Online Aggregation for Large MapReduce Jobs

Niketan Pansare[1], Vinayak Borkar[2],
Chris Jermaine[1], Tyson Condie[3]

[1]Rice University, [2]UC Irvine, [3]Yahoo! Research

Outline

- Motivation
- Implementation
- Experiments
- Conclusion

Outline

- Motivation
- Implementation
- Experiments
- Conclusion

OLA example

select avg(stock_price) from nasdaq_db where company = 'xyz';

(Note: final answer for this query is 1000)

OLA example

select avg(stock_price) from nasdaq_db where company = 'xyz';

After 1 second,

- Conventional Database:

- With OLA extension:
  - Output range estimate: [0, 2000] with 95% probability

OLA example

select avg(stock_price) from nasdaq_db where company = 'xyz';

After 2 minutes,

- Conventional Database:

- With OLA extension:
  - Output range estimate: [900, 1100] with 95% probability

OLA example

select avg(stock_price) from nasdaq_db where company = 'xyz';

After 4 minutes,

- Conventional Database:

- With OLA extension:
  - Output range estimate: [950, 1040] with 95% probability

OLA example

select avg(stock_price) from nasdaq_db where company = 'xyz';

After 6 minutes,

- Conventional Database:

- With OLA extension:
  - Output range estimate: [990, 1010] with 95% probability

OLA example

select avg(stock_price) from nasdaq_db where company = 'xyz';

After 10 minutes,

- Conventional Database:

- With OLA extension:
  - Output range estimate: [995, 1005] with 95% probability

OLA example

select avg(stock_price) from nasdaq_db where company = 'xyz';

After 30 minutes,

- Conventional Database:

- With OLA extension:
  - Output range estimate: [999, 1001.5] with 95% probability

OLA example

select avg(stock_price) from nasdaq_db where company = 'xyz';

After 2 hours,

- Conventional Database:
  - Output final answer: 1000
- With OLA extension:
  - Output final answer: 1000

Benefit of OLA

- **If acceptably accurate answer reached quickly, the query can be aborted**

After 6 minutes,

- Conventional Database:

- With OLA extension:
  - Output range estimate: [990, 1015] with 95% probability

STOP EARLY !!!

Why Stop Early ?

- Save human time (1 hour 54 minutes)
  - 'Answer 1000' v/s 'Estimate 1002.5'

    - For exploratory apps

    - Inaccuracies in ETL process

Why Stop Early ?

- Save human time (1 hour 54 minutes)
  - 'Answer 1000' v/s 'Estimate 1002.5'

    - For exploratory apps

    - Inaccuracies in ETL process
- Save machine time → Cost ↓

Why Stop Early ?

- Save human time (1 hour 54 minutes)
    - 'Answer 1000' v/s 'Estimate 1002.5'

      - For exploratory apps

      - Inaccuracies in ETL process
- Save machine time → Cost ↓
- Very important when dealing with large data

Why Stop Early ?

- Sa
  -
- Sa
- Ve

Online Aggregation
- Introduced in 1997
- Significant research impact (606 citations)
- ACM SIGMOD Test of Time Award

But, limited commercial impact
- Database market (self-managed)

Self-managed DB → Cloud

- Cost model
  - In Self-managed DB: costs are fixed
  - In Cloud: You pay for amount of hardware used

    - Less resources → Less cost

    - 10 node cluster: 1h 54min → save $12.92/query on EC2
  - User needs to justify the cost to the organization

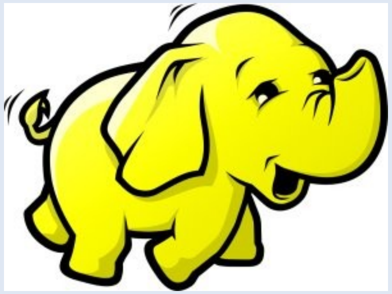
Self-managed DB → Cloud

- Cost model
  - In Self-managed DB: costs are fixed
  - In Cloud: You pay for amount of hardware used

    - Less resources → Less cost

    - 10 node cluster: 1h 54min → save $12.92/query on EC2
  - User needs to justify the cost to the organization

- Modifiying engine to support randomization
  - Traditional DB: Notoriously difficult
  - Cloud: Much simpler



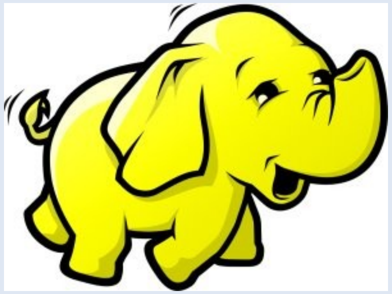
Self-managed DB → Cloud

- Cost model
  - In Self-managed DB: costs are fixed
  - In Cloud: You pay for amount of hardware used

    - Less resources → Less cost

    - 10 node cluster: 1h 54min → save $12.92/query on EC2
  - User needs to justify the cost to the organization

- Modifiying engine to support randomization
  - Traditional DB: Notoriously difficult
  - Cloud: Much simpler



- Therefore, OLA for cloud is an interesting problem

Extend existing approaches

- OLA over single machine
- OLA over multiple machine
- Why it won't work ?
- How do we deal with those issues ?

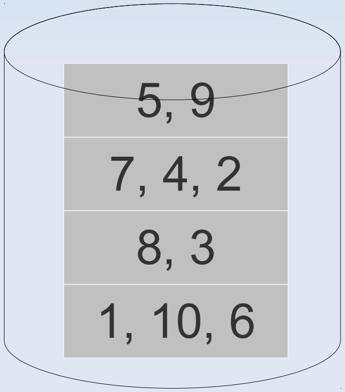
Extend existing approaches

- OLA over single machine
  - Confidence interval found using classical sampling theory
  - Tuples are bundled into blocks
  - Blocks arrive in random order
- OLA over multiple machines
- Why it won't work ?
- How do we deal with those issues ?

OLA over single machine

- Confidence interval found using classical sampling theory
- Tuples are bundled into blocks
- Blocks arrive in random order

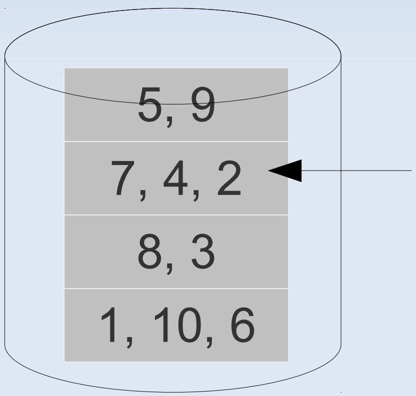
- Example: Find SUM of below values



Note: True answer = 55

OLA over single machine

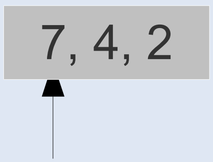
- Confidence interval found using classical sampling theory
- Tuples are bundled into blocks
- Blocks arrive in random order
- Example: Find SUM of below values

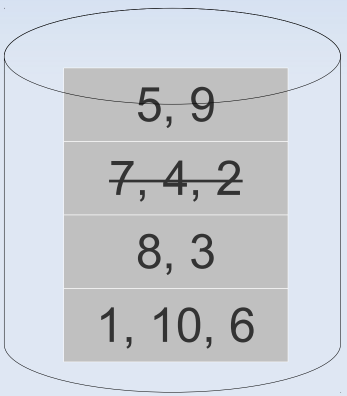


OLA over single machine

- Confidence interval found using classical sampling theory
- Tuples are bundled into blocks
- Blocks arrive in random order

