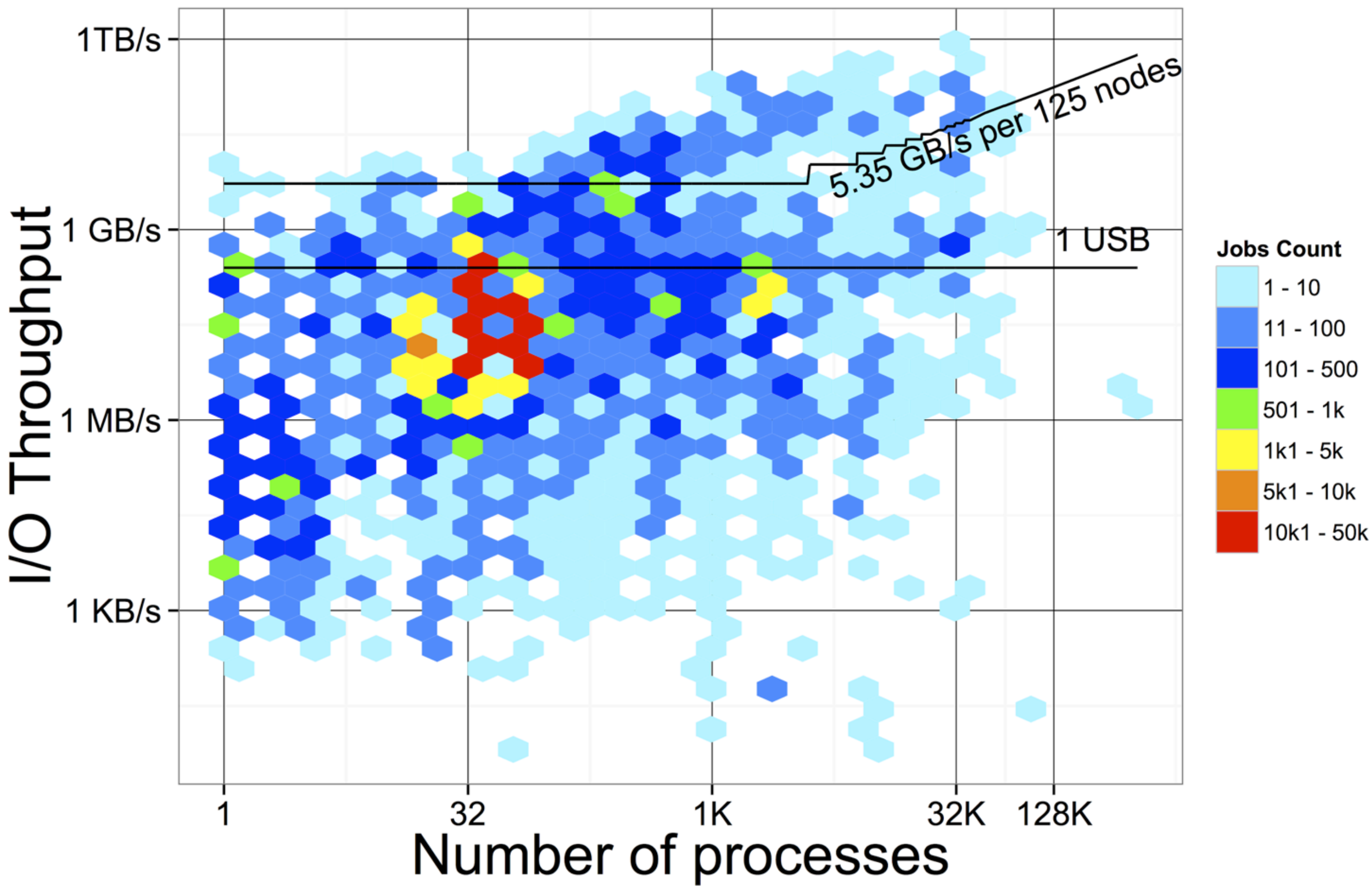
Most jobs transfer little data. Many big-data jobs also have very low thruput

