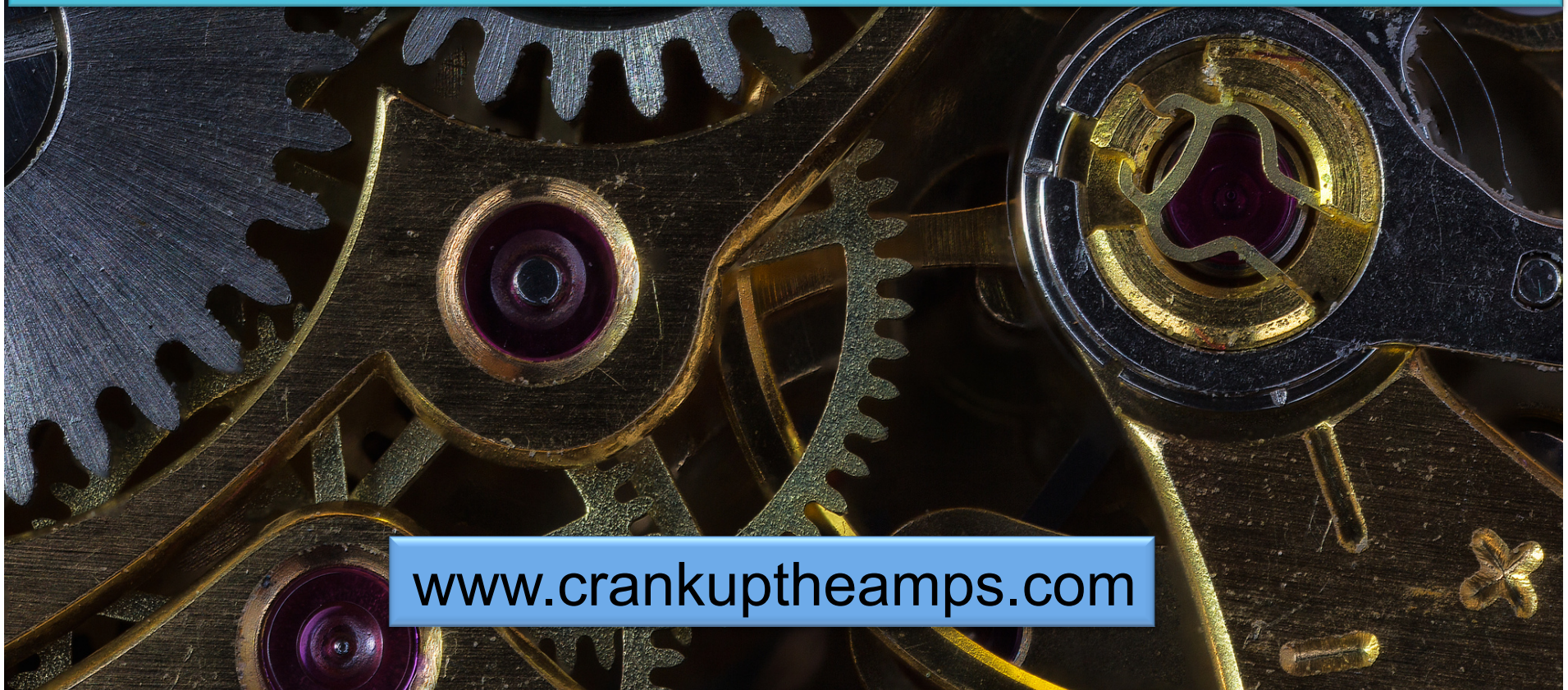
High Performance on Wall Street 2016 Challenges of HPC Code Writing – Capacity, Performance, Speed and Cost

Jeffrey M. Birnbaum

CEO and Co-Founder of 60East Technologies, Inc.

Makers of AMPS : The Advanced Messaging Processing System



Premise

By considering how our AMPS Technology outperforms popular systems,
we see how much systems leave on the table in terms of H/W resources ?