- Example: Find SUM of below values





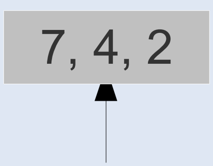
Sample = {}

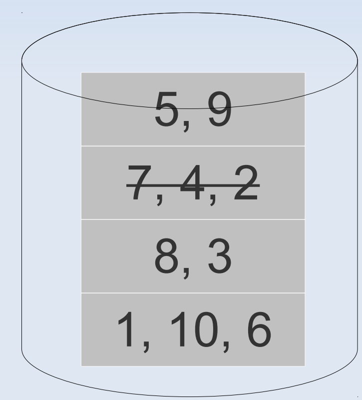
Estimate = Not available

OLA over single machine

- Confidence interval found using classical sampling theory
- Tuples are bundled into blocks
- Blocks arrive in random order

- Example: Find SUM of below values



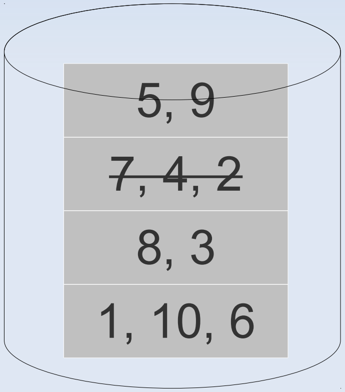


Sample = {}

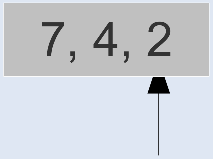
Estimate = Not available

OLA over single machine

- Confidence interval found using classical sampling theory
- Tuples are bundled into blocks
- Blocks arrive in random order



- Example: Find SUM of below values



Sample = {}

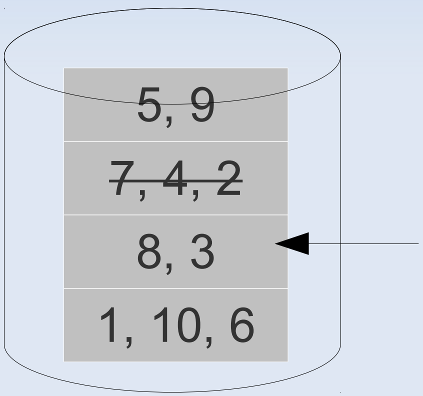
Estimate = Not available

OLA over single machine

- Confidence interval found using classical sampling theory
- Tuples are bundled into blocks
- Blocks arrive in random order

- Example: Find SUM of below values

7, 4, 2



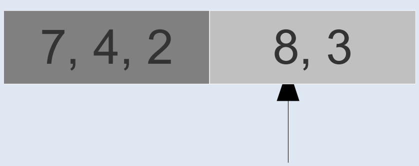
Sample = {13}

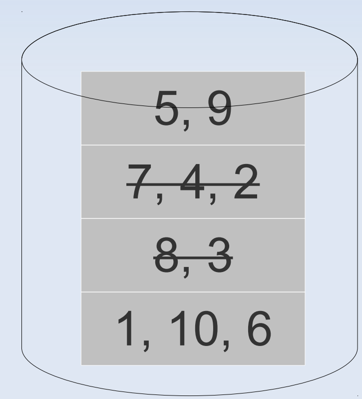
Estimate = 13 * 4 / 1 = 52

OLA over single machine

- Confidence interval found using classical sampling theory
- Tuples are bundled into blocks
- Blocks arrive in random order

- Example: Find SUM of below values





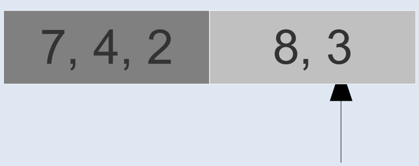
Sample = {13}

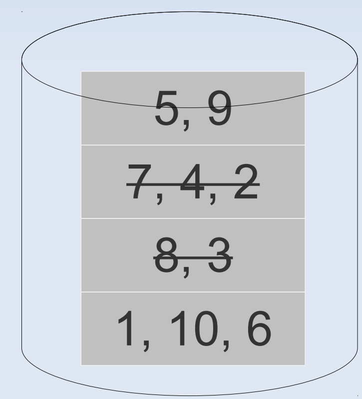
Estimate = 13 * 4 / 1 = 52

OLA over single machine

- Confidence interval found using classical sampling theory
- Tuples are bundled into blocks
- Blocks arrive in random order

- Example: Find SUM of below values





Sample = {13}

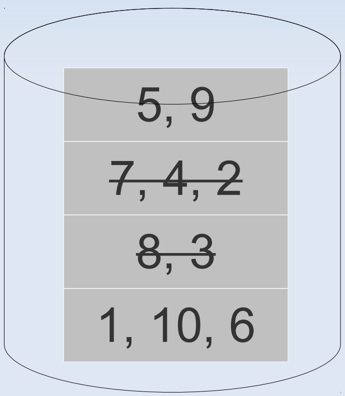
Estimate = 13 * 4 / 1 = 52

OLA over single machine

- Confidence interval found using classical sampling theory
- Tuples are bundled into blocks
- Blocks arrive in random order

- Example: Find SUM of below values

7, 4, 2	8, 3



Sample = {13, 11}

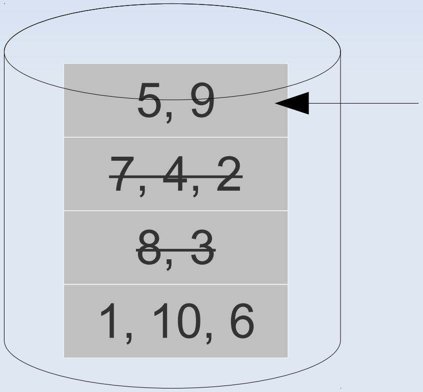
Estimate = (13 + 11) * 4 / 2 = 48

OLA over single machine

- Confidence interval found using classical sampling theory
- Tuples are bundled into blocks
- Blocks arrive in random order

- Example: Find SUM of below values

7, 4, 2	8, 3



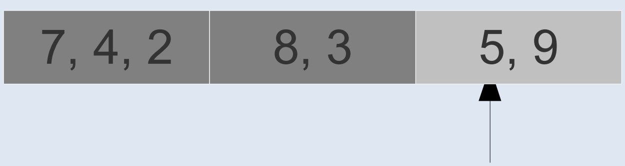
Sample = {13, 11}

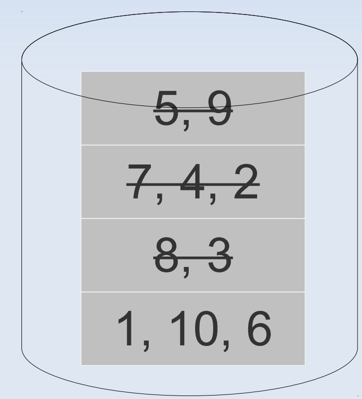
Estimate = (13 + 11) * 4 / 2 = 48

OLA over single machine

- Confidence interval found using classical sampling theory
- Tuples are bundled into blocks
- Blocks arrive in random order

- Example: Find SUM of below values





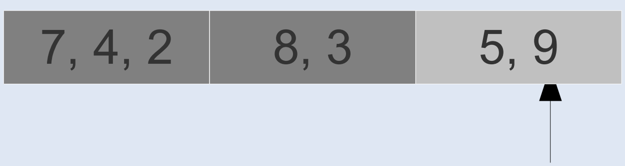
Sample = {13, 11}

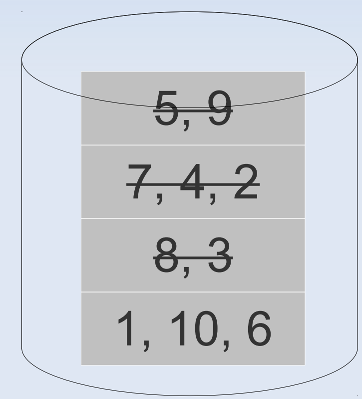
Estimate = (13 + 11) * 4 / 2 = 48

OLA over single machine

- Confidence interval found using classical sampling theory
- Tuples are bundled into blocks
- Blocks arrive in random order