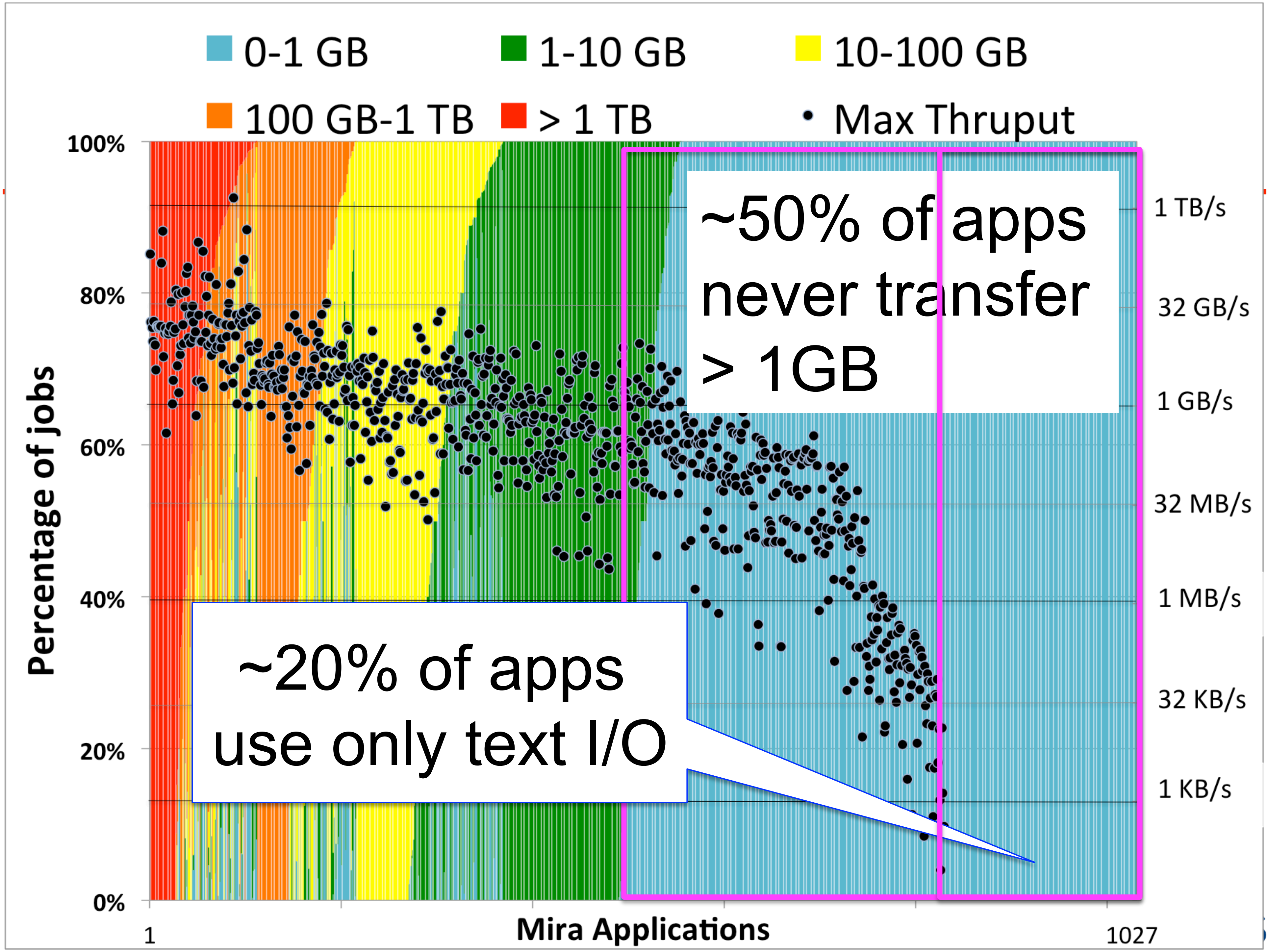


Most Jobs Read/Write Little Data (Blue Waters data)



