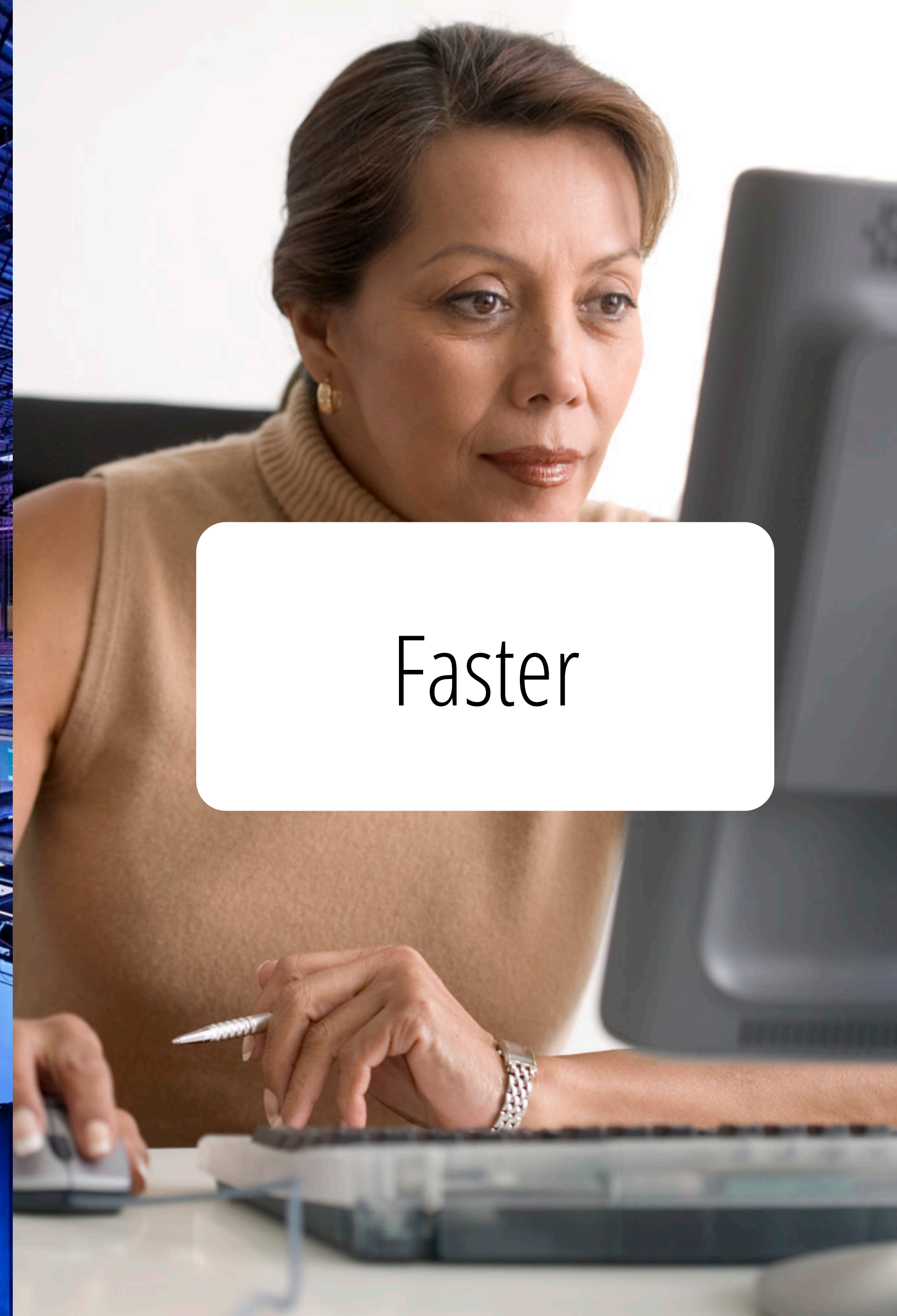
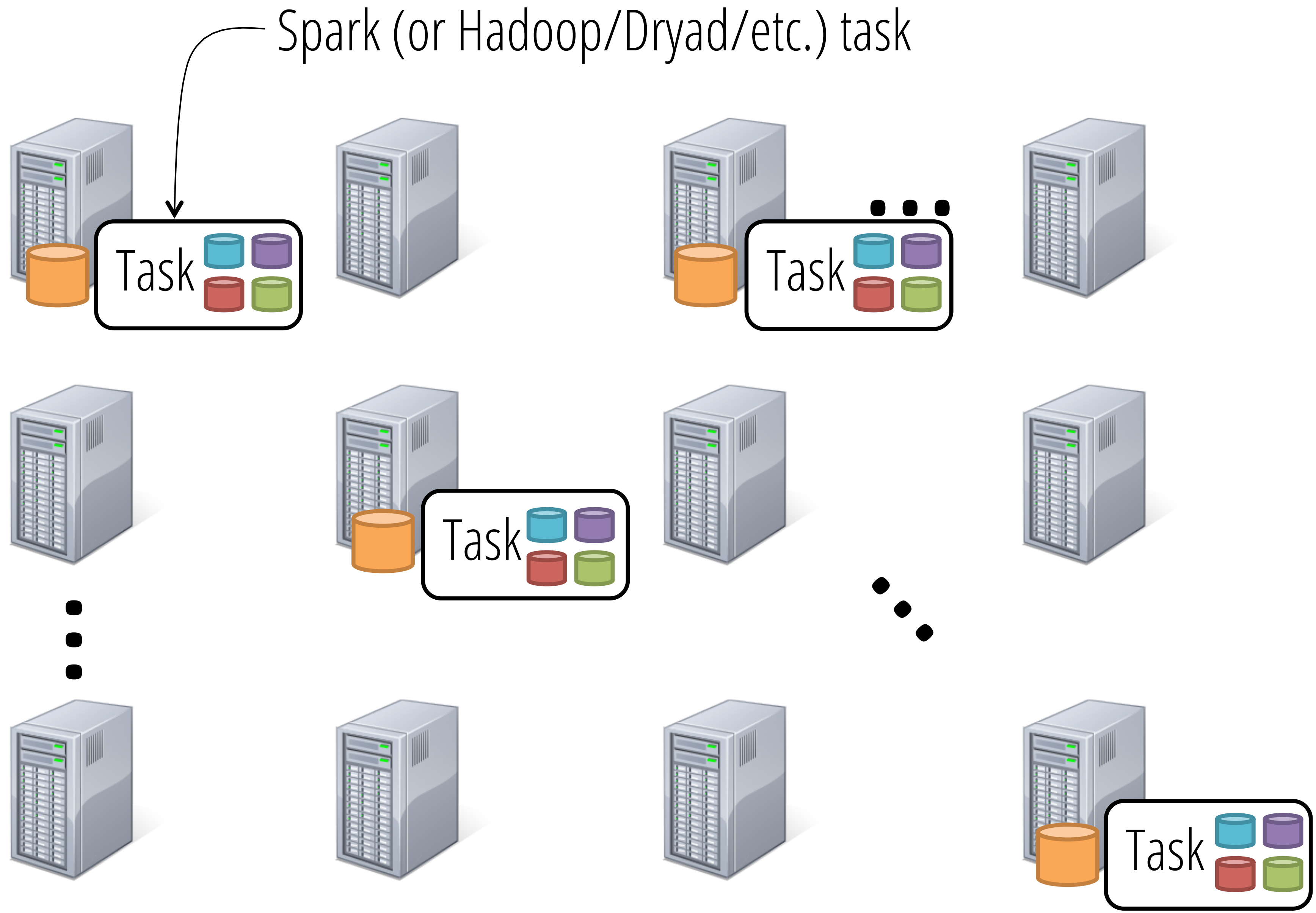
Making Sense of Performance in Data Analytics Frameworks

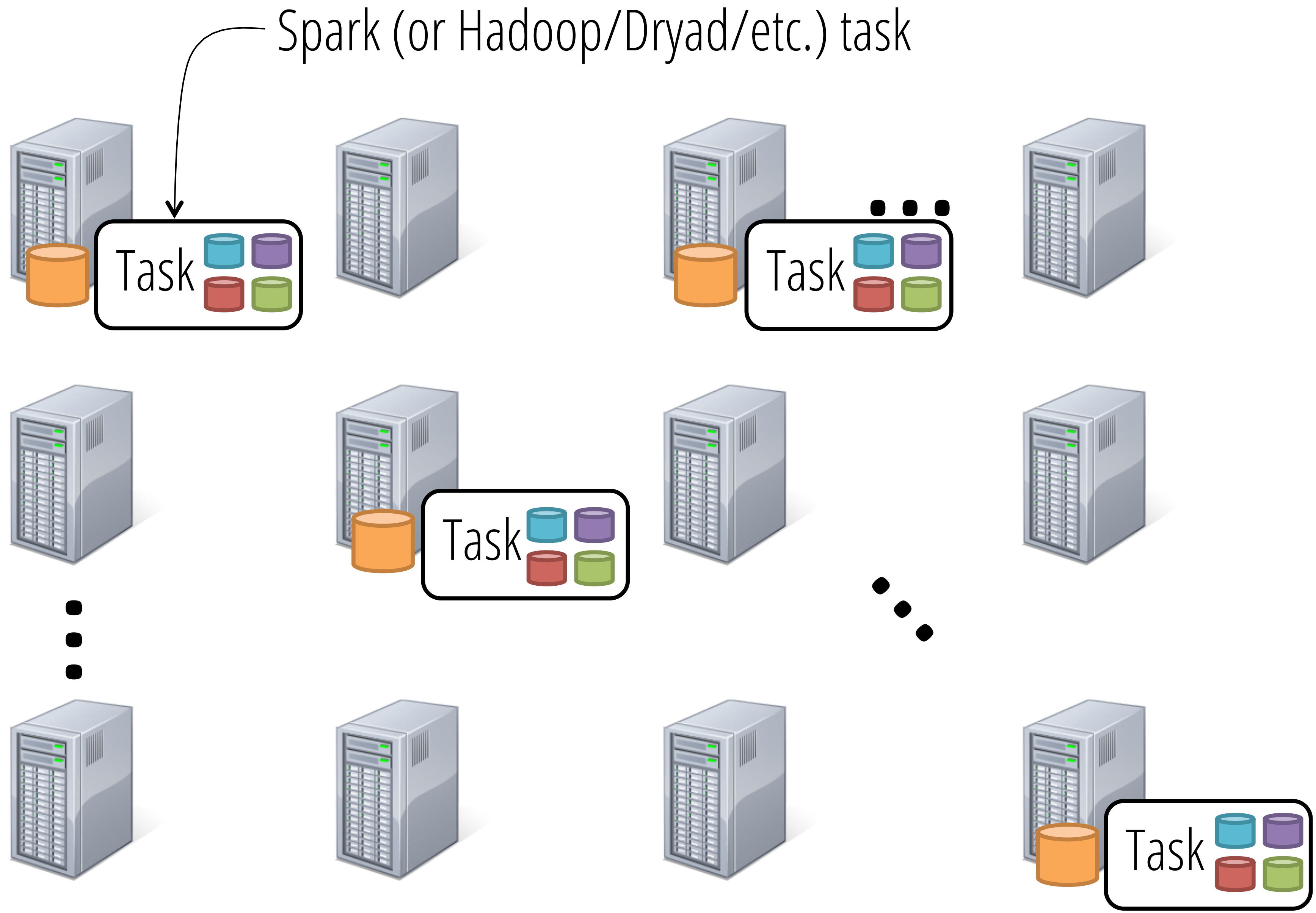
Kay Ousterhout, Ryan Rasti, Sylvia Ratnasamy,
Scott Shenker, Byung-Gon Chun