- **NoSQL : 30X Performance over MongoDB on Ingestion and Queries**

- **Queue : Over 25X the Throughput of RabbitMQ at up to 60X lower latency**
  - Reducing Footprint from 11 machines to 1

- **4 X More Throughput than a Pub Sub System with 0 message loss tolerance**

- **2.5 X More Durable Throughput than Hardware Messaging Appliance**

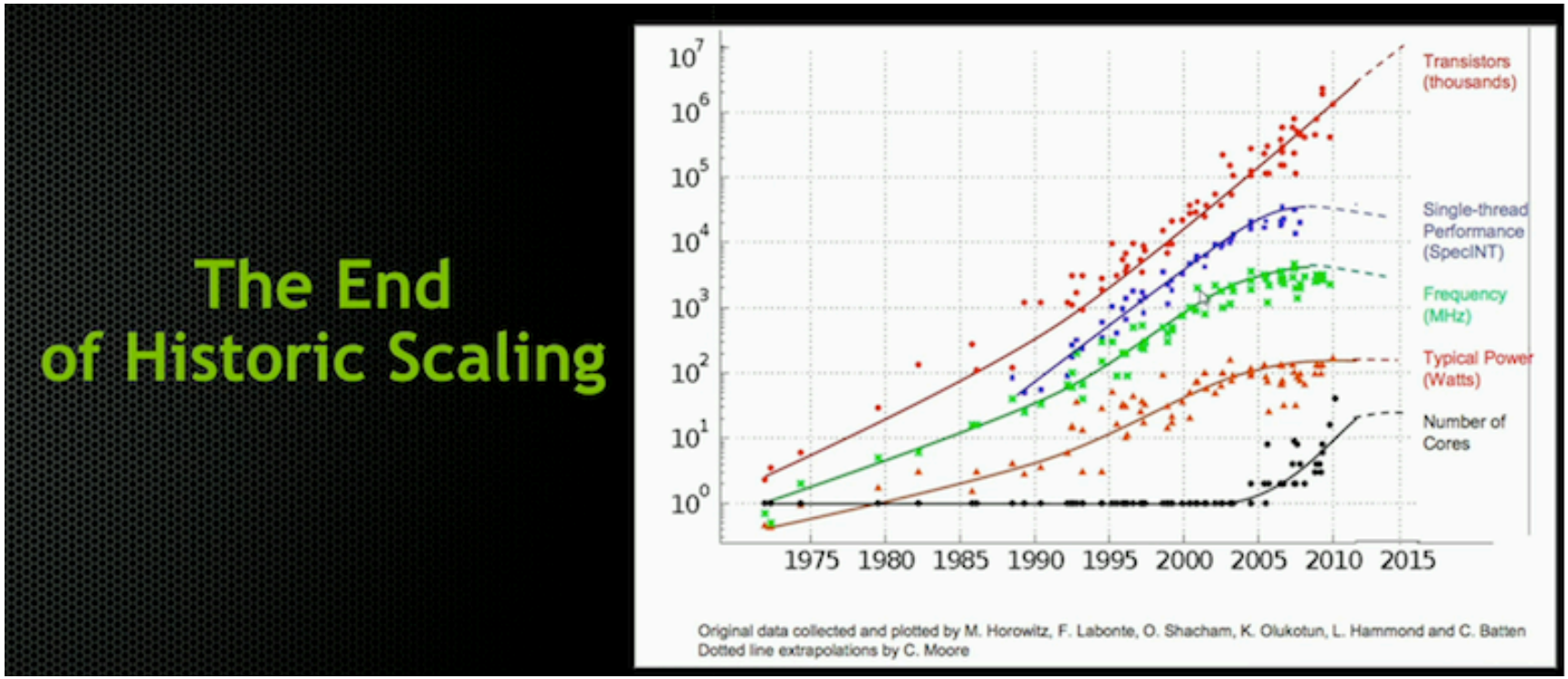
For more context, please see the following blog articles:

http://www.crankuptheamps.com//blog/posts/2016/02/26/rabbitmq-comparison-to-amps/
http://www.crankuptheamps.com//blog/posts/2015/07/22/reality-check-pure-software-beats-hardware/
http://www.crankuptheamps.com//blog/posts/2014/09/24/ultimate-shock-absorber/