- Example: Find SUM of below values





Sample = {13, 11}

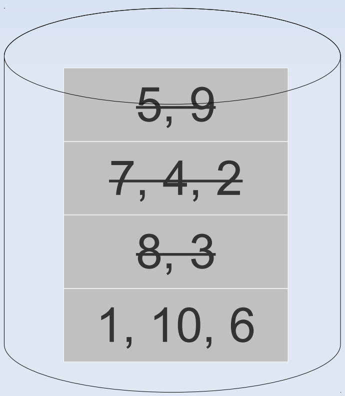
Estimate = (13 + 11) * 4 / 2 = 48

OLA over single machine

- Confidence interval found using classical sampling theory
- Tuples are bundled into blocks
- Blocks arrive in random order

- Example: Find SUM of below values

7, 4, 2	8, 3	5, 9



Sample = {13, 11, 14}

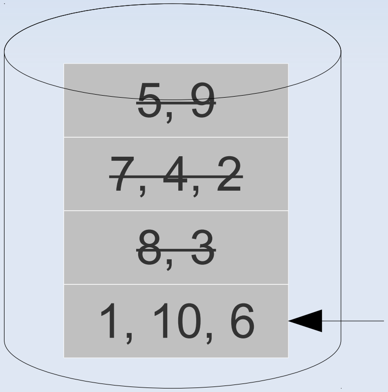
Estimate = (13 + 11 + 14) * 4 / 3 = 50.67

OLA over single machine

- Confidence interval found using classical sampling theory
- Tuples are bundled into blocks
- Blocks arrive in random order

- Example: Find SUM of below values

7, 4, 2	8, 3	5, 9	1, 10, 6

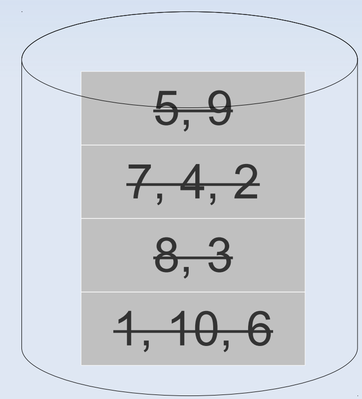


Sample = {13, 11, 14}

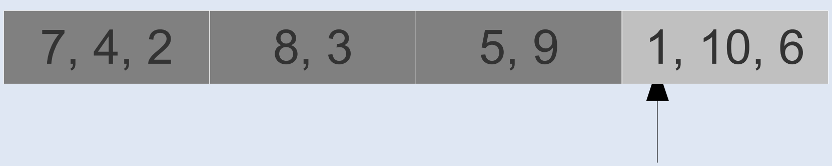
Estimate = (13 + 11 + 14) * 4 / 3 = 50.67

OLA over single machine

- Confidence interval found using classical sampling theory
- Tuples are bundled into blocks
- Blocks arrive in random order



- Example: Find SUM of below values



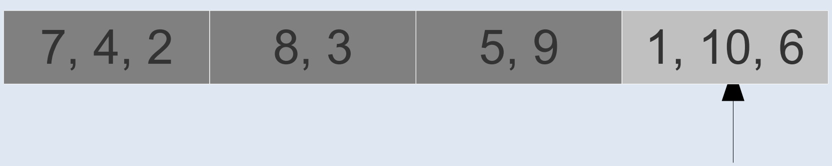
Sample = {13, 11, 14}

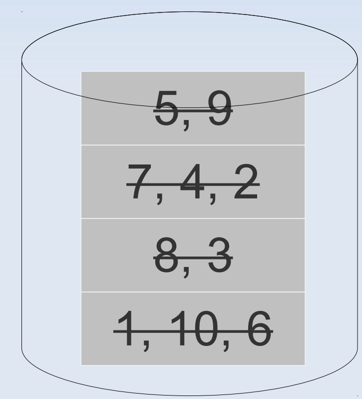
Estimate = (13 + 11 + 14) * 4 / 3 = 50.67

OLA over single machine

- Confidence interval found using classical sampling theory
- Tuples are bundled into blocks
- Blocks arrive in random order

- Example: Find SUM of below values





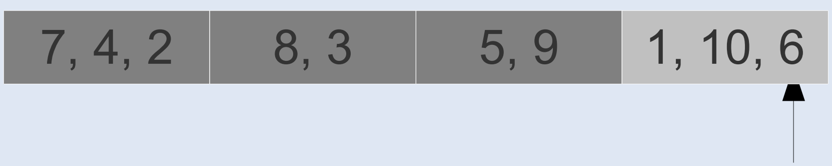
Sample = {13, 11, 14}

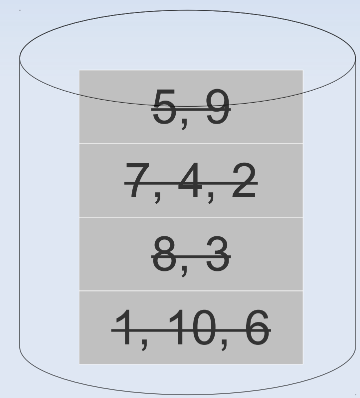
Estimate = (13 + 11 + 14) * 4 / 3 = 50.67

OLA over single machine

- Confidence interval found using classical sampling theory
- Tuples are bundled into blocks
- Blocks arrive in random order

- Example: Find SUM of below values



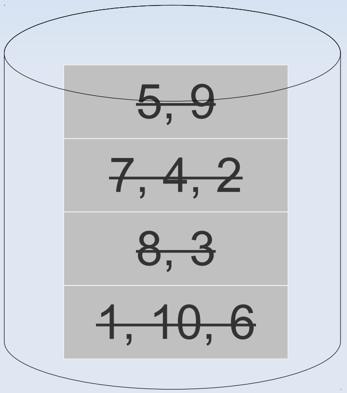


Sample = {13, 11, 14}

Estimate = (13 + 11 + 14) * 4 / 3 = 50.67

OLA over single machine

- Confidence interval found using classical sampling theory
- Tuples are bundled into blocks
- Blocks arrive in random order



- Example: Find SUM of below values

7, 4, 2	8, 3	5, 9	1, 10, 6

Sample = {13, 11, 14, 17}

Estimate = (13 + 11 + 14 + 17) * 4 / 4 = 55

Extend existing approaches

- OLA over single machine
  - Confidence interval found using classical sampling theory
  - Tuples are bundled into blocks
  - Blocks arrive in random order
- OLA over multiple machines
  - Blocks → Non-uniform → Size, Locality, Machine, Network
  - Processing time for block can be large and highly variable
- Why it won't work ?
- How do we deal with those issues ?