Large-scale data analytics has become widespread

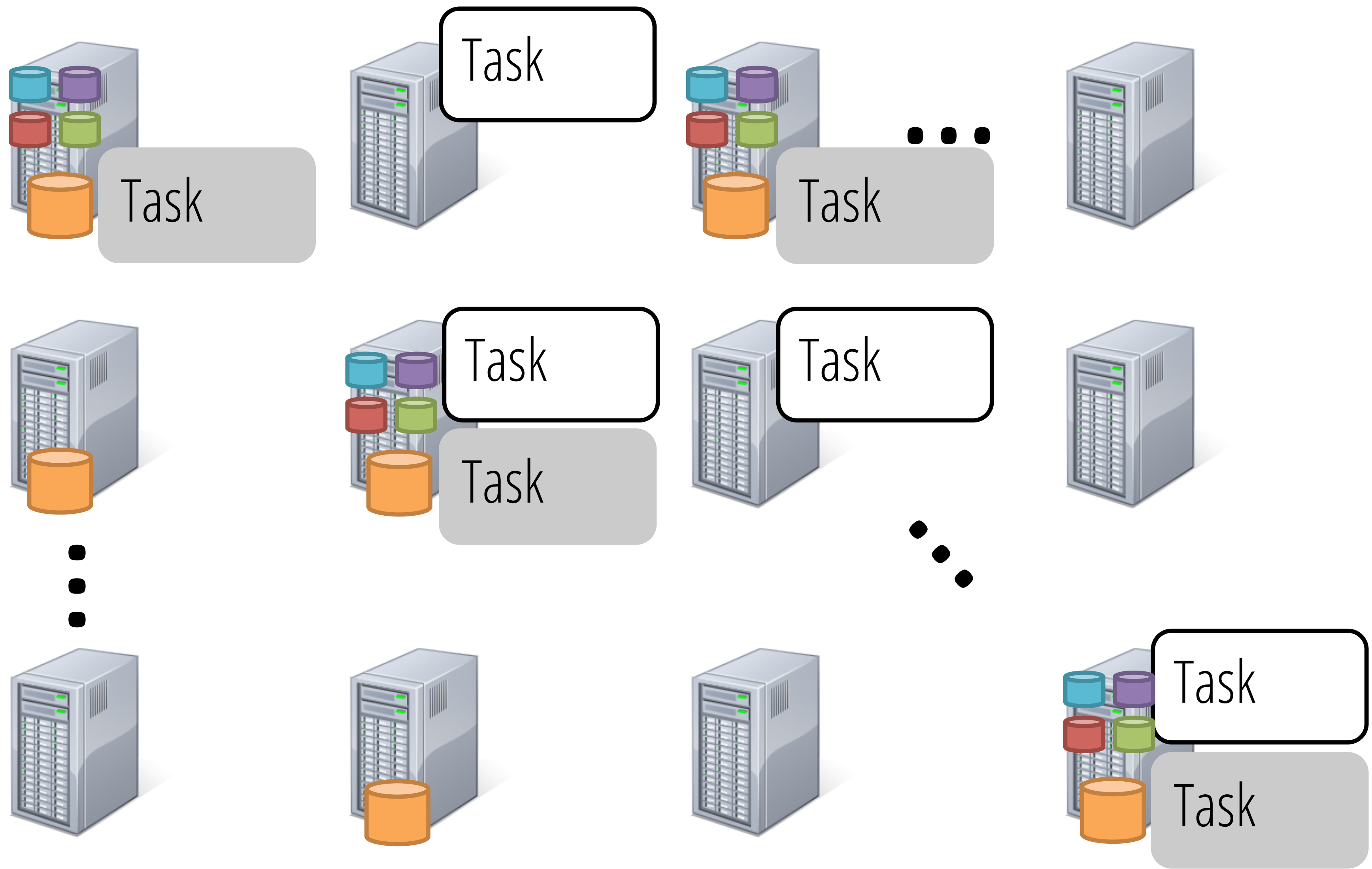

More resource-efficient


Faster


Spark (or Hadoop/Dryad/etc.) task
Task
Task
Task
Task



Spark (or Hadoop/Dryad/etc.) task
Task
Task
Task
Task


Task
Task
Task
Task
Task
Task
Task
Task

Network

Load balancing: VL2 [SIGCOMM ‘09], Hedera [NSDI ’10], Sinbad [SIGCOMM ’13]
Application semantics: Orchestra [SIGCOMM ’11], Baraat [SIGCOMM ‘14], Varys [SIGCOMM ’14]
Reduce data sent: PeriSCOPE [OSDI ‘12], SUDO [NSDI ’12]
In-network aggregation: Camdoop [NSDI ’12]
Better isolation and fairness: Oktopus [SIGCOMM ’11], EyeQ [NSDI ‘12], FairCloud [SIGCOMM ’12]

Disk

Themis [SoCC ‘12], PACMan [NSDI ’12], Spark [NSDI ’12], Tachyon [SoCC ’14]

Stragglers

Scarlett [EuroSys ‘11], SkewTune [SIGMOD ‘12], LATE [OSDI ‘08], Mantri [OSDI ‘10], Dolly [NSDI ‘13], GRASS [NSDI ‘14], Wrangler [SoCC ’14]

Network

Load balancing: VL2 [SIGCOMM '09], Hedera [NSDI '10], Sinbad [SIGCOMM '13]
Application semantics: Orchestra [SIGCOMM '11], Baraat [SIGCOMM '14], Varys [SIGCOMM '14]
Reduce data sent: PeriSCOPE [OSDI '12], SUDO [NSDI '12]
In-network aggregation: Camdoop [NSDI '12]
Better isolation and fairness: Oktopus [SIGCOMM '11]), EyeQ [NSDI '12], FairCloud [SIGCOMM '12]

Missing: what's most important to end-to-end performance?

Disk

Themis [SoCC '12], PACMan [NSDI '12], Spark [NSDI '12], Tachyon [SoCC '14]

Stragglers

Scarlett [EuroSys '11], SkewTune [SIGMOD '12], LATE [OSDI '08], Mantri [OSDI '10], Dolly [NSDI '13], GRASS [NSDI '14], Wrangler [SoCC '14]

Network

Load balancing: VL2 [SIGCOMM '09], Hedera [NSDI '10], Sinbad [SIGCOMM '13]
Application semantics: Orchestra [SIGCOMM '11], Baraat [SIGCOMM '14], Varys [SIGCOMM '14]
Reduce data sent: PeriSCOPE [OSDI '12], SUDO [NSDI '12]
In-network aggregation: Camdoop [NSDI '12]
Better isolation and fairness: Oktopus [SIGCOMM '11], EyeQ [NSDI '12], FairCloud [SIGCOMM '12]

Disk

Themis [SoCC '12], PACMan [NSDI '12], Spark [NSDI '12], Tachyon [SoCC '14]

Stragglers

Scarlett [EuroSys '11], SkewTune [SIGMOD '12], LATE [OSDI '08], Mantri [OSDI '10], Dolly [NSDI '13], GRASS [NSDI '14], Wrangler [SoCC '14]

Widely-accepted mantras:

Network and disk I/O are bottlenecks

Stragglers are a major issue with unknown causes

This work

(1) How can we quantify performance bottlenecks?

Blocked time analysis

(2) Do the mantras hold?

Takeaways based on three workloads run with Spark

Takeaways based on three Spark workloads:

Network optimizations
can reduce job completion time by **at most 2%**

CPU (not I/O) often the bottleneck
<19% reduction in completion time from optimizing disk

Many straggler causes can be identified and fixed

Takeaways will not hold
for every single analytics workload
nor for all time

This work:

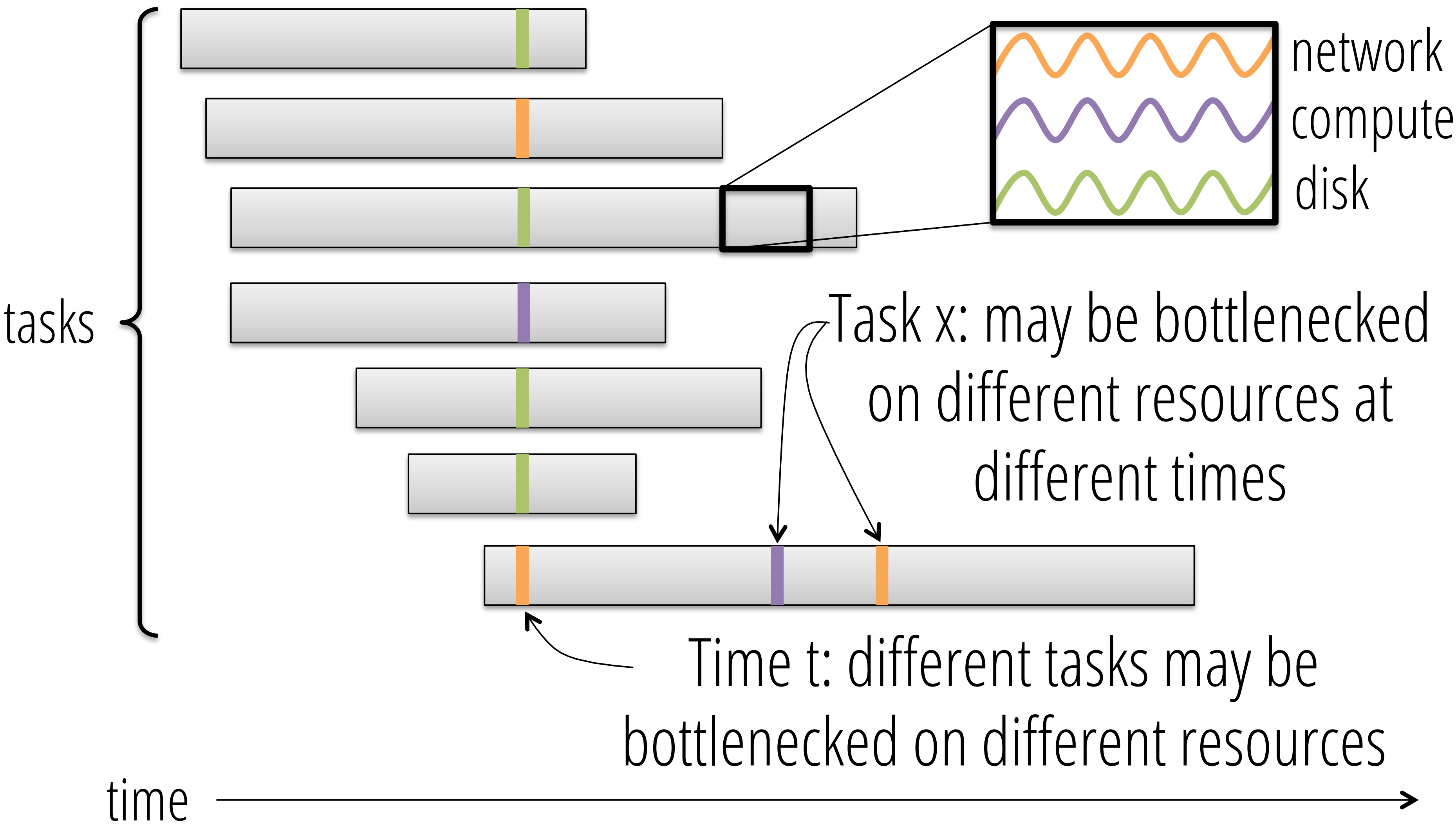
Accepted mantras are often not true

Methodology to avoid performance misunderstandings in the future

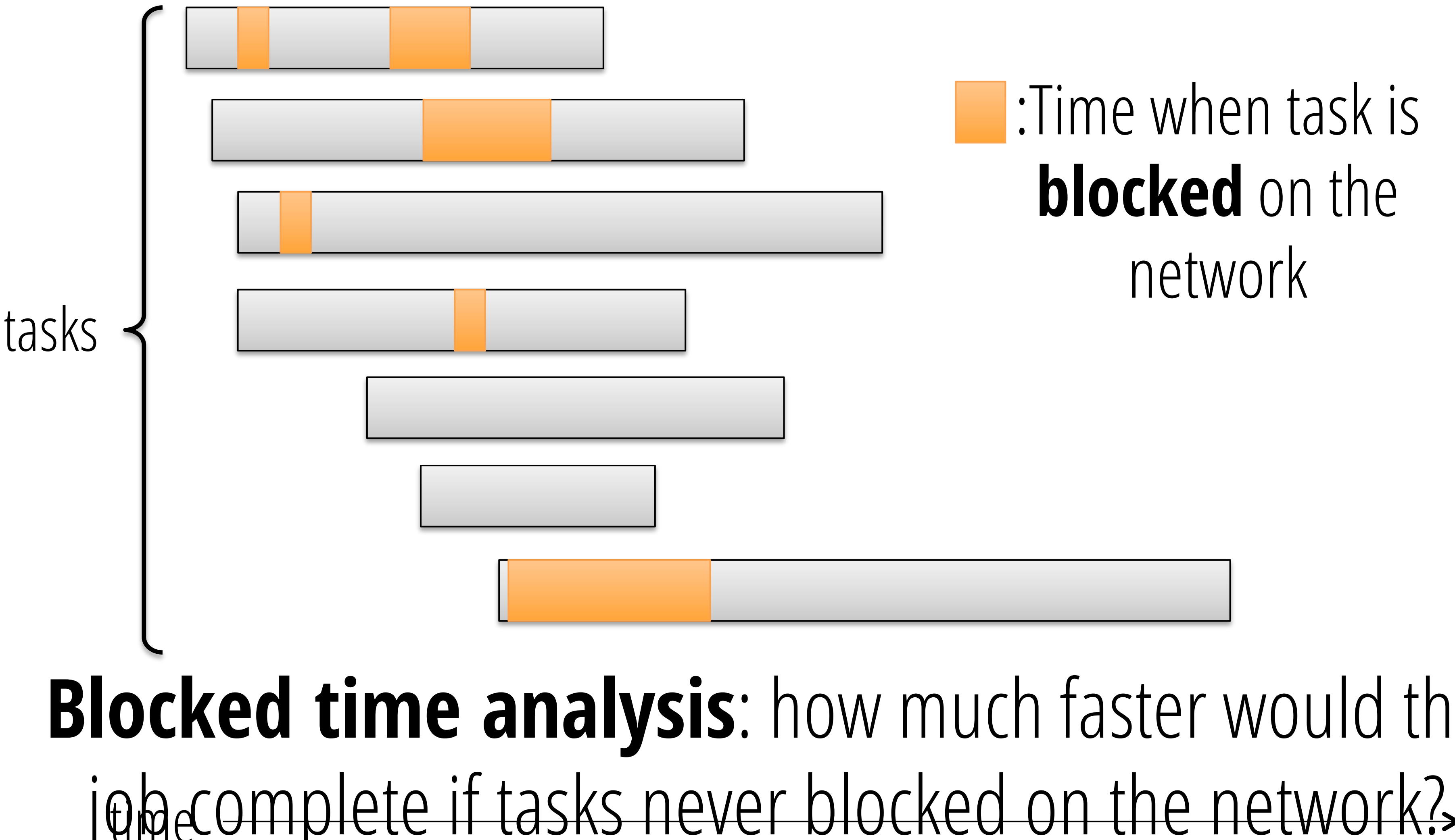
Outline

- **Methodology:** How can we measure bottlenecks?
- **Workloads:** What workloads did we use?
- **Results:** How well do the mantras hold?
- **Why?:** Why do our results differ from past work?

What is the job's bottleneck?

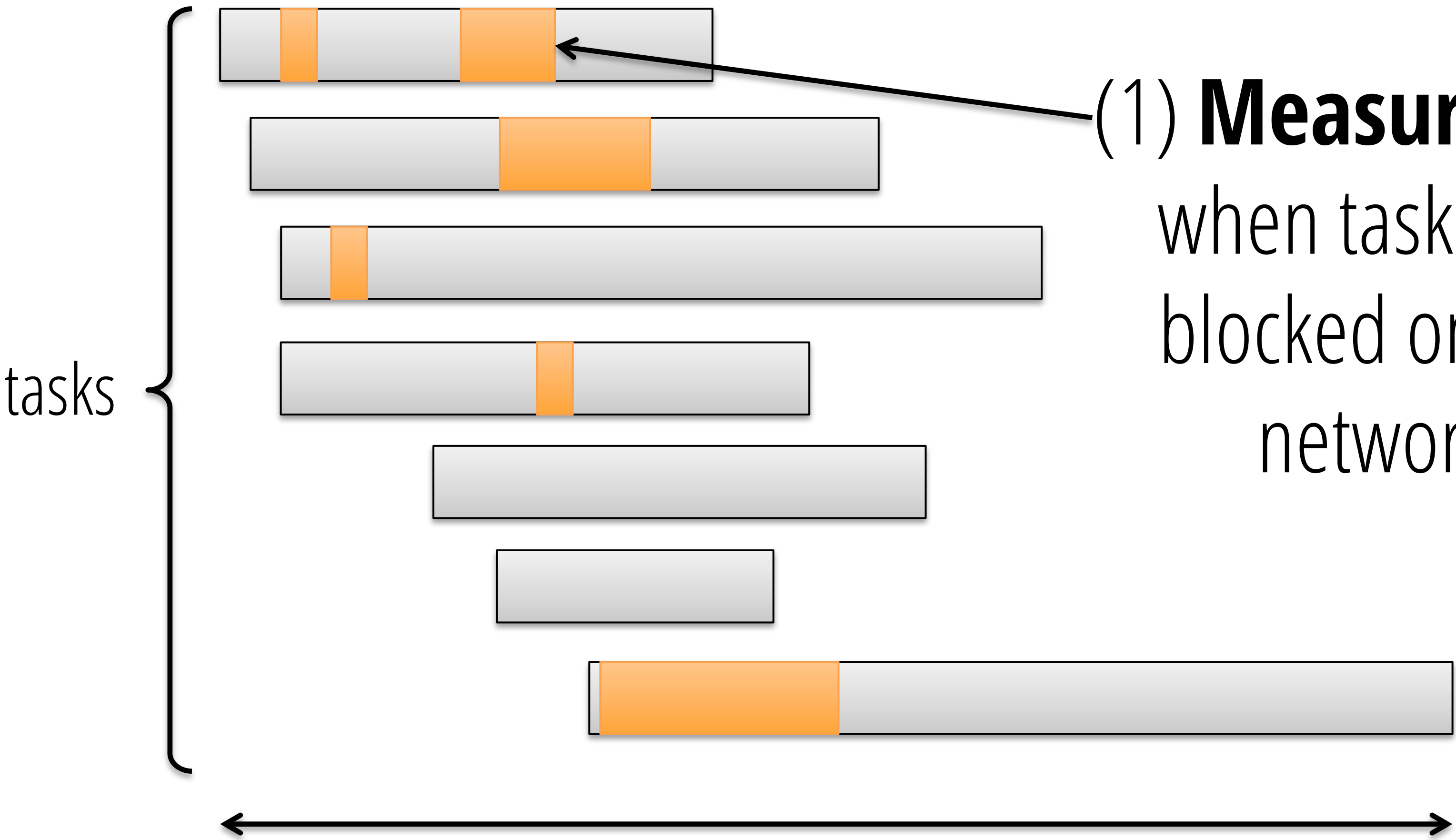


How does network affect the job's completion time?



Blocked time analysis: how much faster would the job complete if tasks never blocked on the network?