Performance Drivers

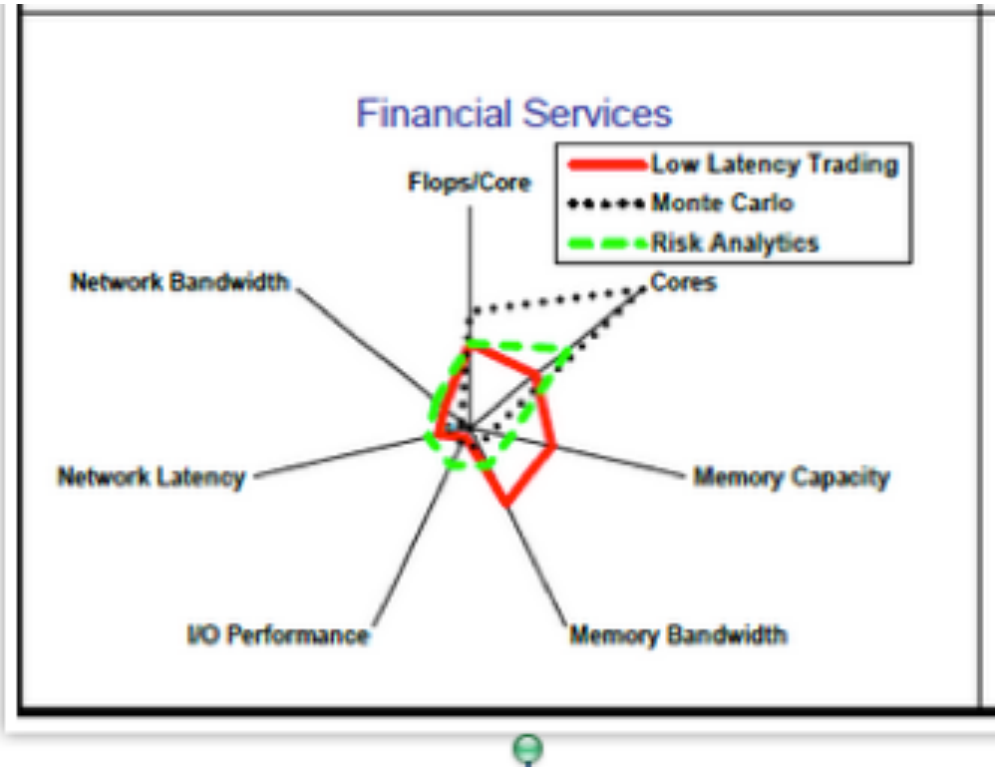
- **Traditional Drivers for HPC?**
  - Critical to Market Making and other low latency markets
  - Handle Larger and Larger Data Volumes and Load
  - Handle Peak Loads by leveraging full extent of Machine resources
- **Why HPC is now needed everywhere?**
  - Significantly Reduce H/W Footprint Costs by scaling out less
  - Lowering Latency enables one to add value-added capabilities that were previously cost prohibitive (i.e. streaming analytics for HPC or content filtering)

Truth or Myth: “No More Free Lunch”?



See Scale Up and Out :
http://www.crankuptheamps.com/downloads/documentation/CXO-Insight-Mar-2015.pdf

But, We are Leaving Too Much on the Table…..



Twenty Years ago, apps were designed for slow disks and networks and single threaded. It seems like they still are.

Architectural Patterns still assume multi-threaded is too hard, disks and networks are too slow and scaling out and out is the norm. We are learning that big data and real time problems require better thinking:
http://www.crankuptheamps.com//blog/posts/2015/10/01/nba-of-data-science/

Let's Review Some Important #s

Numbers (Jeff Dean says) Everyone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

http://static.googleusercontent.com/media/research.google.com/en//people/jeff/stanford-295-talk.pdf

Evolving Elements of HPC

- **CPUs** – Cores from 8 to 24
- **Storage** – march towards SSD, NVME, XPoint
- **Network** – around $3000 for a 100Gb per port (i.e. Arista 100Tb switch)
- **Memory** – 512 GB is common; Larger cache sizes

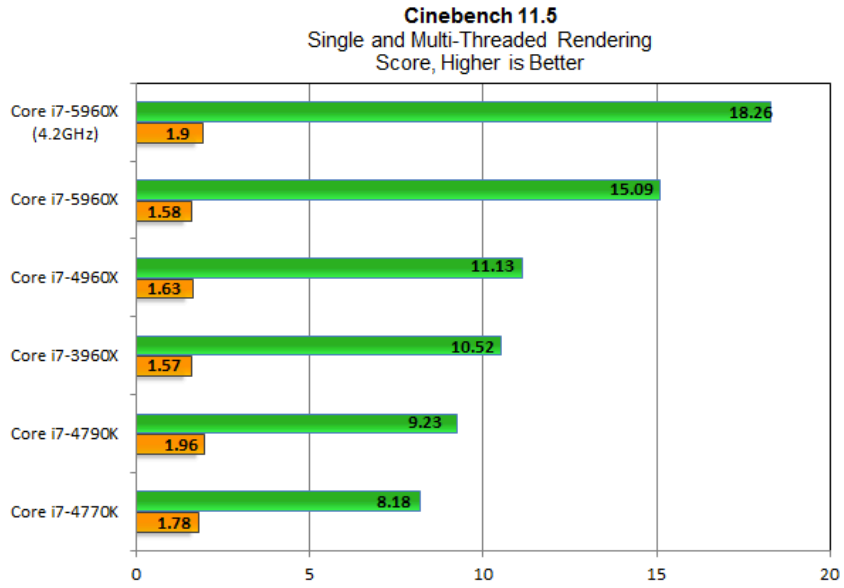
MEMORY NETWORK CPU STORAGE

CPUs, Cores and Concurrency

- **Embracing Multi-core**
  - Multi-Threaded
  - Beyond the 80-20 Rule
    - Tune Everything
  - Lock free Data Structures
    - Generation Count