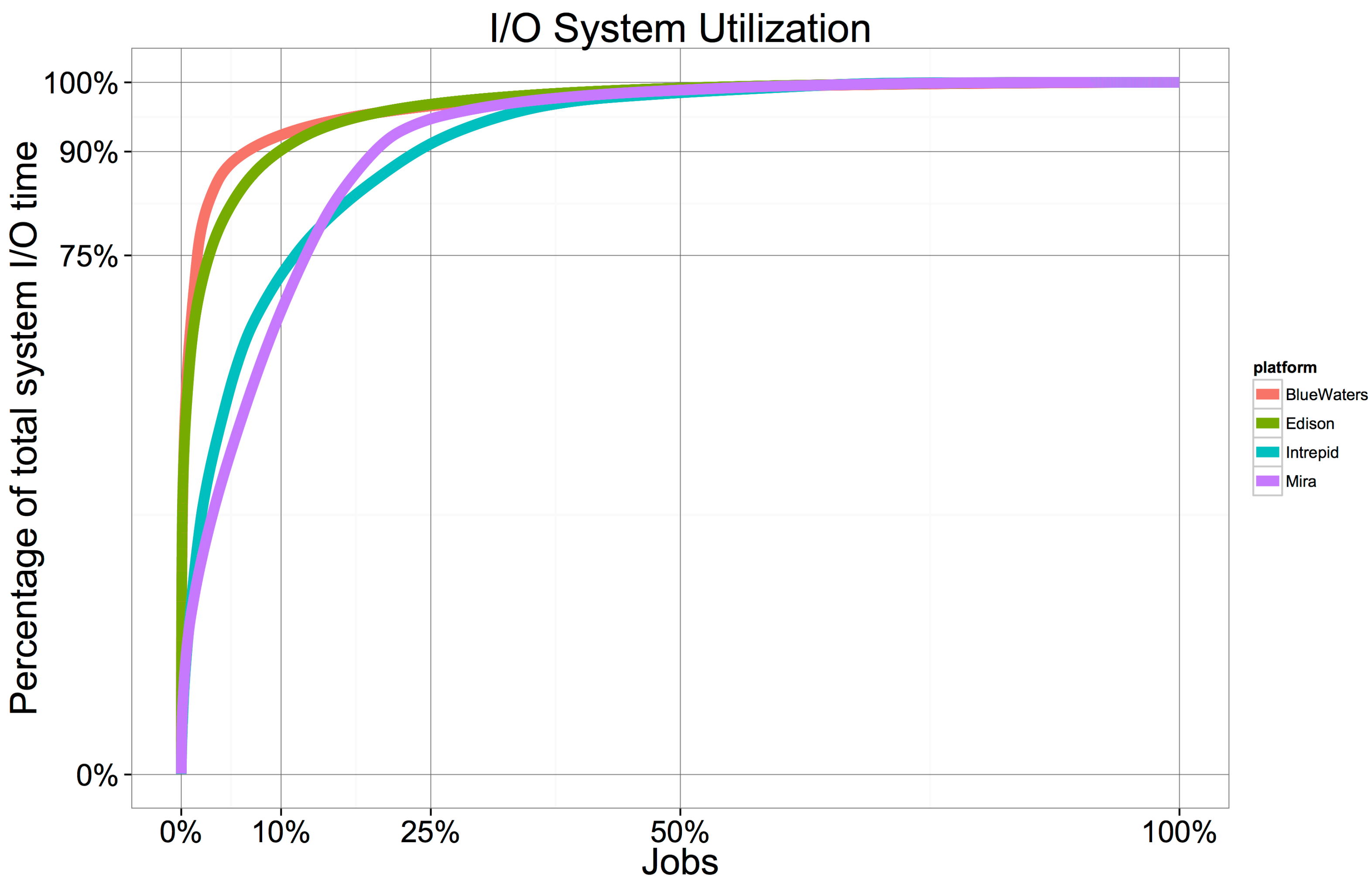
I/O Thruput vs Relative Peak





0-1 GB
1-10 GB
10-100 GB
100 GB-1 TB
> 1 TB
Max Thruput
100%
80%
60%
40%
20%
0%
Percentage of jobs
1 TB/s
32 GB/s
1 GB/s
32 MB/s
1 MB/s
32 KB/s
1 KB/s
1
Mira Applications
1027
~50% of apps never transfer > 1GB
~20% of apps use only text I/O

I/O Time Usage Is Dominated By A Small Number Of Jobs/Apps



PARALLEL@ILLINOIS

Improving the performance of the top 15 apps can save a lot of I/O time

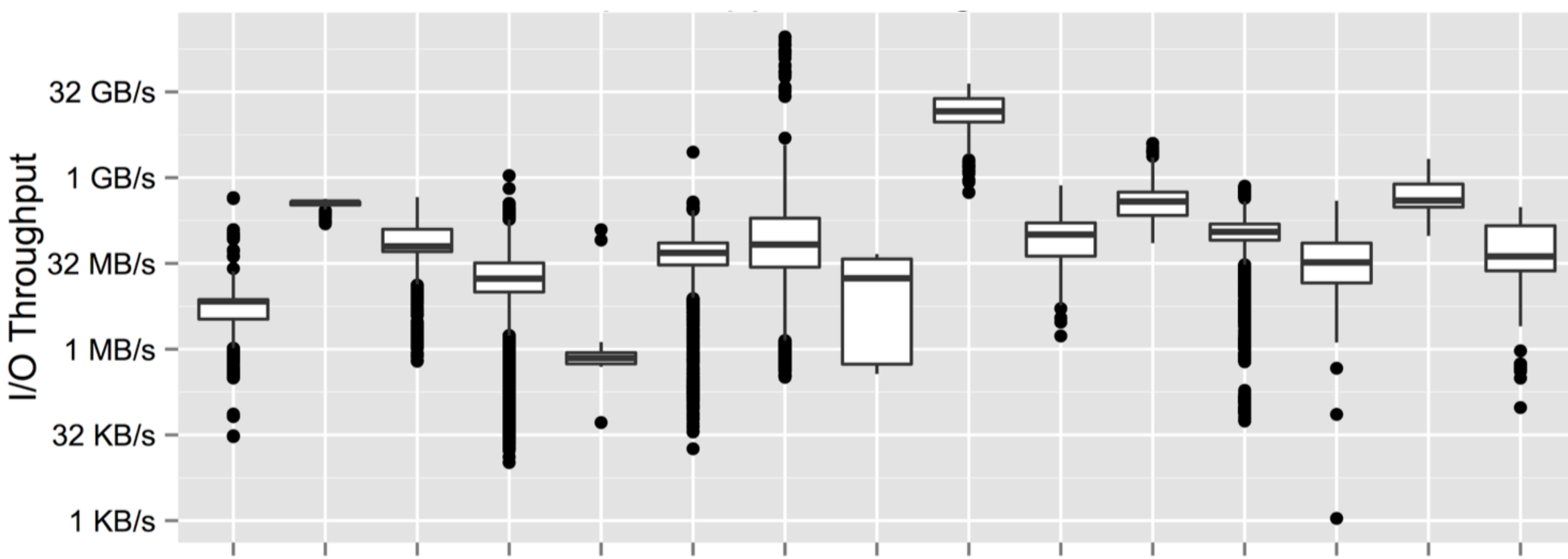
	Platform I/O time percent	Percent of platform I/O time saved if min thruput = 1 GB/s
Mira	83%	32%
Intrepid	73%	31%
Edison	70%	60%
Blue Waters	75%	63%