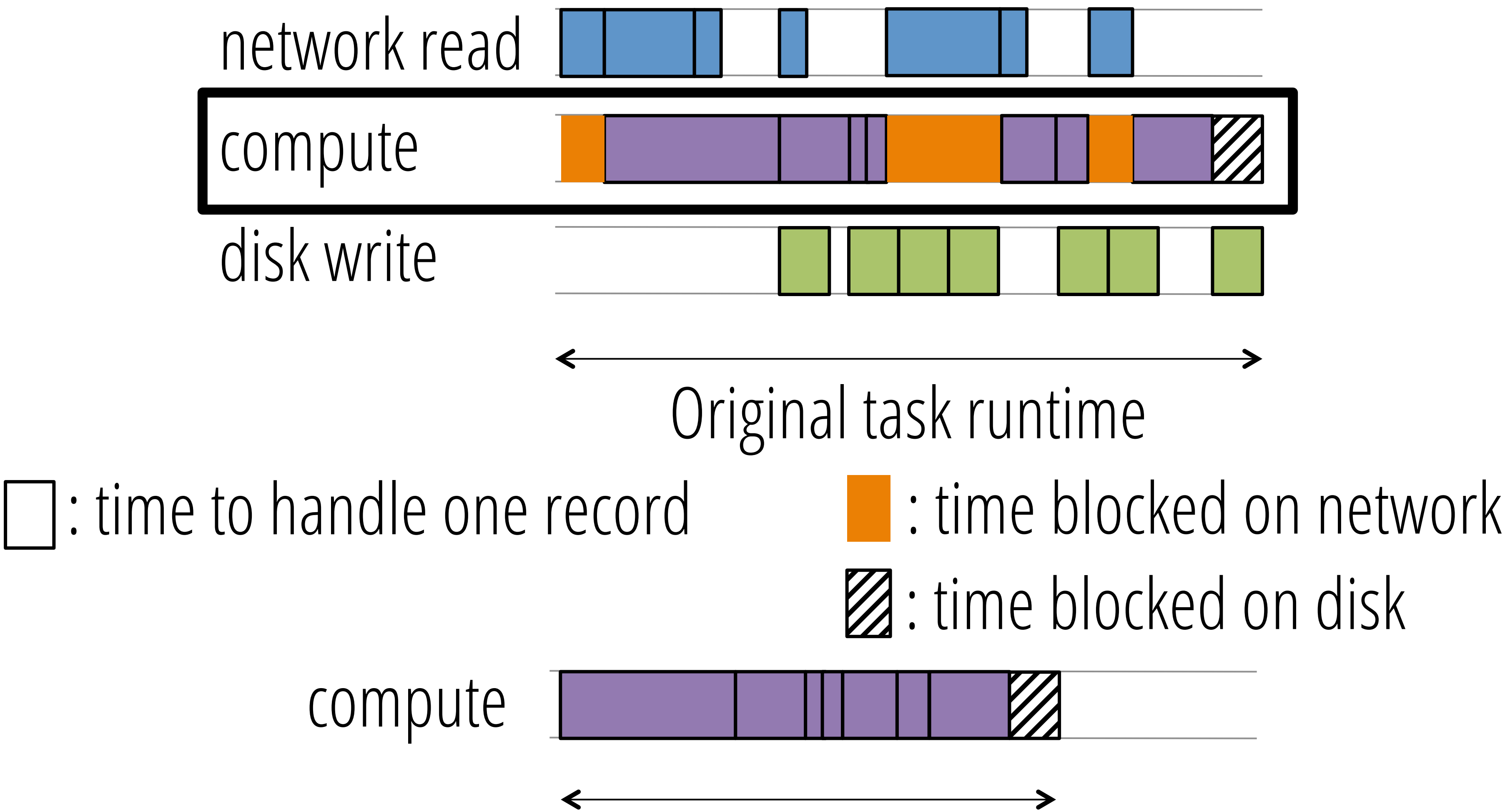
Blocked time analysis



(1) **Measure** time when tasks are blocked on the network

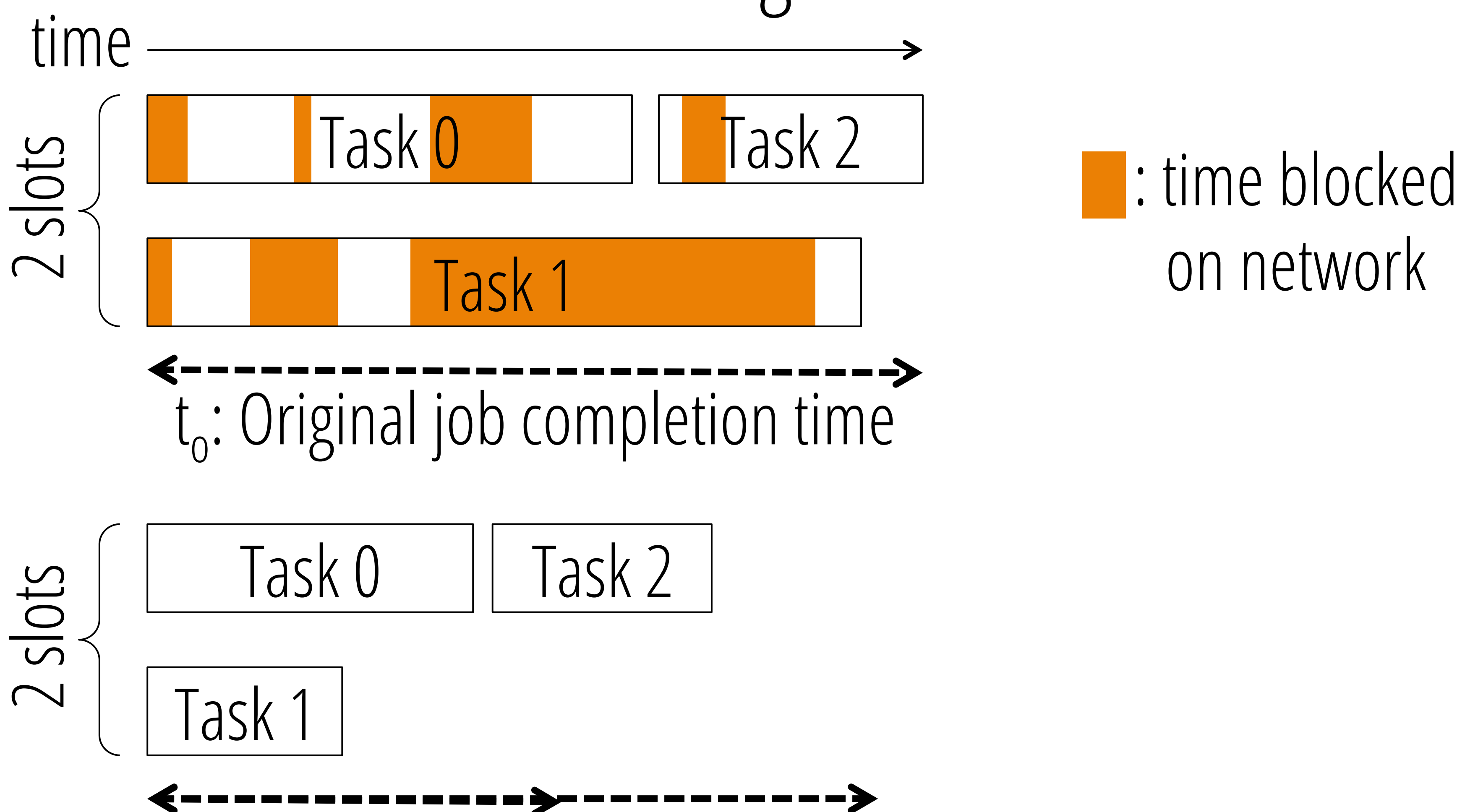
(2) **Simulate** how job completion time would change

(1) **Measure** time when tasks are blocked on network



Best case task runtime if network were infinitely fast

(2) **Simulate** how job completion time would change



t_n: Job completion time with infinitely fast network

Incorrectly computed time: doesn't account for task scheduling

Blocked time analysis: how quickly could a job have completed if a resource were infinitely fast?

Outline

- **Methodology:** How can we measure bottlenecks?
- **Workloads:** What workloads did we use?
- **Results:** How well do the mantras hold?
- **Why?:** Why do our results differ from prior work?

Large-scale traces?

Don't have enough instrumentation for blocked-time analysis

SQL Workloads run on Spark

Only 3 workloads **1 Framework**

TPC-DS (20 machines, 850GB;
60 machines, 2.5TB; 200 machines, 2.5TB)

Big Data Benchmark (5 machines, 60GB)

Databricks (Production; 9 machines, tens of GB)