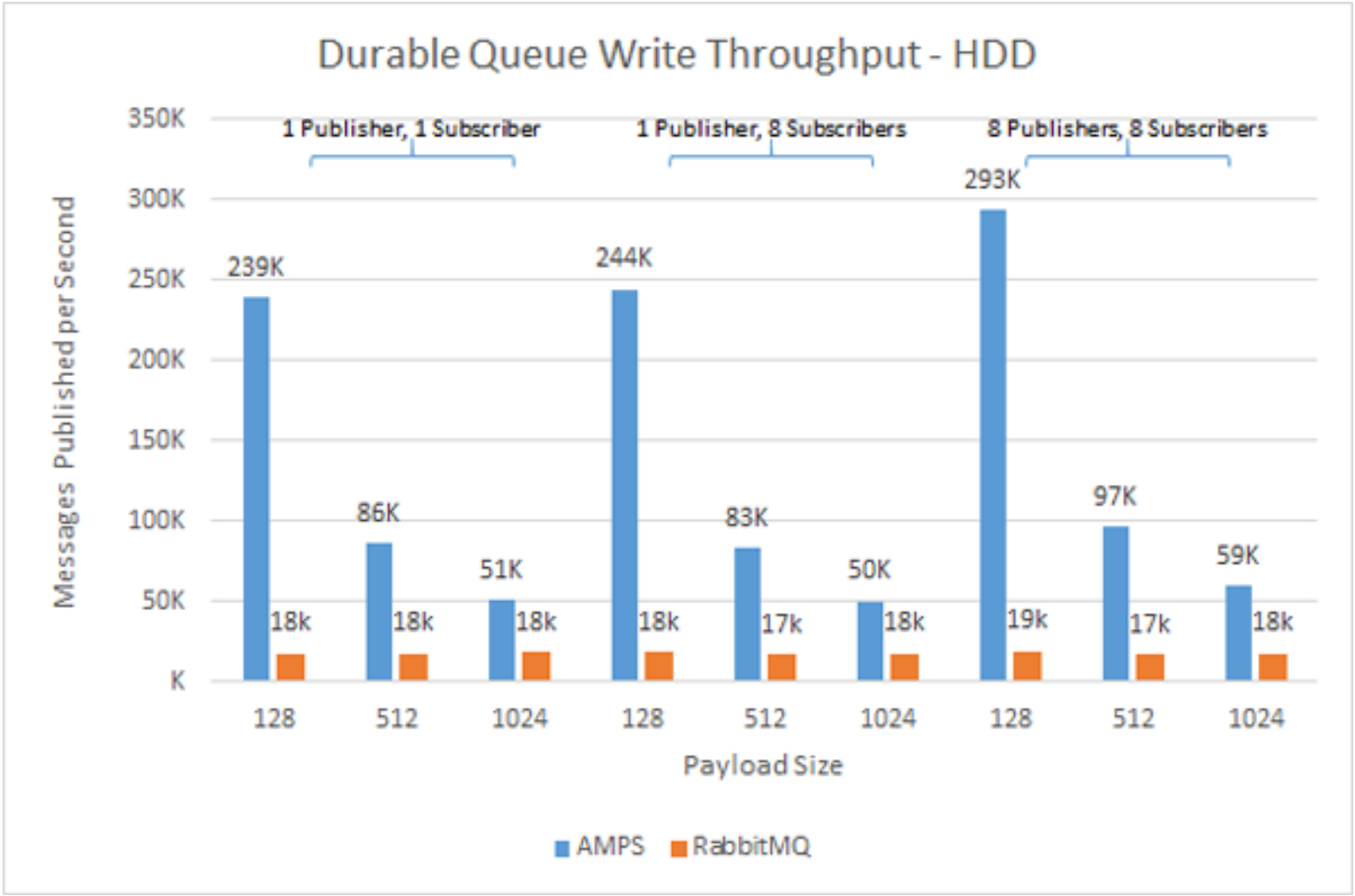
- **NUMA tuning**
  - Most DB products say turn NUMA off. We keep it on because we did the hard work.