1867

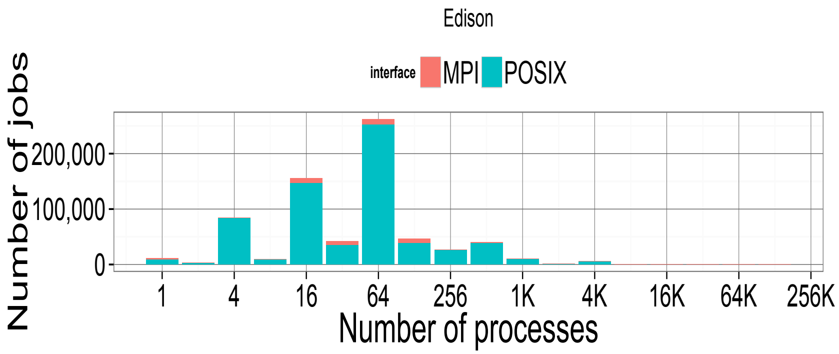
PARALLEL@ILLINOIS

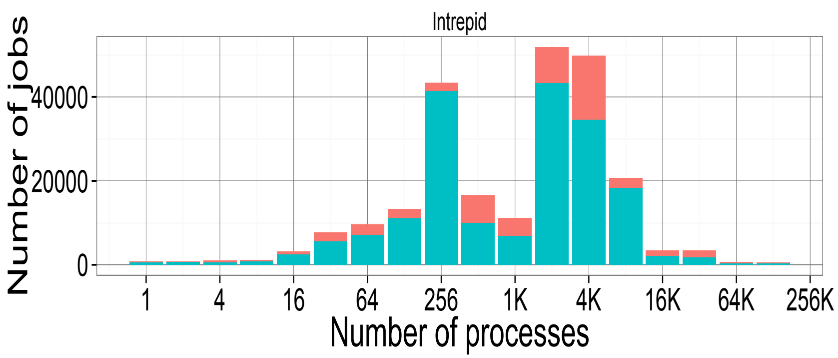
Top 15 apps with largest I/O time (Blue Waters)

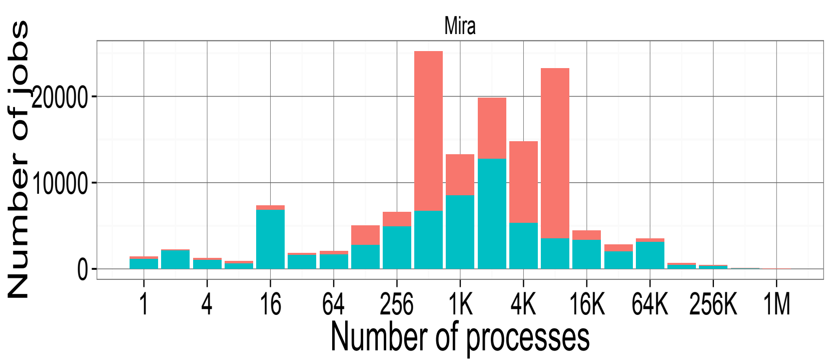
- Consumed 1500 hours of I/O time (75% total system I/O time)