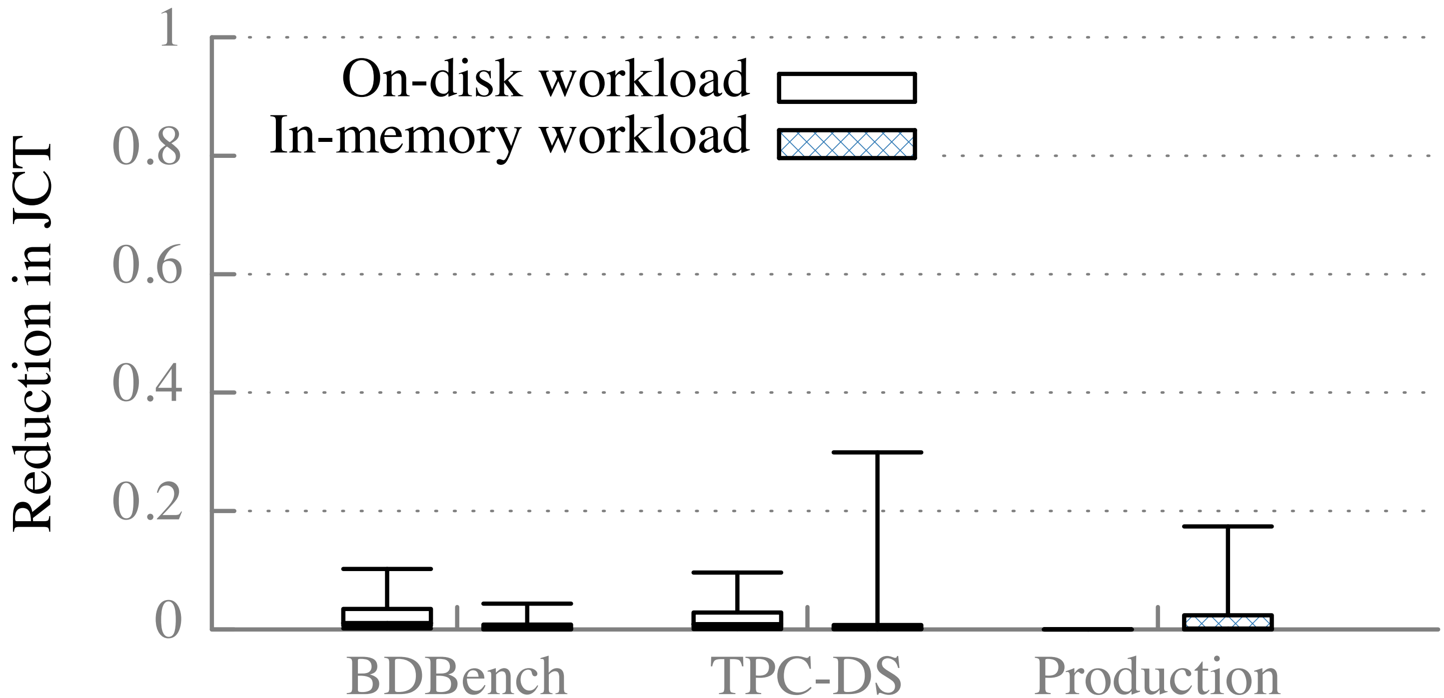
Small cluster sizes

2 versions of each: in-memory, on-disk

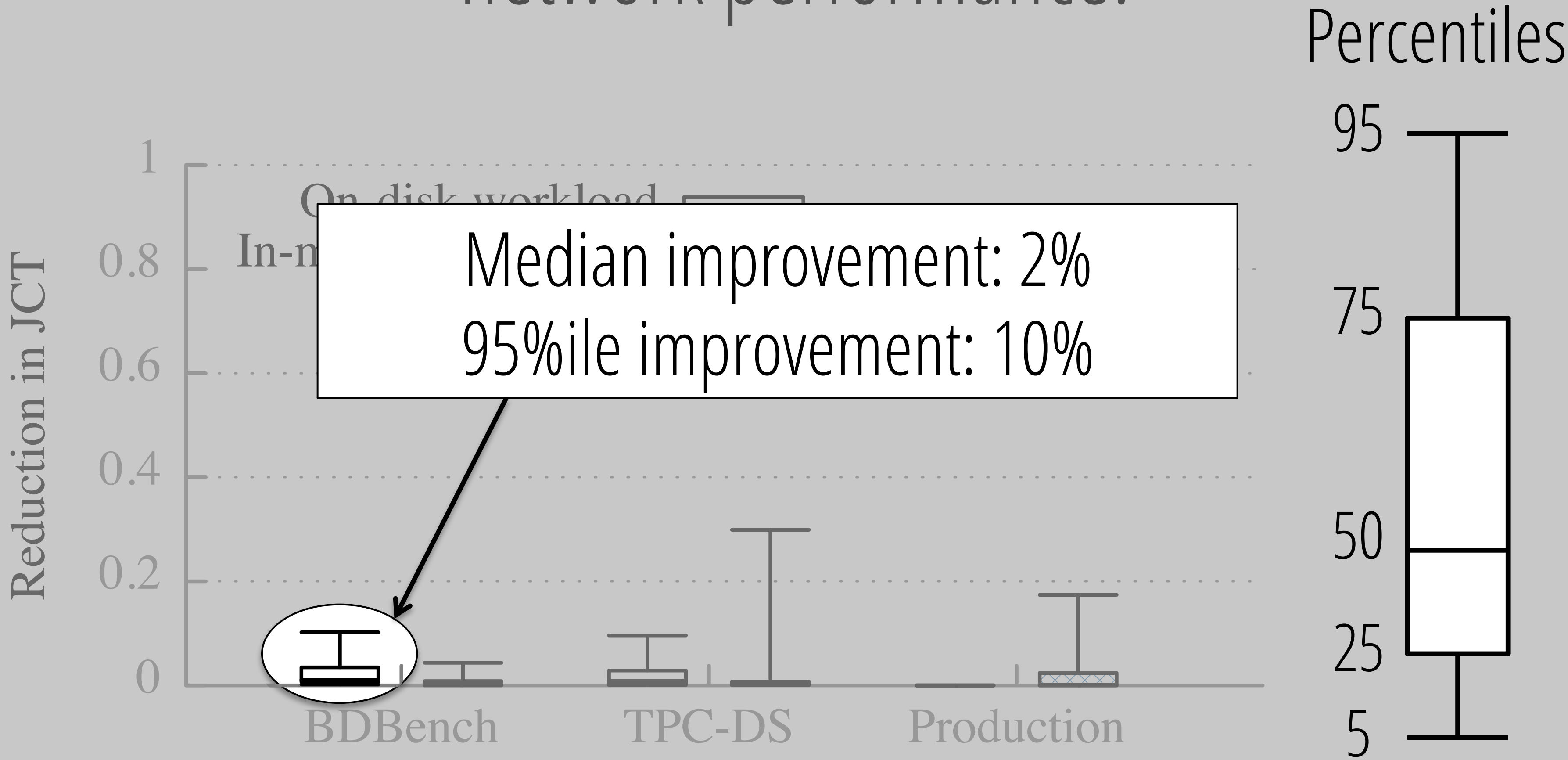
Outline

- **Methodology:** How can we measure bottlenecks?
- **Workloads:** What workloads did we use?
- **Results:** How well do the mantras hold?
- **Why?:** Why do our results differ from prior work?

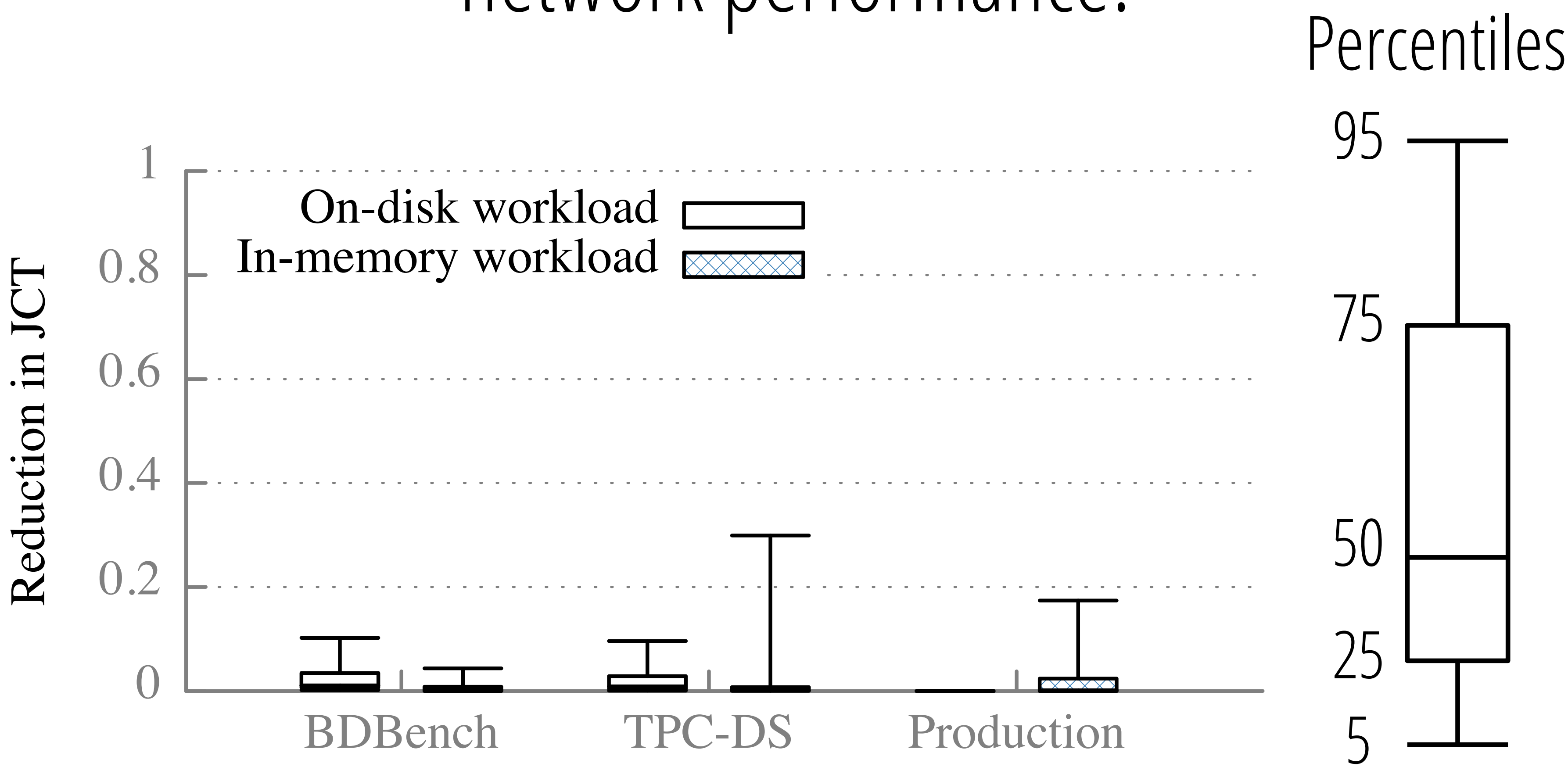
How much faster could jobs get from optimizing network performance?



How much faster could jobs get from optimizing network performance?