- **Throughput :**
  - Message Pipelines
  - Divide and Conquer



http://www.extremetech.com/computing/188911-intel-haswell-e-review-the-best-consumer-performance-chip-you-can-buy-with-some-caveats/2

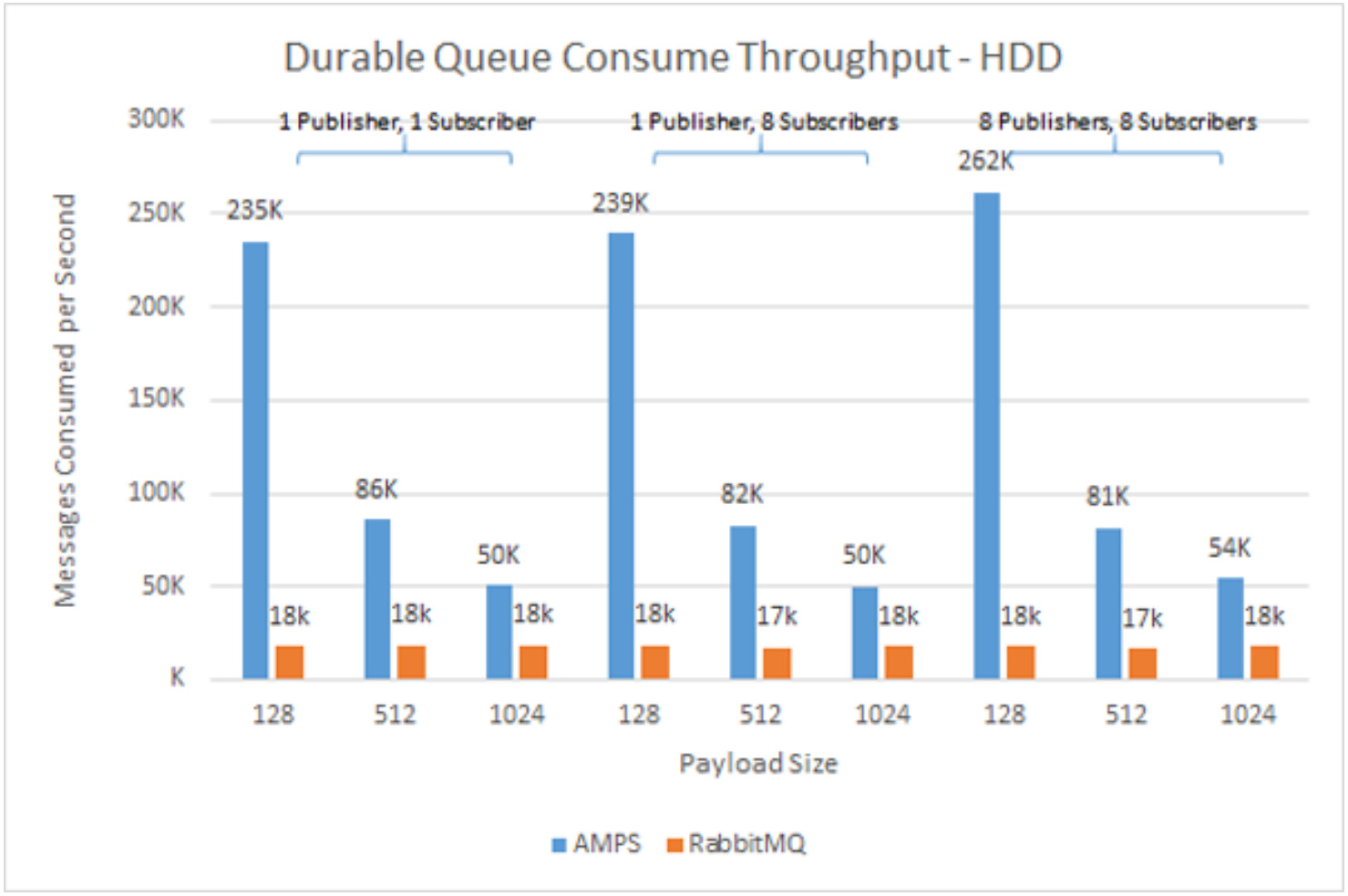
Leaving Things on the Table



One is not taking advantage of the resources . AMPS Queue store and forward model is impacted by larger message sizes written to persistent disk. The bottle neck should be due to the h/w limits.