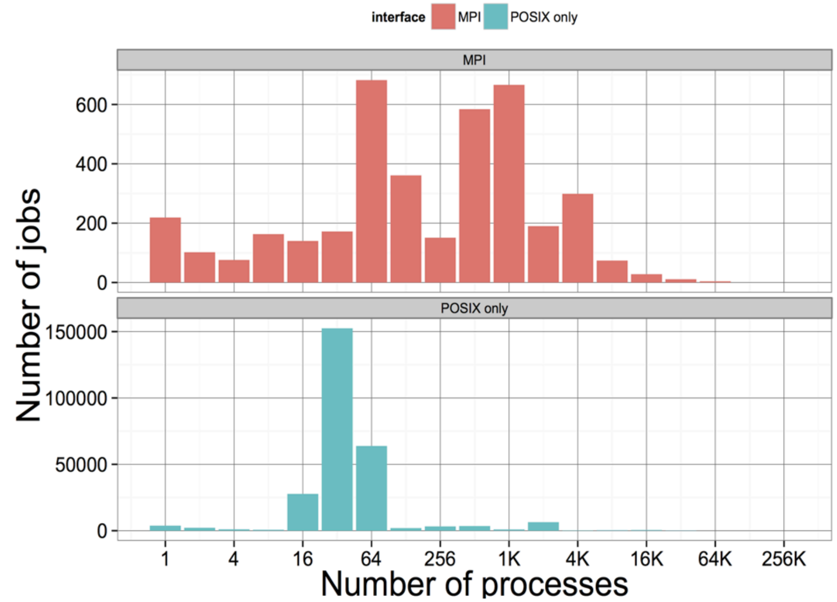


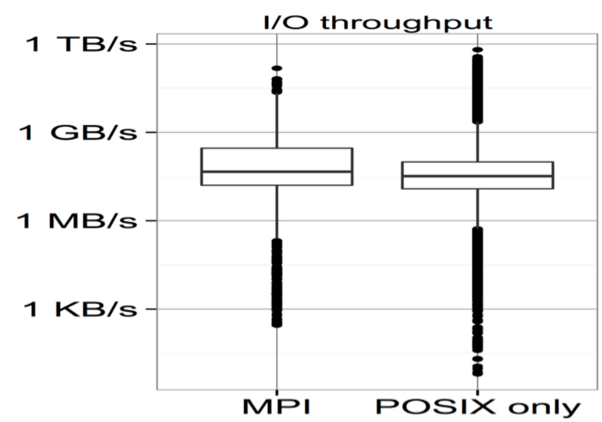
POSIX I/O is far more widely used than parallel I/O libraries.











PARALLEL@ILLINOIS

What Are Some of the Problems?

- POSIX I/O has a strong consistency model
  - Hard to cache effectively
  - Applications need to transfer block-aligned and sized data to achieve performance
- Files as I/O objects add metadata "choke points"
  - Serialize operations, even with "independent" files
- Burst buffers will *not* fix <u>these</u> problems – must change the semantics of the operations
- "Big Data" file systems have very different consistency models and meta data structures, designed for their application needs
  - Why doesn't HPC?
    - There have been some efforts, such as PVFS, but the <u>**requirement**</u> for POSIX has <u>**held up**</u> progress