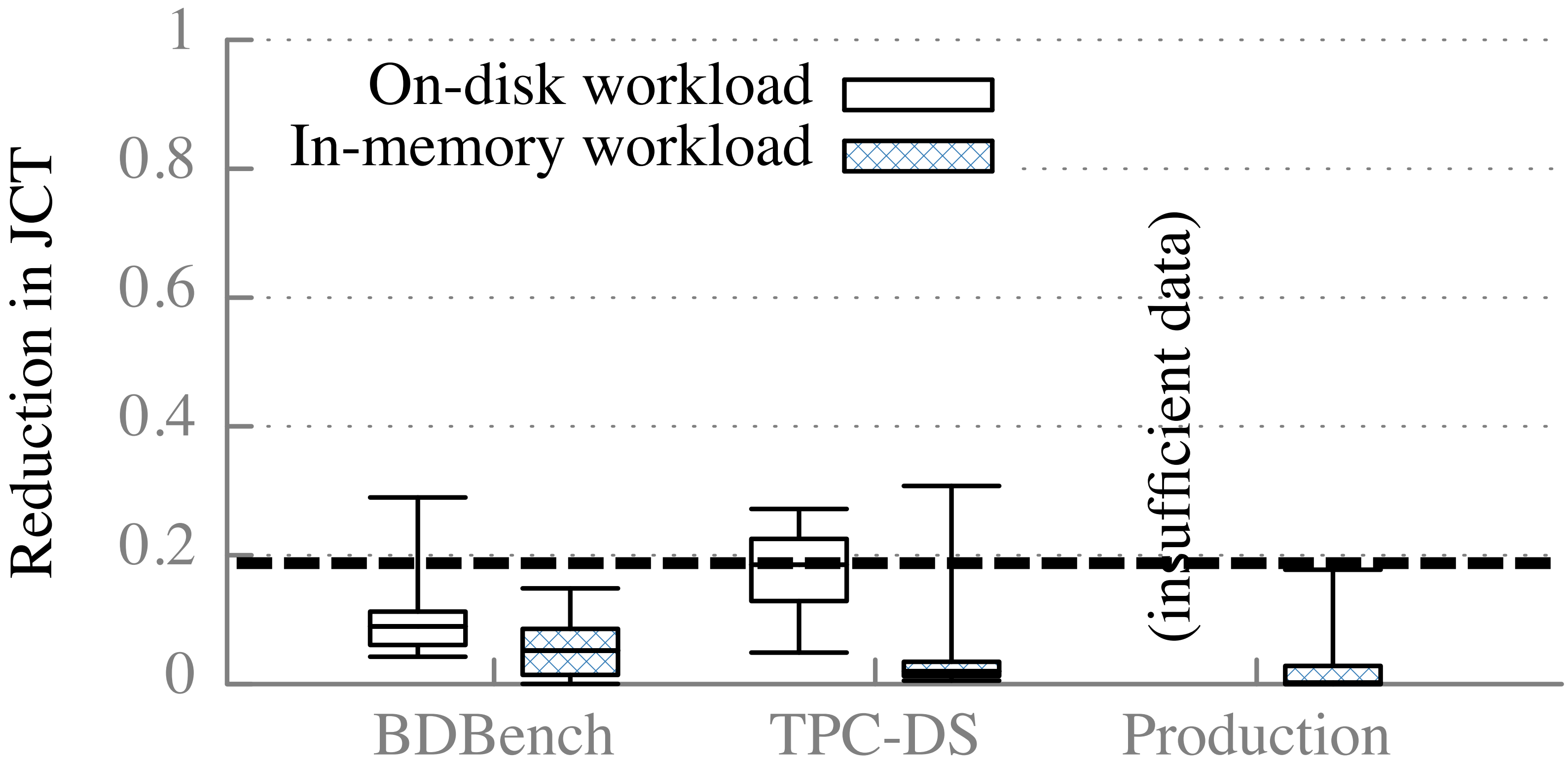


How much faster could jobs get from optimizing network performance?



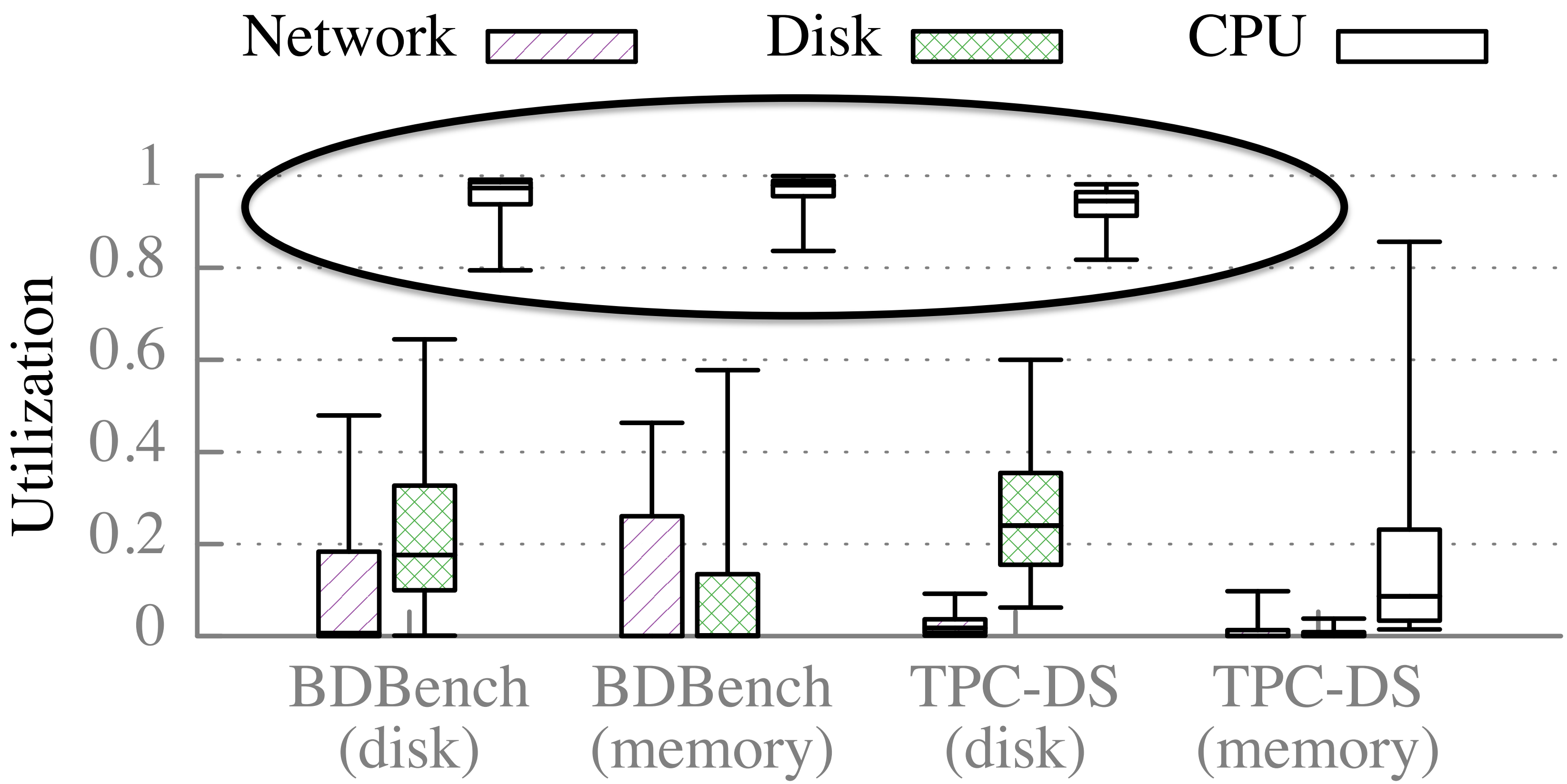
Median improvement at most 2%

How much faster could jobs get from optimizing disk performance?



Median improvement at most 19%

How important is CPU?



CPU much more highly utilized than disk or network!

What about stragglers?

5-10% improvement from eliminating stragglers

- Based on simulation

Can explain >60% of stragglers in >75% of jobs

Fixing underlying cause can speed up other tasks too!

- 2x speedup from fixing one straggler cause

Takeaways based on three Spark workloads:

Network optimizations
can reduce job completion time by **at most 2%**

CPU (not I/O) often the bottleneck
<19% reduction in completion time from optimizing disk

Many straggler causes can be identified and fixed

Outline

- **Methodology:** How can we measure bottlenecks?
- **Workloads:** What workloads did we use?
- **Results:** How well do the mantras hold?
- **Why?:** Why do our results differ from past work?

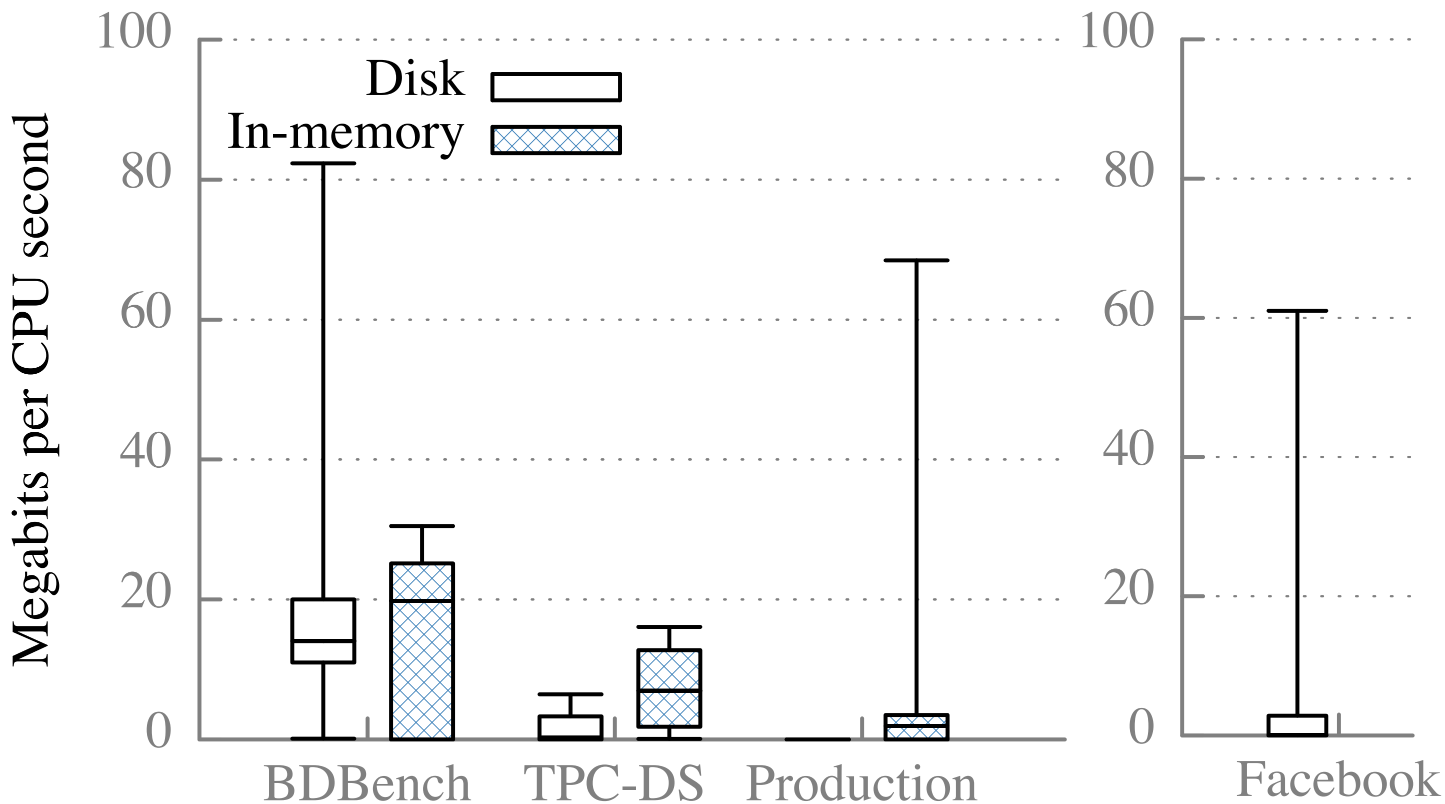
network

Why are our results so different than what's stated in prior work?