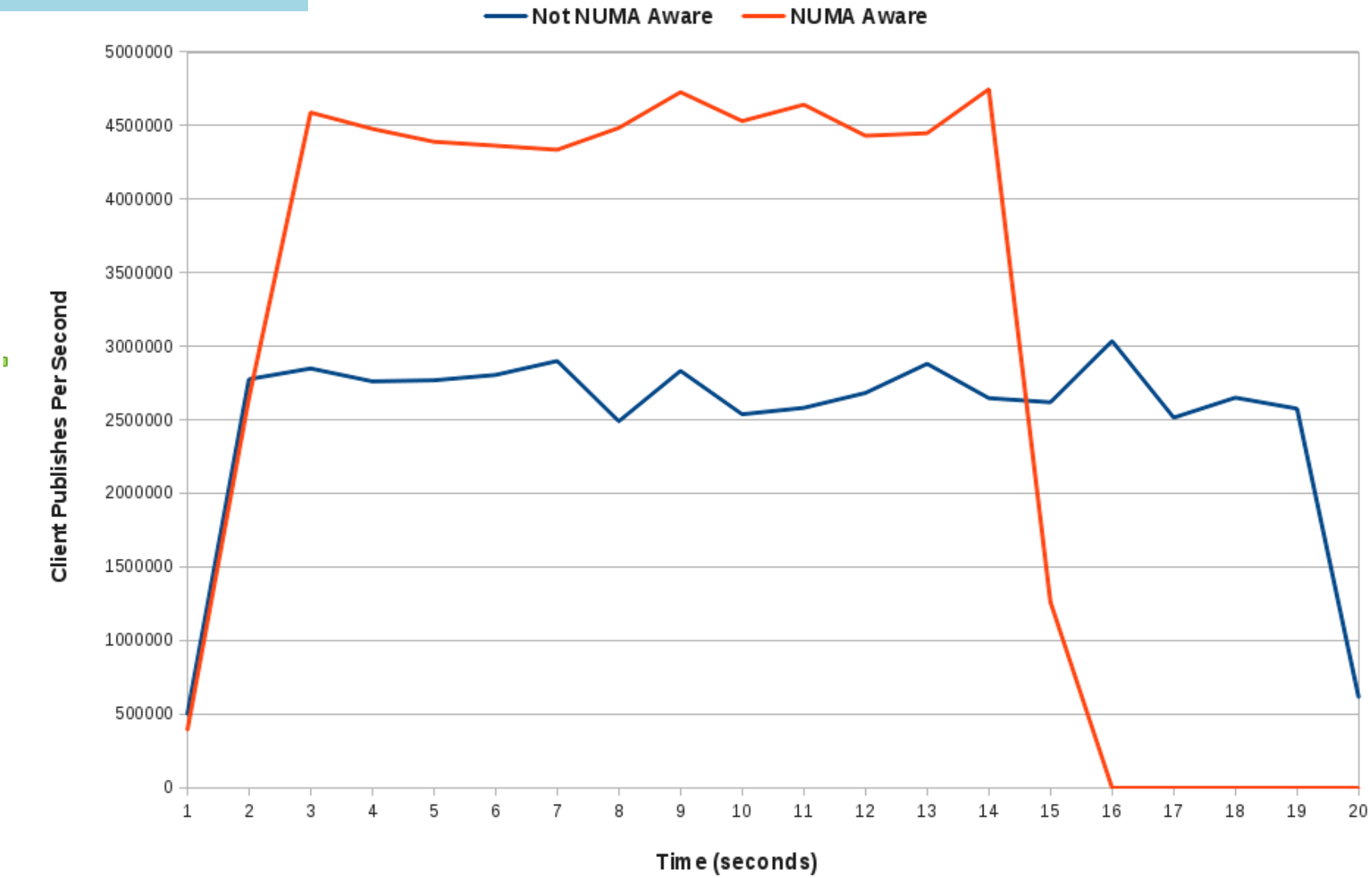
http://www.crankuptheamps.com//blog/posts/2016/02/26/rabbitmq-comparison-to-amps/

Leaving Things on the Table



http://www.crankuptheamps.com//blog/posts/2016/02/26/rabbitmq-comparison-to-amps/

Leaving Things on the Table



http://www.crankuptheamps.com/downloads/documentation/hpcws-apr2013-v4.pdf

Best Practices 1

Focus on Forward Scalability :

When the next chip comes with a jump from 14 to 28 cores on a single die, the more concurrency that can be realized.

Flip Side : Attention to Detail

Single threaded can be less worse than poorly/ excessively implemented locking. *They are only scaling via partitioning the data i.e. REDIS, VoltDB*

Best Practices 2

- **Many Approaches:**
  - ScyllaDB /Cassandra -are sending work across node constantly without NUMA-ness. (A Dispatch Model that partitions per core)
  - Just got to do it the right way (every core is assigned a processing engine- with a wide scale dispatch (i.e. 48 core= 48 workers (+ dispatch worker)).