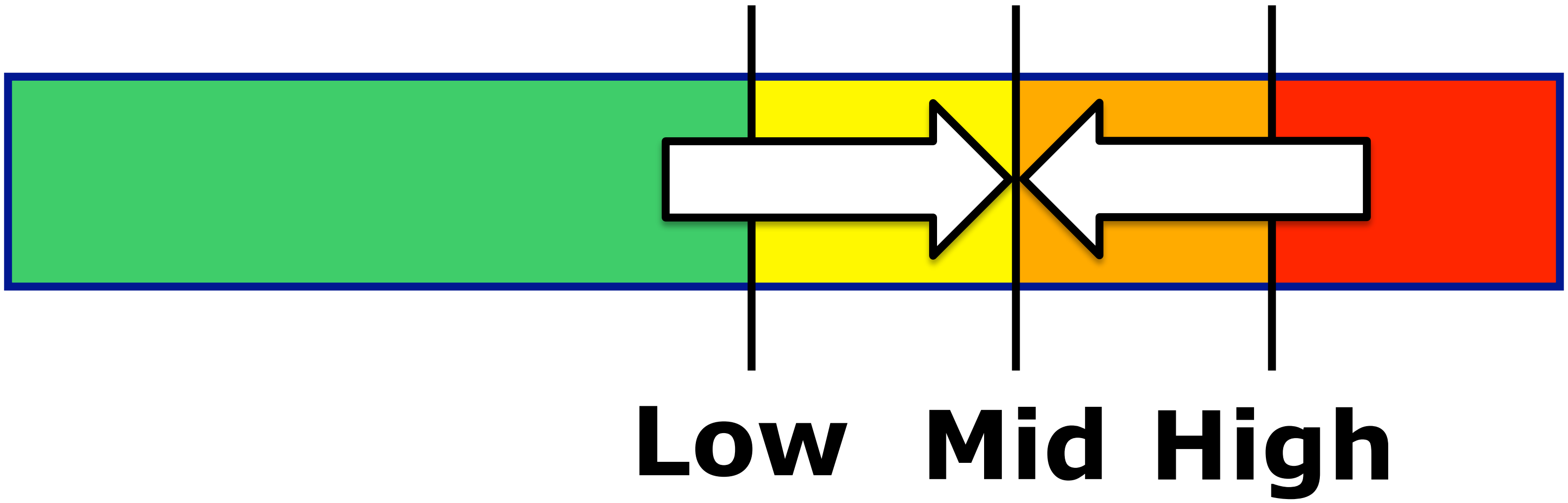
Big Data is More Than I/O

- One example is distributed, out-of-core graph processing
  - ♦ Constantly growing graph sizes with large memory footprints
  - ♦ Current distributed graph processing frameworks assume graphs fit in memory
    - Including all intermediate states
    - "Easy" but expensive fix is very large memory nodes
  - ♦ Can we use out-of-core techniques?
- This is work of Hassan Eslami, conducted during a summer internship at Facebook

Solution

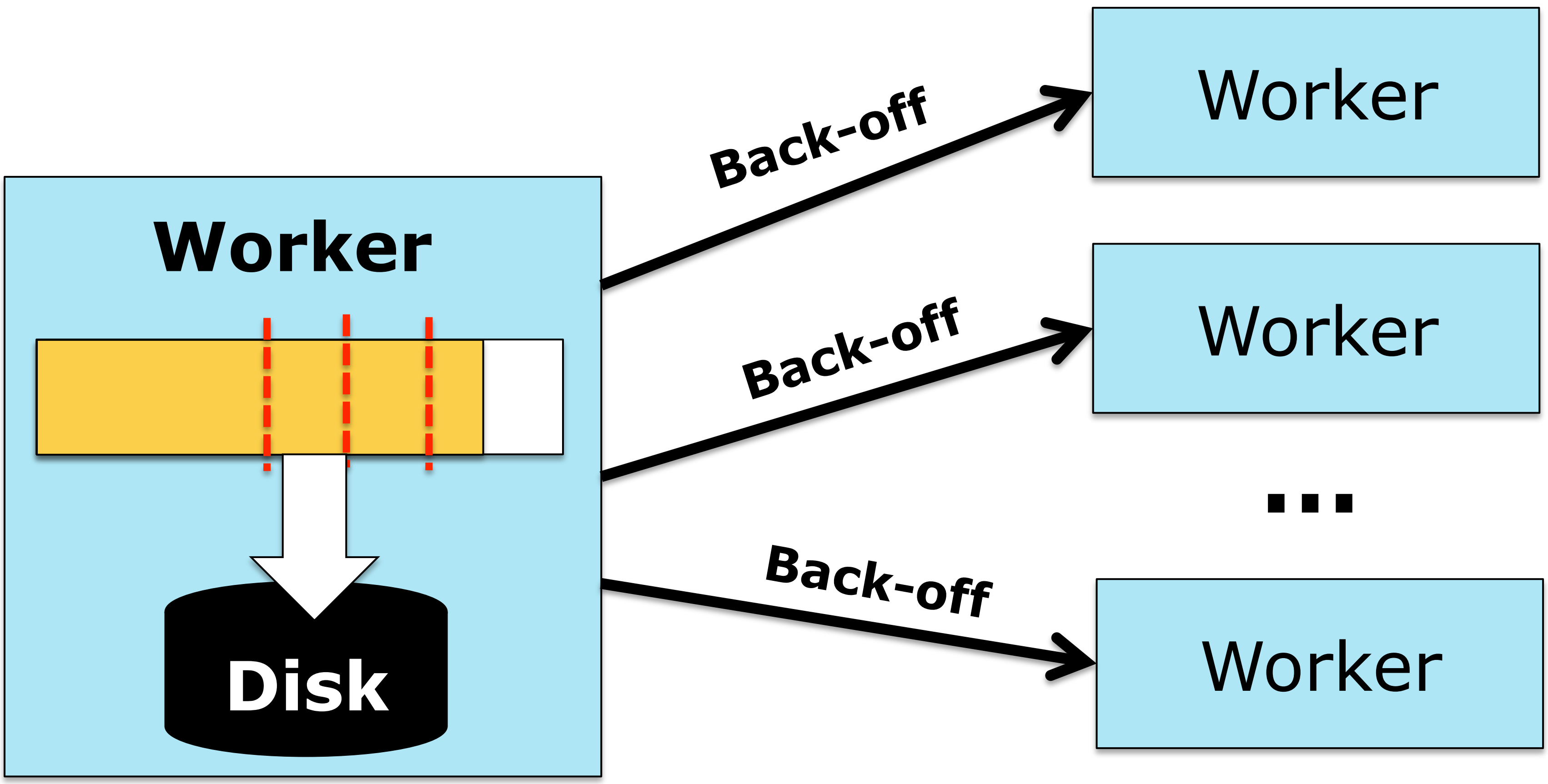
- We need a strategy to automatically and intelligently decide which data should be in-memory or out-of-core.
- This is done by:
  - ♦ Adaptive control of in-memory data
  - ♦ Congestion control of incoming messages
  - ♦ Capacity control of outgoing messages