Are the workloads we measured unusually network-light?

How can we compare our workloads to large-scale traces used to motivate prior work?

How much data is transferred per CPU second?



Microsoft '09-'10: **1.9–6.35 Mb / task second**

Google '04-'07: **1.34–1.61 Mb / machine second**

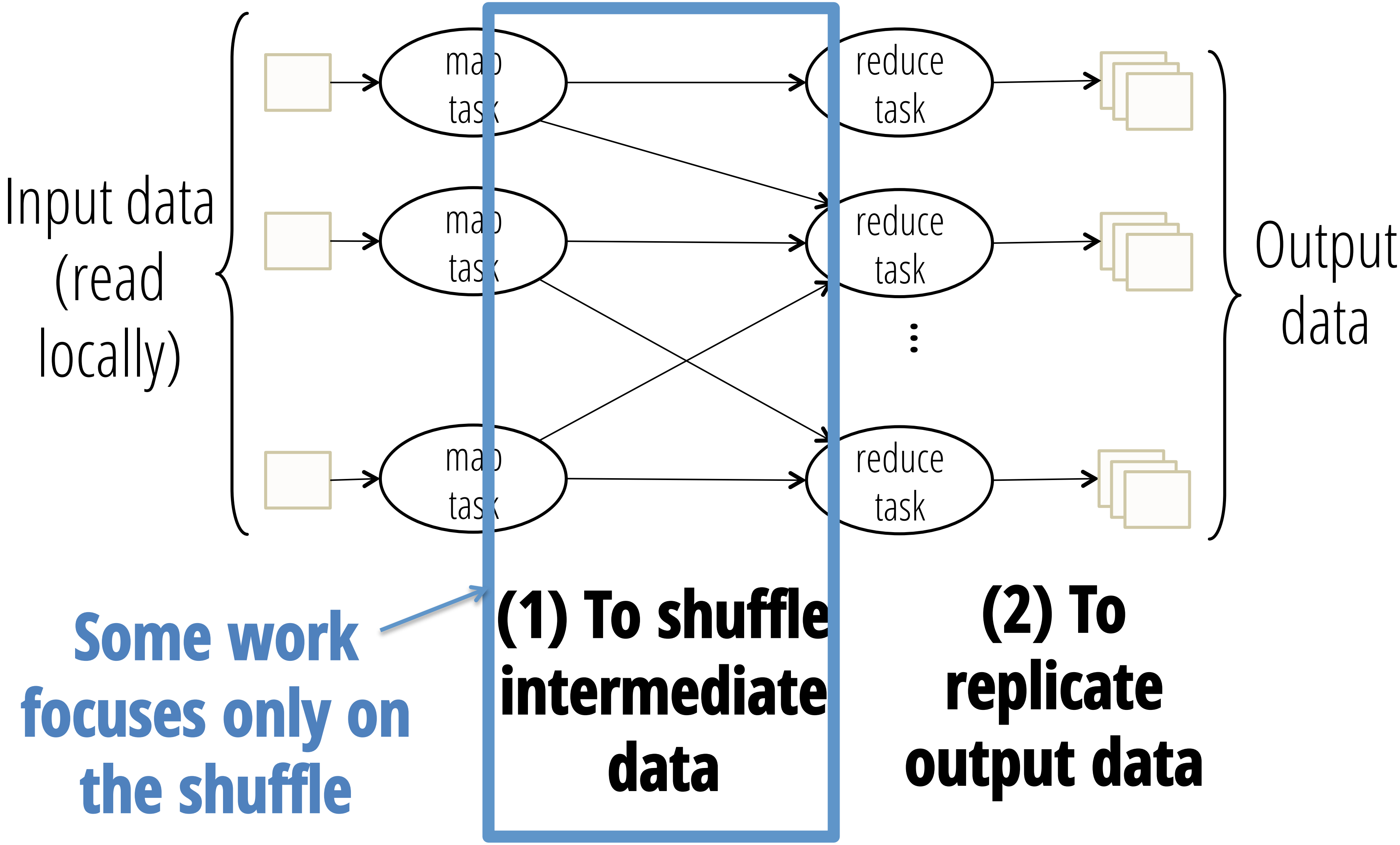
Why are our results so different than what's stated in prior work?

~~Our workloads are network light~~

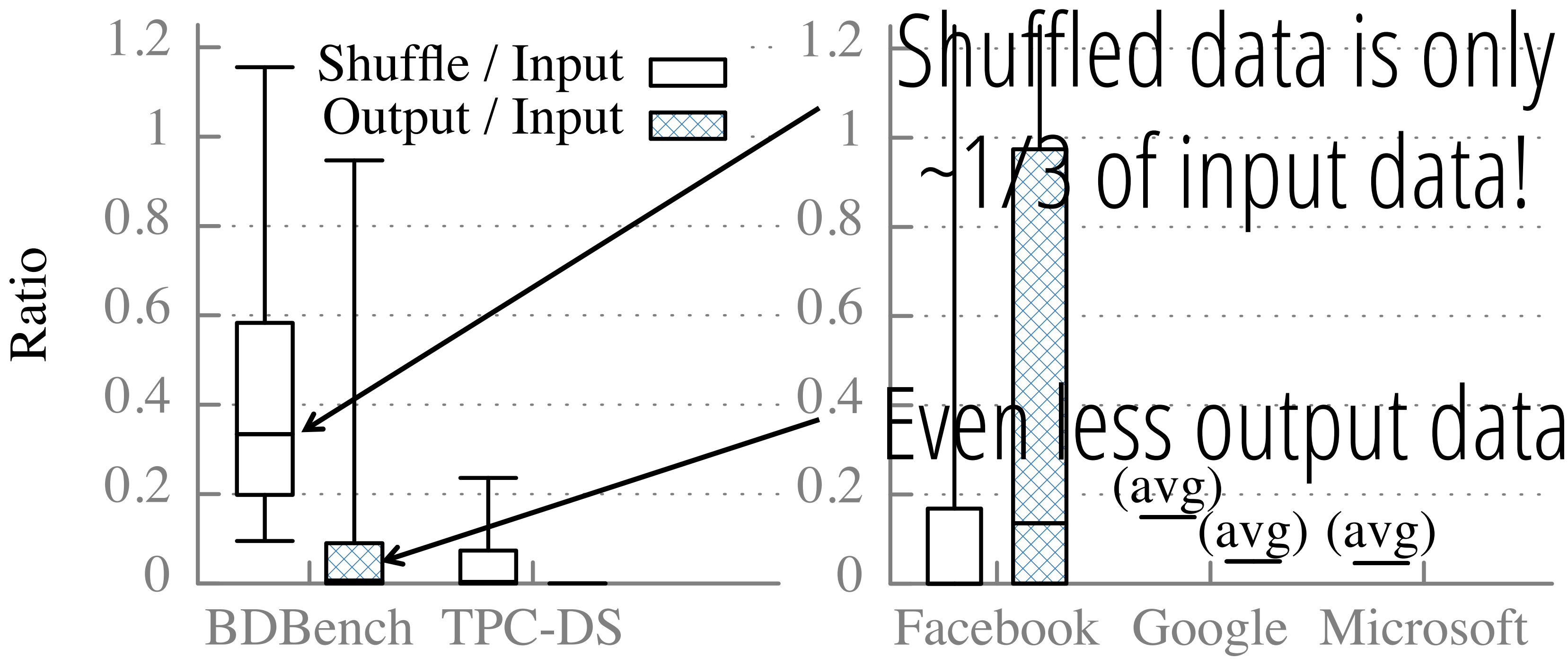
1) Incomplete metrics

2) Conflation of CPU and network time

When is the network used?



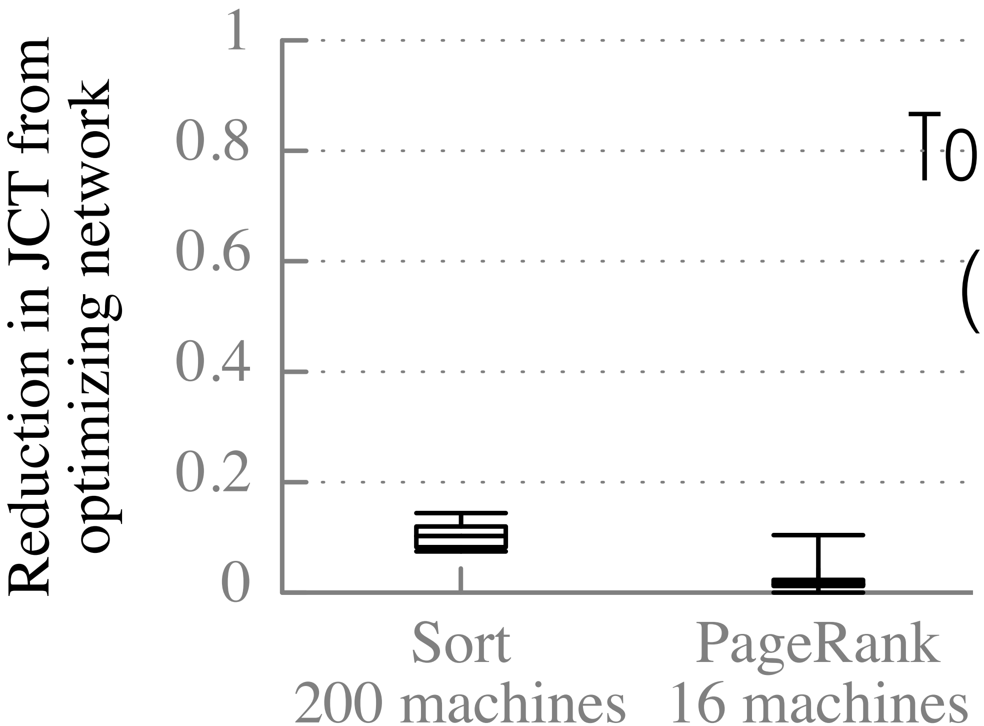
How does the data transferred over the network compare to the input data?



Not realistic to look only at shuffle!