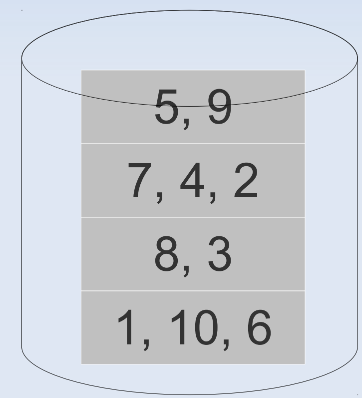
OLA over multiple machines

- Blocks → Non-uniform → Size, Locality, Machine, Network
- Processing time for block can be large and highly variable

So, instead of

- Example: Find SUM of below values

7, 4, 2	8, 3	5, 9	1, 10, 6

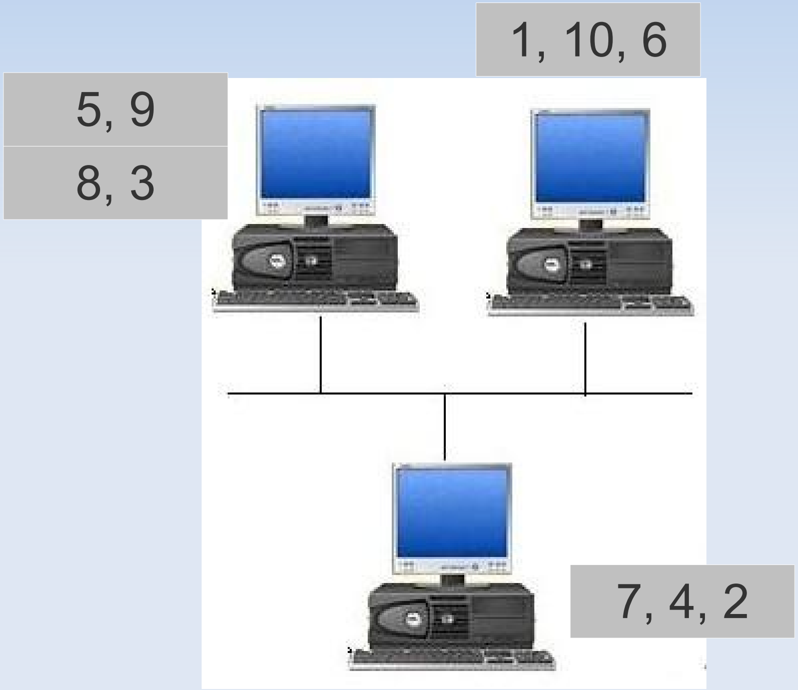


OLA over multiple machines

- Blocks → Non-uniform → Size, Locality, Machine, Network
- Processing time for block can be large and highly variable
- Example: Find SUM of below values

7, 4, 2	8, 3	5, 9	1, 10, 6

X axis = Processing Time



OLA over multiple machines

- Blocks → Non-uniform → Size, Locality, Machine, Network
- Processing time for block can be large and highly variable
- Example: Find SUM of below values

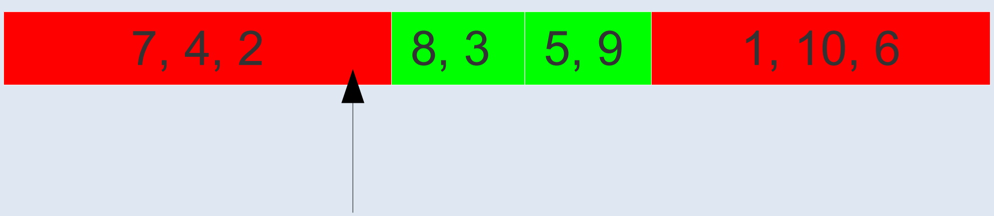
7, 4, 2	8, 3	5, 9	1, 10, 6

- Blocks that take
  - long time to process = RED
  - Short time to process = Green

OLA over multiple machines

- Blocks → Non-uniform → Size, Locality, Machine, Network
- Processing time for block can be large and highly variable

- Example: Find SUM of below values

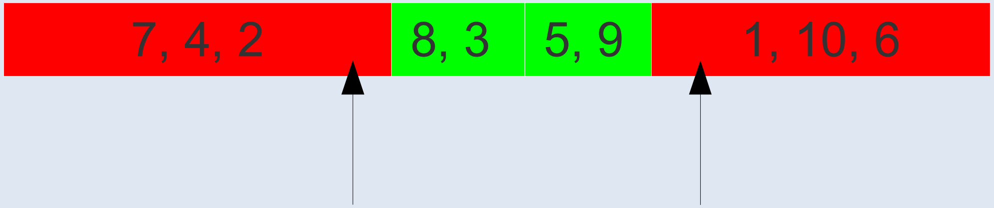


Arrows = Random Time Instances (Polling blocks)

OLA over multiple machines

- Blocks → Non-uniform → Size, Locality, Machine, Network
- Processing time for block can be large and highly variable

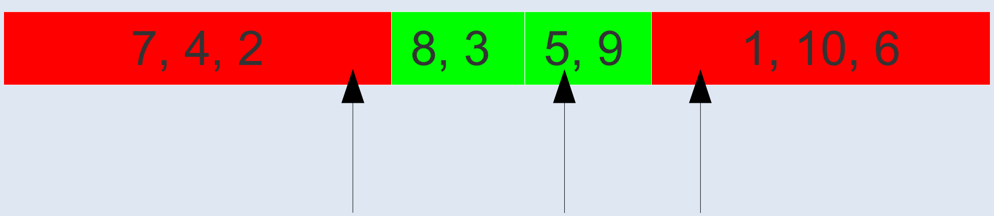
- Example: Find SUM of below values



OLA over multiple machines

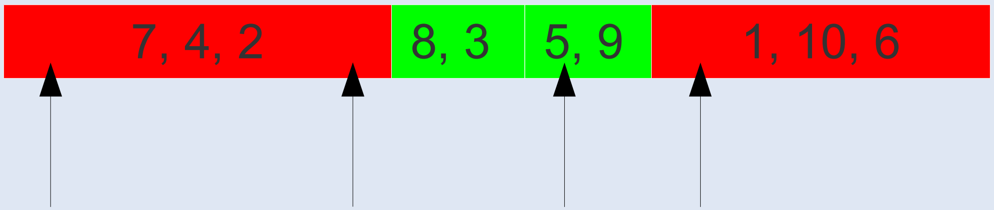
- Blocks → Non-uniform → Size, Locality, Machine, Network
- Processing time for block can be large and highly variable

- Example: Find SUM of below values



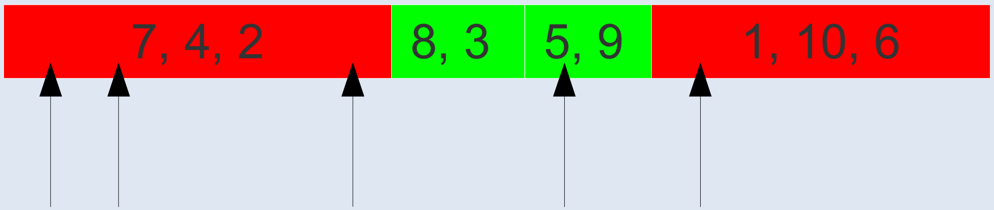
OLA over multiple machines

- Blocks → Non-uniform → Size, Locality, Machine, Network
- Processing time for block can be large and highly variable
- Example: Find SUM of below values



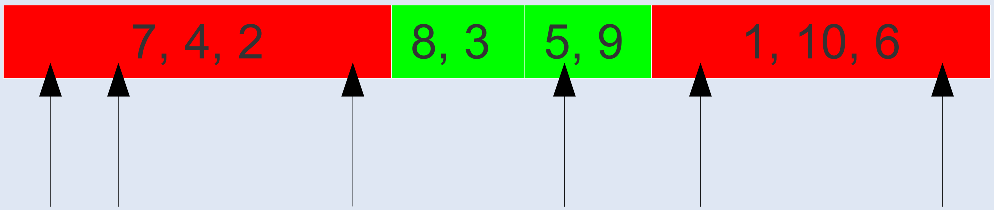
OLA over multiple machines

- Blocks → Non-uniform → Size, Locality, Machine, Network
- Processing time for block can be large and highly variable
- Example: Find SUM of below values