Adaptive Control of In-memory Data

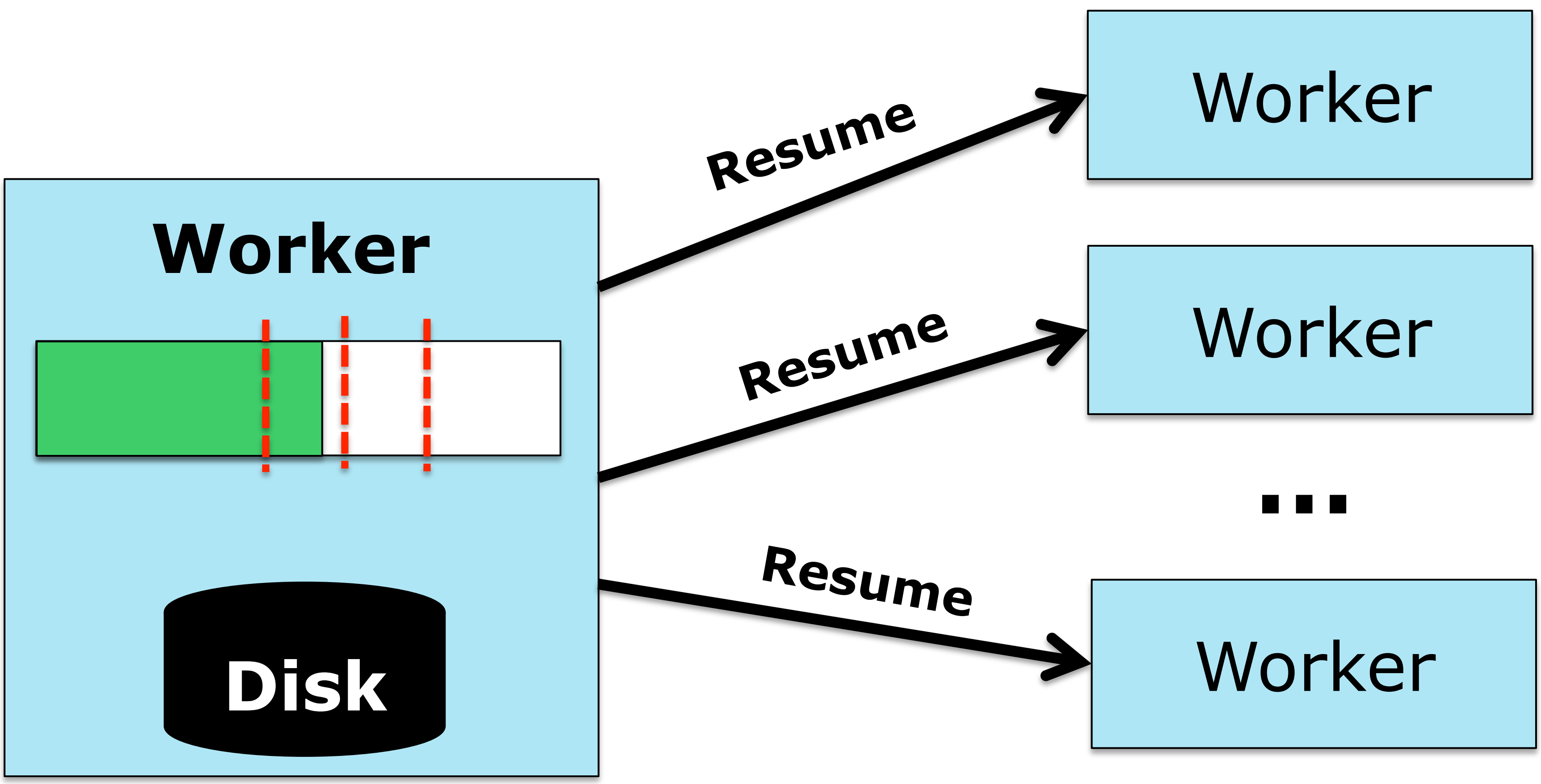


- **Data usage > High:** offload data to disk until usage below Mid
- **Data usage < Low:** lazily load data of latest offload from disk