- **Obvious thing isn't always the best thing in a highly concurrent context**. A "B-Tree" is common for db index schemes but its not the best data structure for highly concurrent activities. (30 X performance gain over MongoDB in Ingestion + Querying).

- **i.e. AMPS doesn't have a single model**.. model for IO is not the same as querying (i.e. parallel divide and conquer) We partition what is important across the machine.

Storage

- **How would you write your system differently if you knew there would be a 20X to 40X improvement in storage I/O?**

- **Disk**
  - Fine for Log Appending (low disk head movement/seek)

- **SSD**
  - Memory Mapped Files, Key-Value Stores

- **Promise of XPoint**

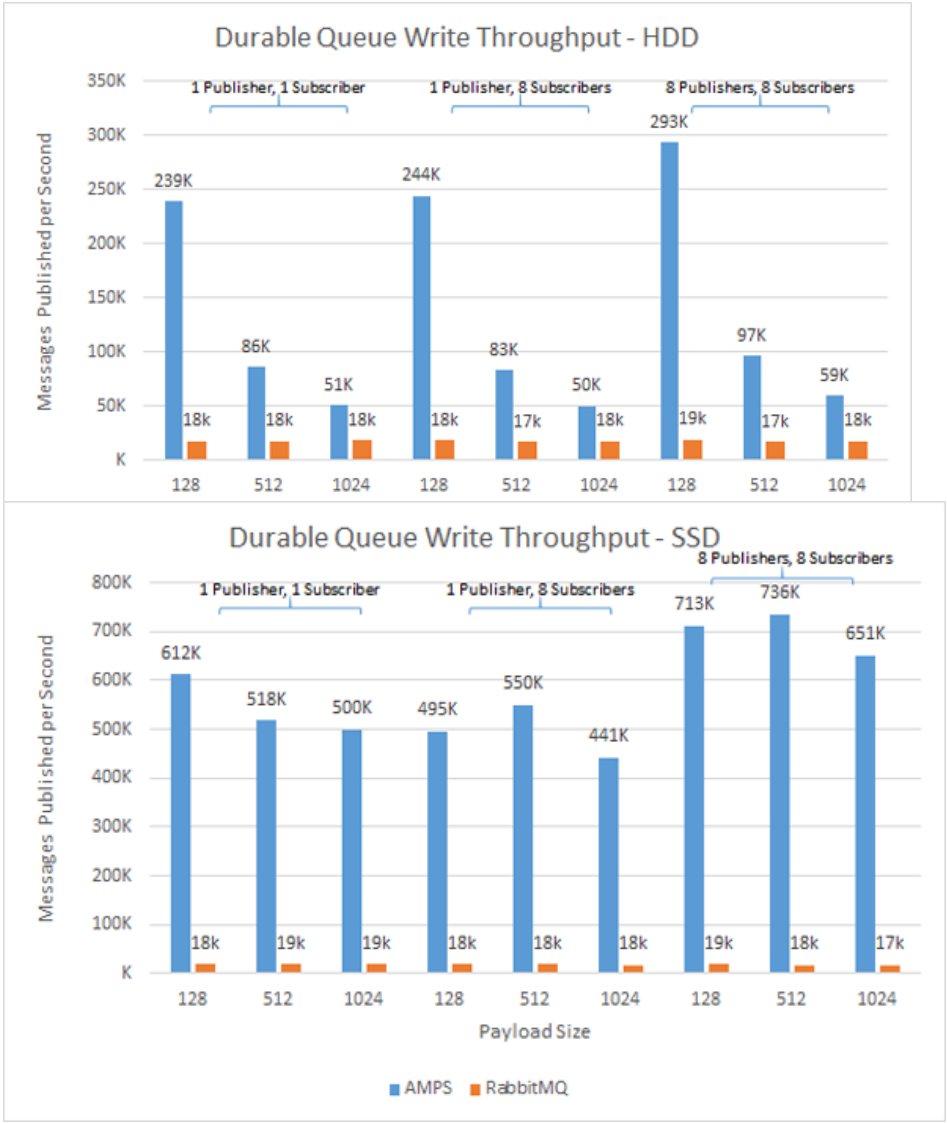
  80 ms throughput limited,

  Once you hit memory limit, it on is stuck at 80ms due to back pressure

  *What if that goes down to 8ms

http://www.crankuptheamps.com//blog/posts/2014/12/08/extreme_storage_performance/
http://www.crankuptheamps.com//blog/posts/2014/05/01/amps-faster-than-ever-with-memory-channel-storage/