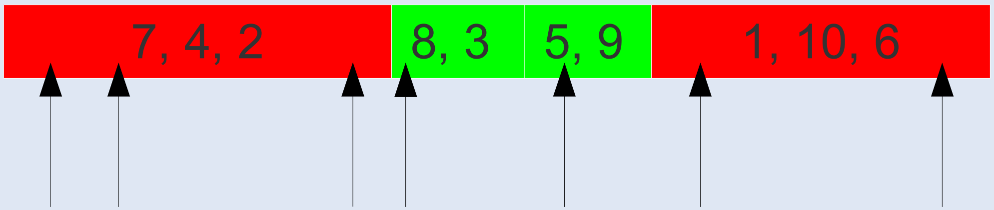
OLA over multiple machines

- Blocks → Non-uniform → Size, Locality, Machine, Network
- Processing time for block can be large and highly variable
- Example: Find SUM of below values



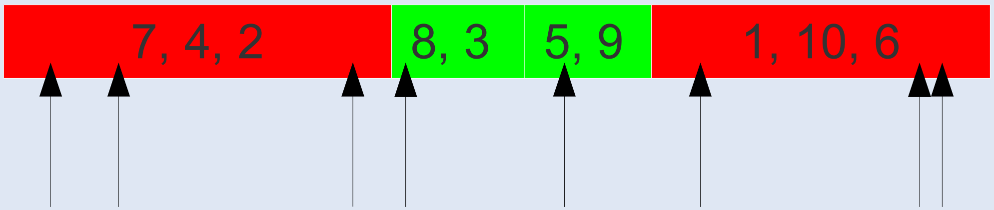
OLA over multiple machines

- Blocks → Non-uniform → Size, Locality, Machine, Network
- Processing time for block can be large and highly variable
- Example: Find SUM of below values



OLA over multiple machines

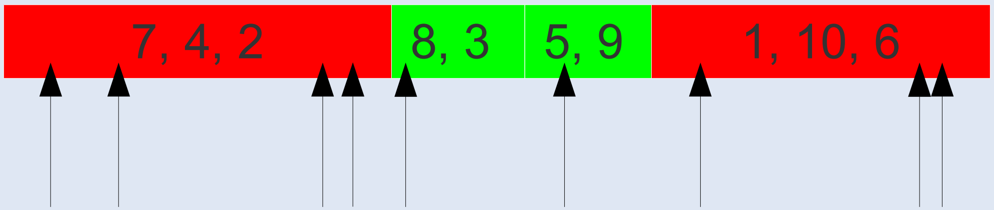
- Blocks → Non-uniform → Size, Locality, Machine, Network
- Processing time for block can be large and highly variable
- Example: Find SUM of below values



OLA over multiple machines

- Blocks → Non-uniform → Size, Locality, Machine, Network
- Processing time for block can be large and highly variable

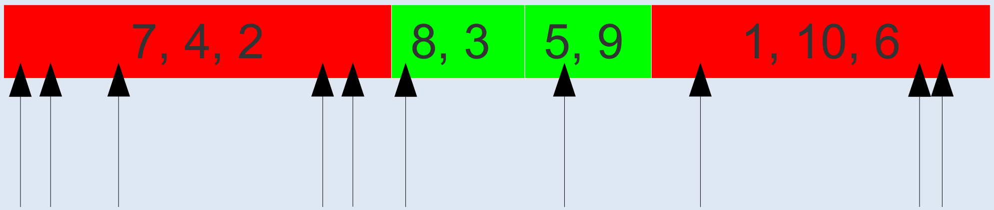
- Example: Find SUM of below values



OLA over multiple machines

- Blocks → Non-uniform → Size, Locality, Machine, Network
- Processing time for block can be large and highly variable

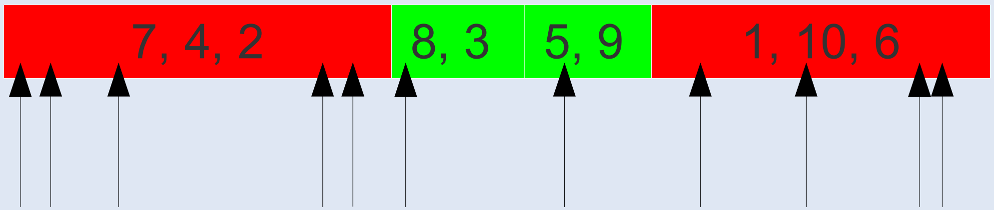
- Example: Find SUM of below values



OLA over multiple machines

- Blocks → Non-uniform → Size, Locality, Machine, Network
- Processing time for block can be large and highly variable

- Example: Find SUM of below values

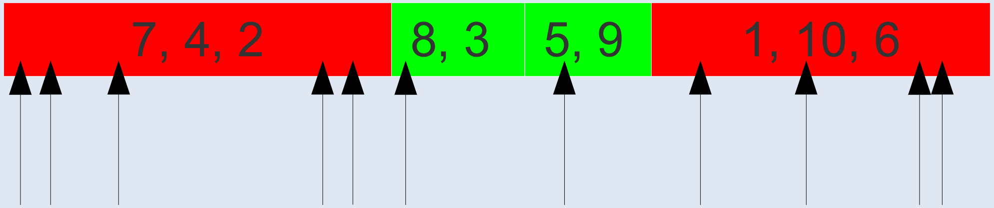


Notice, there are more arrows on red region than green region

OLA over multiple machines

- Blocks → Non-uniform → Size, Locality, Machine, Network
- Processing time for block can be large and highly variable

- Example: Find SUM of below values



Notice, there are more arrows on red region than green region

Inspection Paradox: At any random time t, (stochastically) you will be processing those blocks that take long time

Extend existing approaches

- OLA over single machine
  - Confidence interval found using classical sampling theory
  - Tuples are bundled into blocks
    - Arrive in random order
- OLA over multiple machines
  - Blocks → Non-uniform → Size, Locality, Machine, Network
  - Processing time for block can be large and highly variable
- Why it won't work ?
- How do we deal with those issues ?

Why won't previous approach work ?

- Inspection paradox → At the time of estimation, processing longer blocks

- Possible: correlation between processing time and value
  - Eg: count query

Why won't previous approach work ?

- Inspection paradox → At the time of estimation, processing longer blocks
- Possible: correlation between processing time and value
  - Eg: count query
- **Biased estimates** → current techniques won't work

Why won't previous approach work ?

- Inspection paradox → At the time of estimation, processing longer blocks

This effect is found experimentally in the paper: 'MapReduce Online'

- Possible: correl e
  - Eg: count query

- **Biased estimates** → current techniques won't work

Why won't previous approach work ?

- Inspection paradox → At the time of estimation, processing longer blocks
- Possible: correlation between processing time and value
  - Eg: count query
- Biased estimates → current techniques won't work
- Therefore, need to deal with inspection paradox in principled fashion

Extend existing approaches

- OLA over single machine
  - Confidence interval found using classical sampling theory
  - Tuples are bundled into blocks

    \- Arrive in random order
- OLA over multiple machines
  - Blocks → Non-uniform → Size, Locality, Machine, Network
  - Processing time for block can be large and highly variable
- Why it won't work ?
- How do we deal with those issues ?