Congestion Control of Incoming Messages



Congestion Control of Incoming Messages

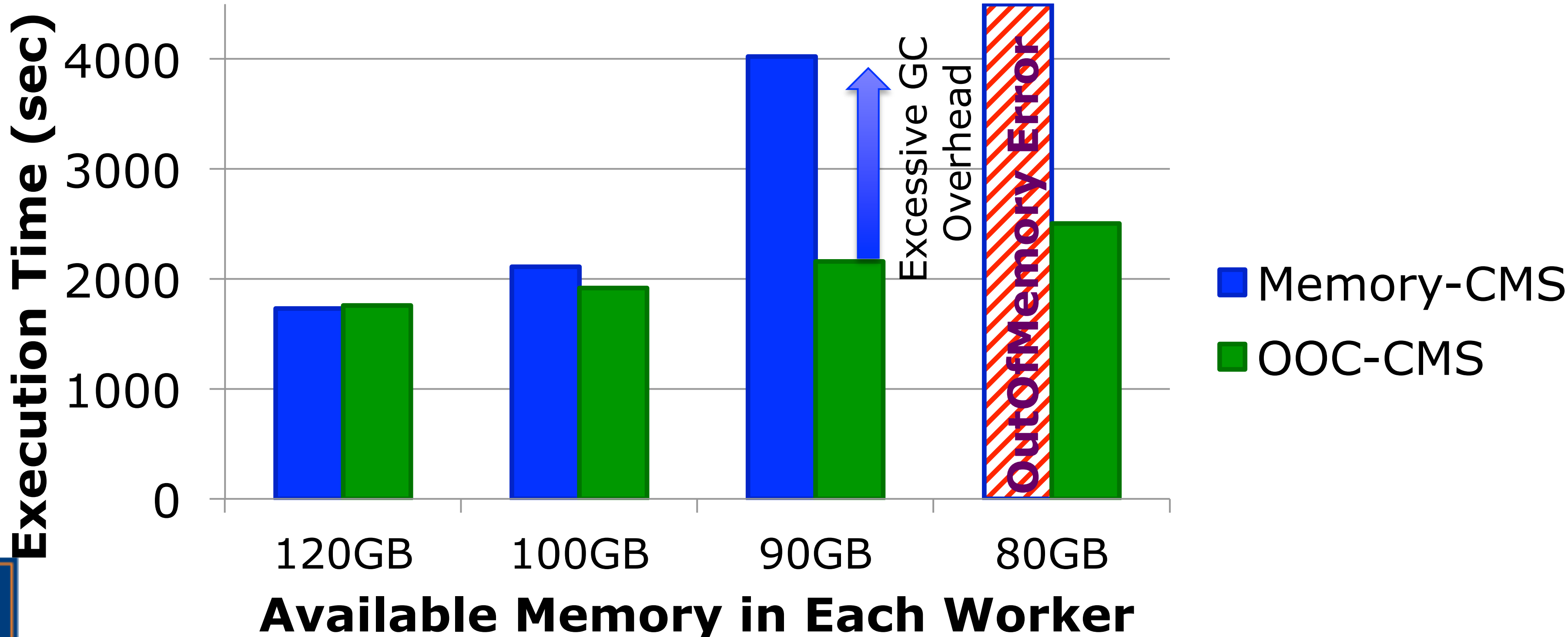


Capacity Control of Outgoing Messages

- Keeps a count of outgoing on-the-fly messages per worker pair
- Limits in-transit messages per each worker pair in a two phase approach
  1. count > MAX-IN-TRANSIT: cache the message
  2. size(cache) > MAX-CACHE-SIZE: stop computation

Result

- Implementation now available in Apache Giraph
- Results for PageRank on 8 workers on an input graph where graph data and messages take roughly 650GB with CMS as garbage collection strategy



PARALLEL@ILLINOIS

Observations

- Dealing with large graphs requires fast messaging
  - Issues such as memory management of “eager” data, flow control, nonblocking operations are important
  - Latency hiding in I/O also important
- Common programming model is BSP

PARALLEL@ILLINOIS

Message 1

- Current I/O performance is poor
  - Metadata operations often a significant source of poor performance
  - Related to mismatch between system and user expectations
    - CS Challenge: Better I/O consistency and programming models
    - Math Challenge: Match algorithms to realities of (changing) hardware; need aggregates, realistic model of data transfer costs

Message 2

- Big data is more than just I/O
  - And more than just operations on nearly independent data, for example…
  - Need to handle large graphs
    - CS Challenge: Low latency, high bandwidth, latency hiding programming and implementation, including multiple levels of memory hierarchy
    - Math Challenge: Match algorithms to problems; exploit years of effective sparse matrix work

Thanks!

- Especially Huong Luu, Babak Behzad, Hassan Eslami
- Funding from:
    - NSF
    - Blue Waters
- Partners at ANL, LBNL; DOE funding
- Internship support for Eslami from Facebook

BLUE WATERS
SUSTAINED PETASCALE COMPUTING