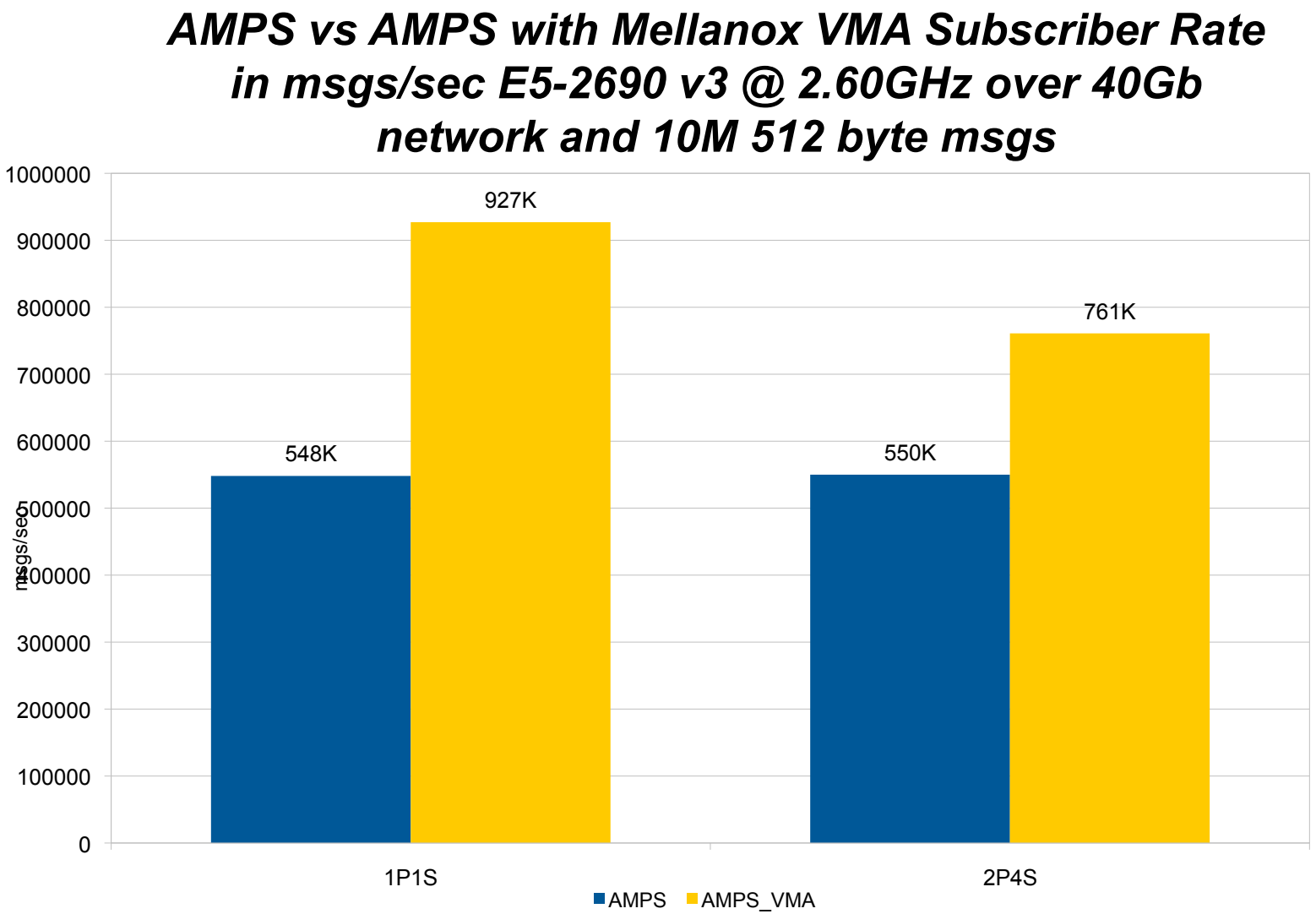
Not Leaving Things on the Table



Invest in a better storage device, and the software should reward you…..

Networking Advancements

How would you write your system differently if you knew there would be a 10X improvement in network I/O?



These are preliminary #s – we haven't optimized it, this was a simple LD_Preload

Memory

- Know your cache lines – and keep things in cache whenever possible
- Keep your structures and access patterns aligned (partition read/write sections)
- Minimize Heap Memory Allocation
- Leverage Modern Allocators
- Avoid Thread Bleeding

Wash, Rinse, Repeat

- Scale Up and then Out
- Scale Forward (“enjoy the free lunch”) and plan for advancements (i.e. 40X Xpoint)
- Forget about 80-20 , Optimize every last part of your code base
- Leverage Many Models/Approaches; Continually Improve

Thank You!

Any Questions?

Feel free to contact me at:

JMB@CRANKUPTHEAMPS.COM