How do we deal with Inspection Paradox

- Capture timing information (i.e. processing time of block)
  - Along with values

- Instead of using classical sampling theory, we output estimates using bayesian model that:
  - Allows for correlation between processing time and values
  - And also takes into account the processing time of current block

Outline

- Motivation
- Implementation
- Experiments
- Conclusion

Implementation Overview

- Framework for distributed systems: MapReduce
  - Hadoop

    - Staged processing → ~~Online~~
  - Hyracks (developed at UC Irvine)

    - Pipelining → ”Online”

    - Architecture (and API) similar to Hadoop

    - http://code.google.com/p/hyracks/

- For estimates of ”Aggregation”,
  - 2 modifications to MapReduce (Hyracks)
  - Bayesian Estimator

Implementation Overview

- Framework for distributed systems: MapReduce
  - Hadoop

    - Staged processing → ~~Online~~
  - Hyracks (developed at UC Irvine)

    - Pipelining → ”Online”

    - Architecture (and API) similar to Hadoop

    - http://code.google.com/p/hyracks/

- For estimates of ”Aggregation”,
  - 2 modifications to MapReduce (Hyracks)
  - Bayesian Estimator

Implementation Overview

- Framework for distributed systems: MapReduce
  - Hadoop

    - Staged processing → ~~Online~~
  - Hyracks (developed at UC Irvine)

    - Pipelining → "Online"

    - Architecture (and API) similar to Hadoop

    - http://code.google.com/p/hyracks/

- For estimates of "Aggregation",
  - 2 modifications to MapReduce (Hyracks)
  - Bayesian Estimator

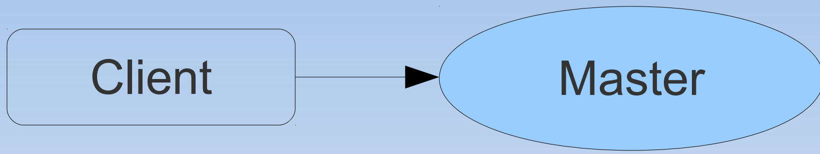
Modifications to MapReduce (Hyracks)

- Master
  - Maintains random ordering of blocks

    - Logical not physical queue
  - Assigns block from head of queue
  - Block comes to head of queue → Timer starts (processing time)

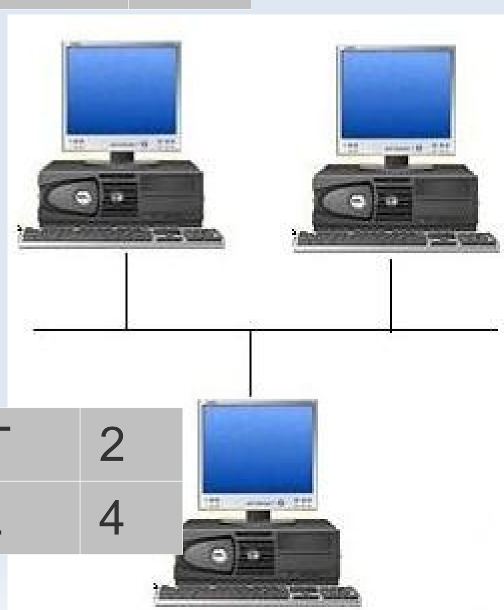
- Two intermediates set of files
  - Data file → Values
  - Metadata file → Timing information
  - Shuffle phase of reducer

Modifications to MapReduce (Hyracks)



select sum(stock_price) from nasdaq_db group by company;

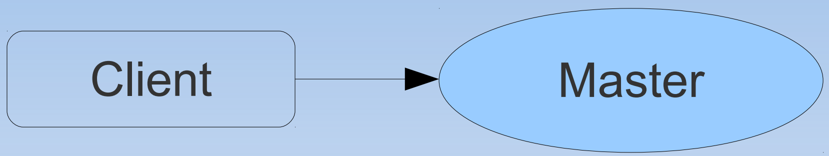
Blk1	MSFT AAPL	2 4
Blk2	ORCL	3
Blk3	AAPL	4



Blk4	MSFT	2
Blk5	ORCL	3

Blk6	MSFT	2
Blk7	AAPL	4

Modifications to MapReduce (Hyracks)



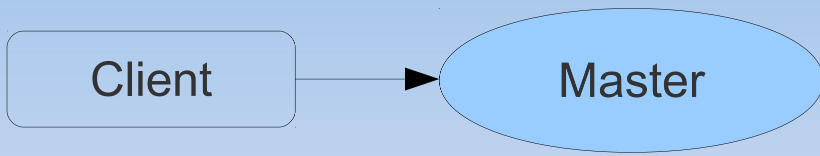
Blk1	MSFT AAPL	2 4
Blk2	ORCL	3
Blk3	AAPL	4

Blk4	MSFT	2
Blk5	ORCL	3

Blk6	MSFT	2
Blk7	AAPL	4

Time t = 0

Modifications to MapReduce (Hyracks)



Blk 1	Blk 2	Blk 3	Blk 4	Blk 5	Blk 6	Blk 7

Blk1	MSFT AAPL	2 4
Blk2	ORCL	3
Blk3	AAPL	4

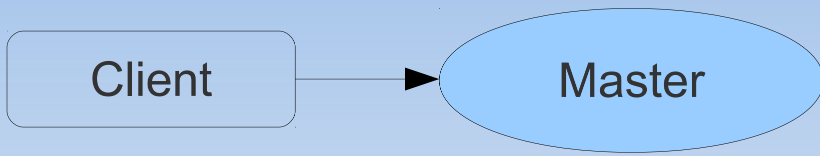
Blk4	MSFT	2
Blk5	ORCL	3

Blk6	MSFT	2
Blk7	AAPL	4

Master maintains a logical queue of the blocks

Time t = 1

Modifications to MapReduce (Hyracks)



Blk 6	Blk 5	Blk 3	Blk 1	Blk 4	Blk 7	Blk 2

Blk1	MSFT AAPL	2 4
Blk2	ORCL	3
Blk3	AAPL	4

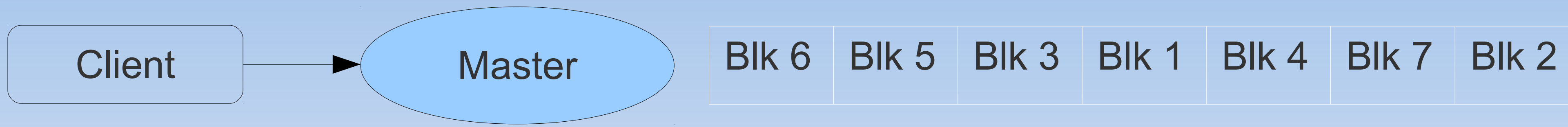
Master randomizes the queue

Blk4	MSFT	2
Blk5	ORCL	3

Blk6	MSFT	2
Blk7	AAPL	4

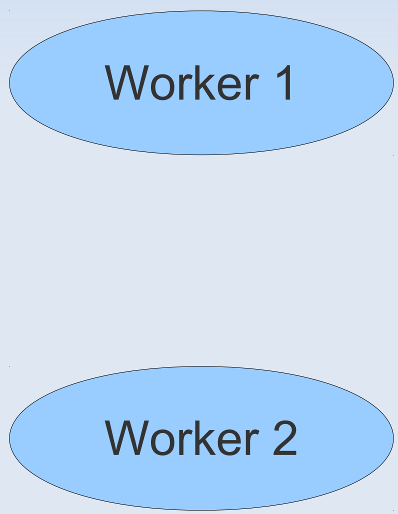
Time t = 1

Modifications to MapReduce (Hyracks)