Or to use workloads where all input is shuffled

Prior work conflates CPU and network time



To send data over network:

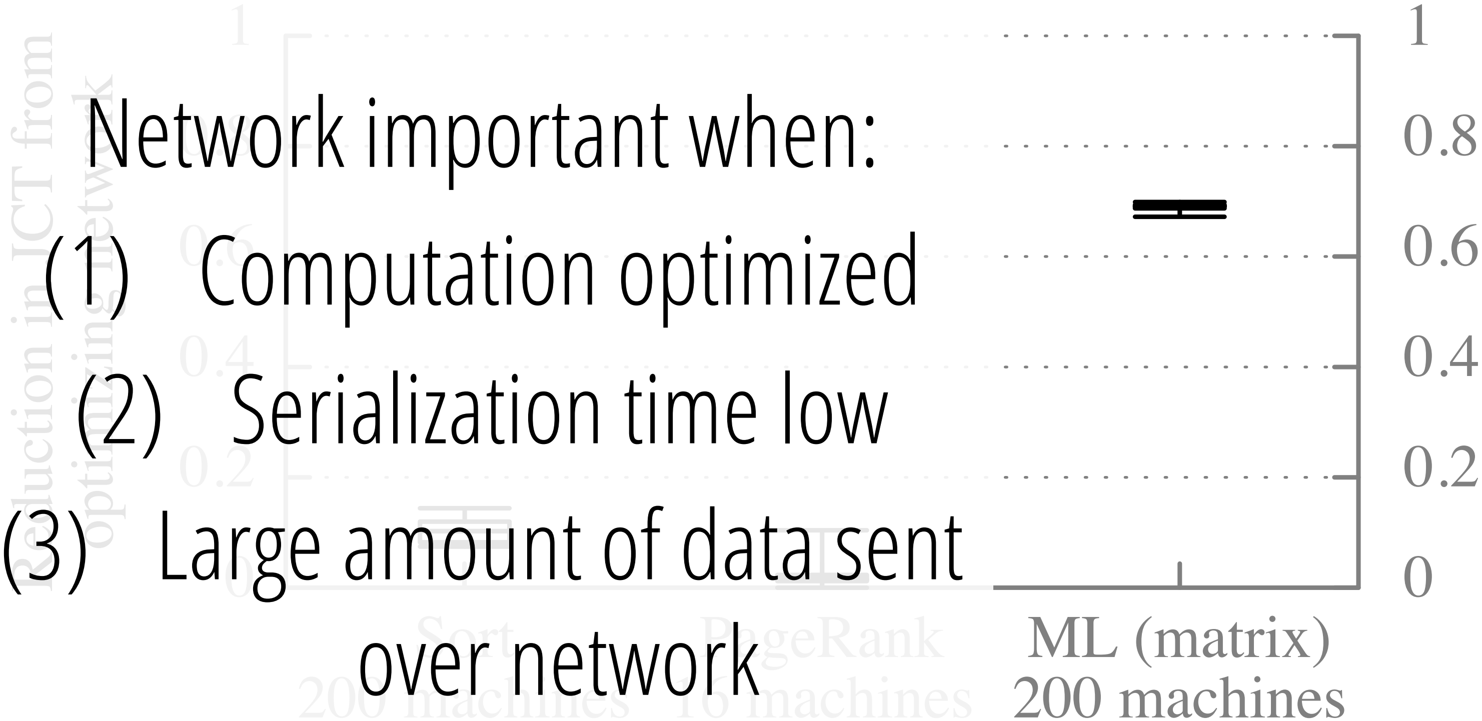
(1) Serialize objects into bytes

(2) Send bytes

(1) and (2) often conflated.

Reducing application data sent reduces both!

When does the network matter?



Network important when:

(1) Computation optimized

(2) Serialization time low

(3) Large amount of data sent over network

Why are our results so different than what's stated in prior work?

~~Our workloads are network light~~

1) Incomplete metrics

e.g., looking only at shuffle time

2) Conflation of CPU and network time

Sending data over the network has an associated CPU cost

Limitations

Only three workloads

Small cluster sizes

One framework (Spark)

Limitations aren’t fatal

Only three workloads

Industry-standard workloads

Results sanity-checked with larger production traces

Small cluster sizes

Takeaways don’t change when we move between cluster sizes

One framework (Spark)

Results sanity-checked with production traces from other frameworks

We instrumented and evaluated Hadoop, with consistent results

Will change with time!

Network optimizations
can reduce job completion time by **at most 2%**

CPU (not I/O) often the bottleneck
<19% reduction in completion time from optimizing disk

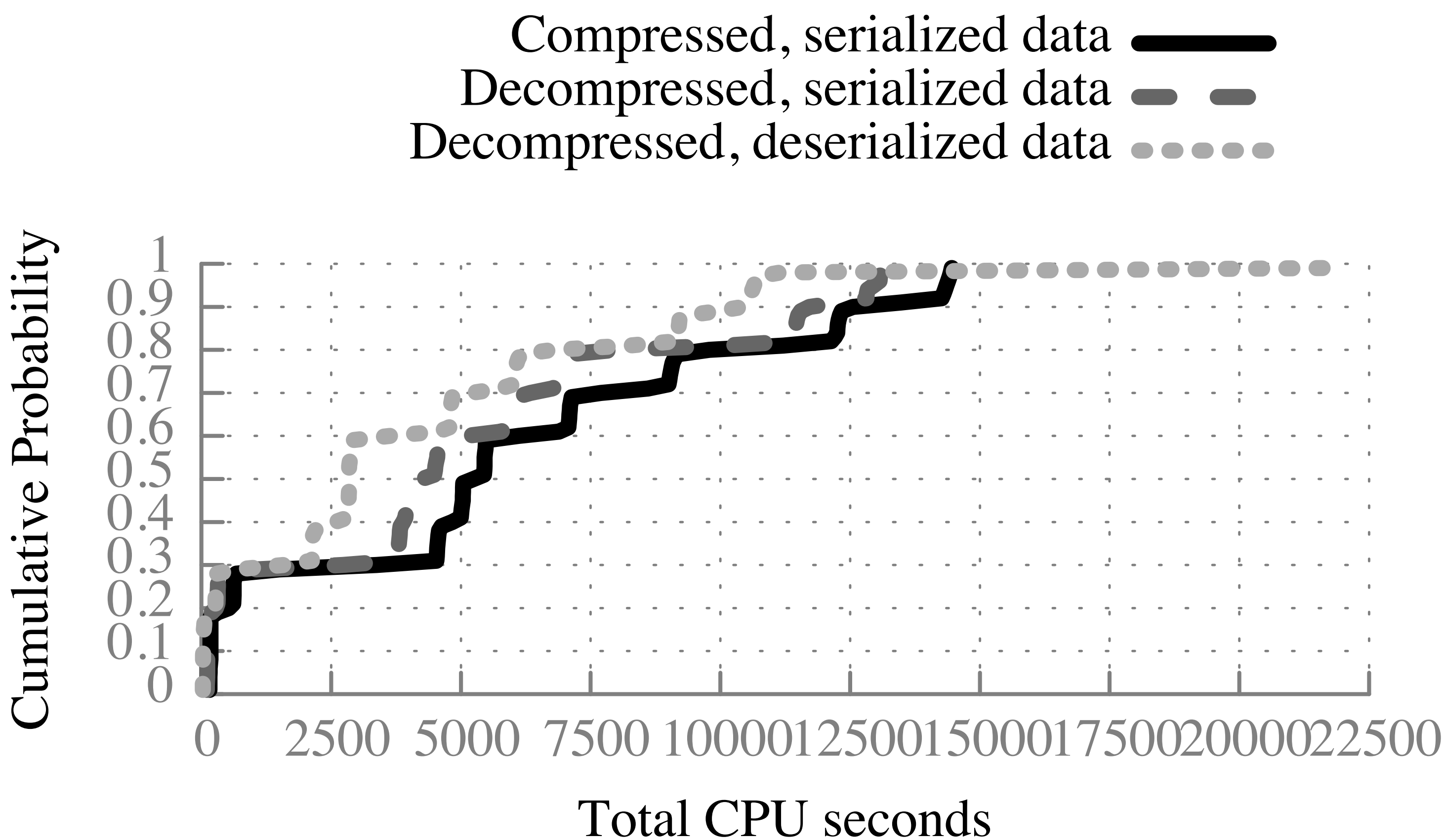
Many straggler causes can be identified and fixed

Takeaway: performance understandability should be a first-class concern!
Instrument systems for blocked time analysis
(almost) All Instrumentation now part of Spark

All traces publicly available: tinyurl.com/nsdi-traces

Backup Slides

Why is the CPU time so high?



Compression and serialization are costly

What can be done to reduce compute time?

Project Tungsten: Bringing Spark Closer to Bare Metal

April 28, 2015 | by Reynold Xin and Josh Rosen

In a previous blog post, we looked back and surveyed performance improvements made to Spark in the past year. In this post, we look forward and share with you the next chapter, which we are calling ***Project Tungsten.*** 2014 witnessed Spark setting the world record in large-scale sorting and saw major improvements across the entire engine from Python to SQL to machine learning. Performance optimization, however, is a never ending process.

Project Tungsten will be the largest change to Spark's execution engine since the project's inception. It focuses on substantially improving the efficiency of ***memory and CPU*** for Spark applications, to push performance closer to the limits of modern hardware. This effort includes three initiatives:

1. ***Memory Management and Binary Processing:*** leveraging application semantics to manage memory explicitly and eliminate the overhead of JVM object model and garbage collection
2. ***Cache-aware computation***: algorithms and data structures to exploit memory hierarchy
3. ***Code generation***: using code generation to exploit modern compilers and CPUs

The focus on CPU efficiency is motivated by the fact that Spark workloads are increasingly bottlenecked by CPU and memory use rather than IO and network communication. This trend is shown by recent research on the performance of big data workloads (Ousterhout et al) and we've arrived at similar findings as part of our ongoing tuning and optimization efforts for Databricks Cloud customers.

Why is CPU the new bottleneck? There are many reasons for this. One is that hardware configurations offer increasingly large aggregate IO bandwidth, such as 10Gbps links in networks and high bandwidth SSD's or striped HDD arrays for storage. From a software perspective, Spark's optimizer now allows many workloads to avoid significant disk IO by pruning input data that is not needed in a given job. In Spark's shuffle subsystem, serialization and hashing (which are CPU bound) have been shown to be key bottlenecks, rather than raw network throughput of underlying hardware. All these trends mean that Spark today is often constrained by CPU efficiency and memory pressure rather than IO.