Blk1	MSFT AAPL	2 4
Blk2	ORCL	3
Blk3	AAPL	4

Master forks workers

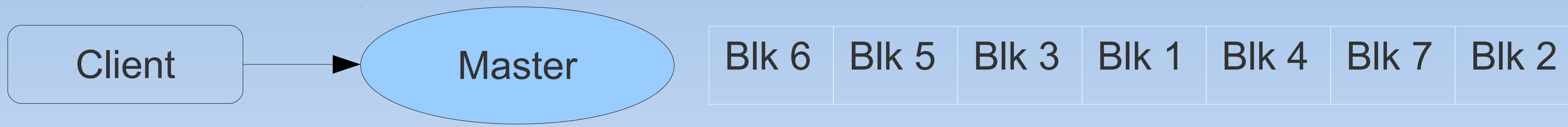


Blk4	MSFT	2
Blk5	ORCL	3

Blk6	MSFT	2
Blk7	AAPL	4

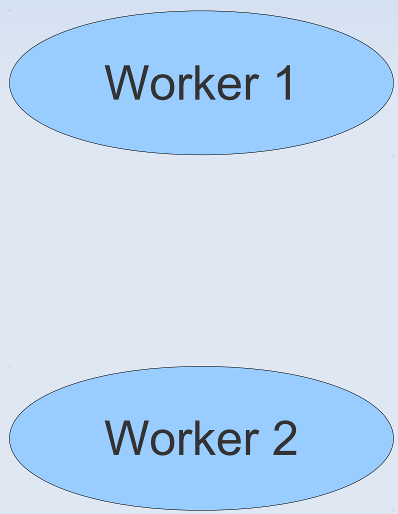
Time t = 2

Modifications to MapReduce (Hyracks)



Blk1	MSFT AAPL	2 4
Blk2	ORCL	3
Blk3	AAPL	4

Workers request for blocks

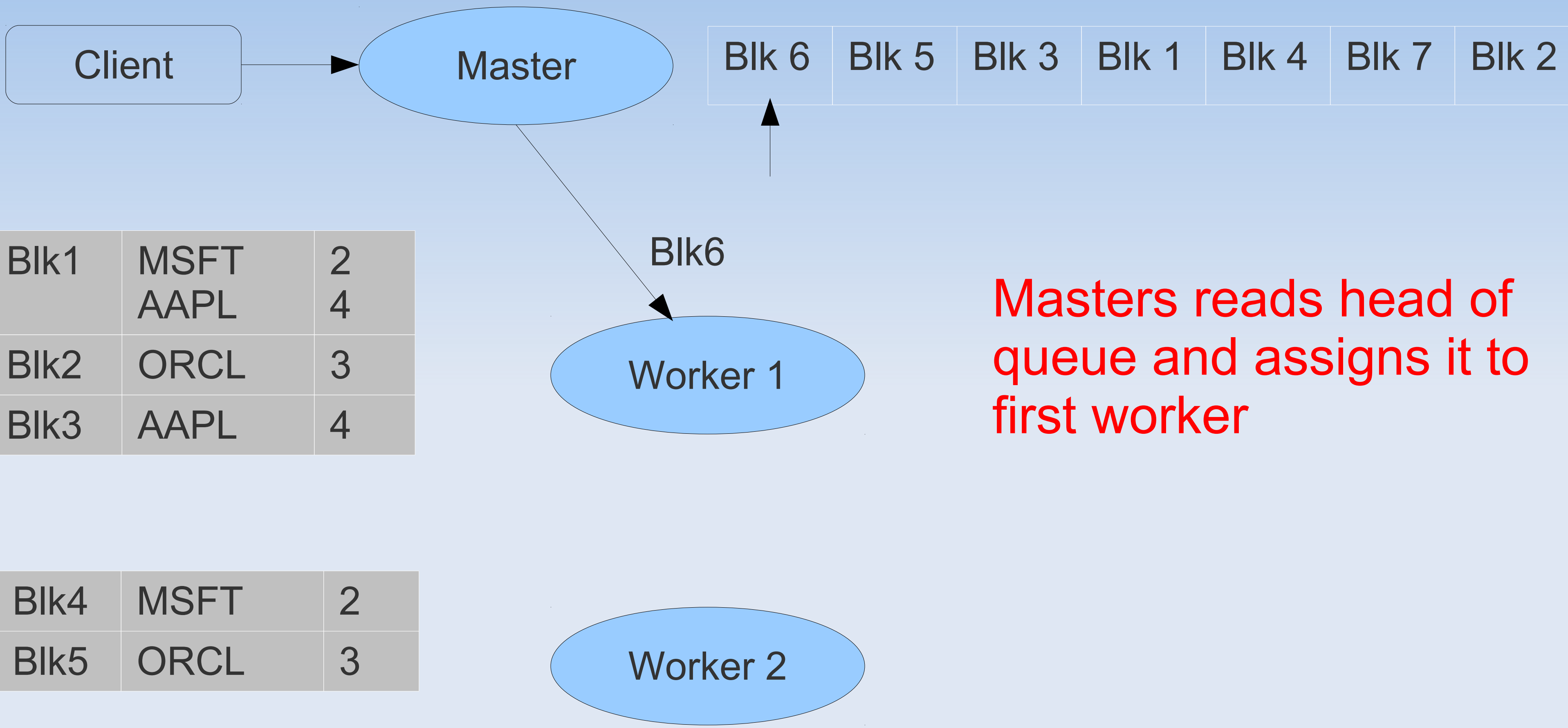


Blk4	MSFT	2
Blk5	ORCL	3

Blk6	MSFT	2
Blk7	AAPL	4

Time t = 3

Modifications to MapReduce (Hyracks)



Blk1	MSFT AAPL	2 4
Blk2	ORCL	3
Blk3	AAPL	4

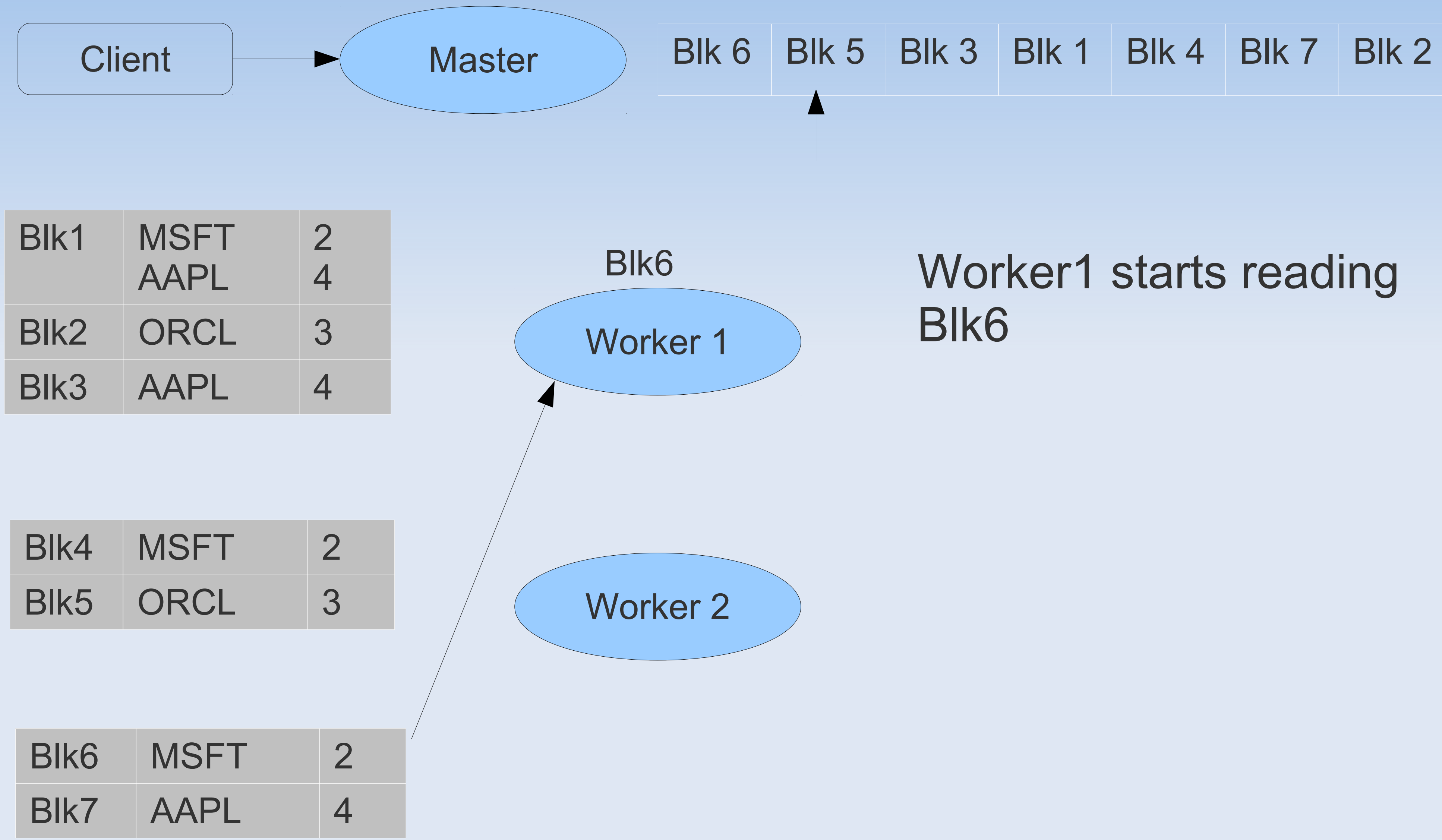
Blk4	MSFT	2
Blk5	ORCL	3

Blk6	MSFT	2
Blk7	AAPL	4

Masters reads head of queue and assigns it to first worker

Time t = 4

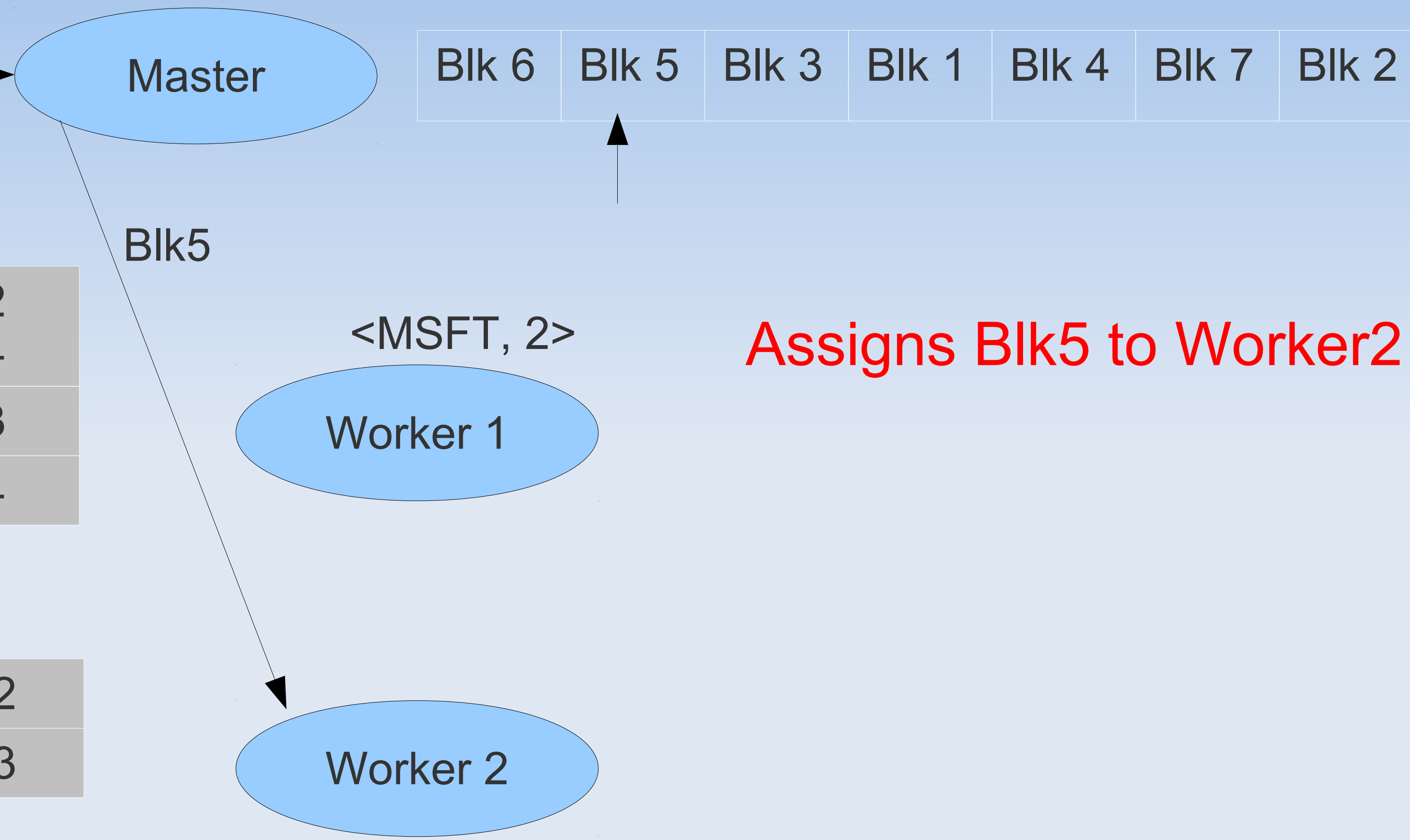
Modifications to MapReduce (Hyracks)



Worker1 starts reading Blk6

Time t = 5

Modifications to MapReduce (Hyracks)



Blk1	MSFT AAPL	2 4
Blk2	ORCL	3
Blk3	AAPL	4

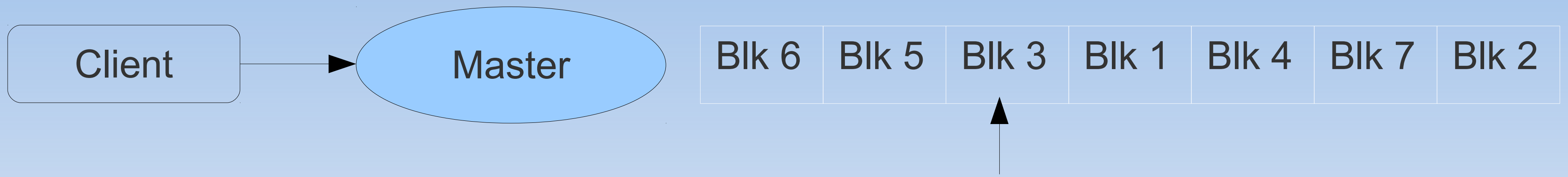
Blk4	MSFT	2
Blk5	ORCL	3

Blk6	MSFT	2
Blk7	AAPL	4

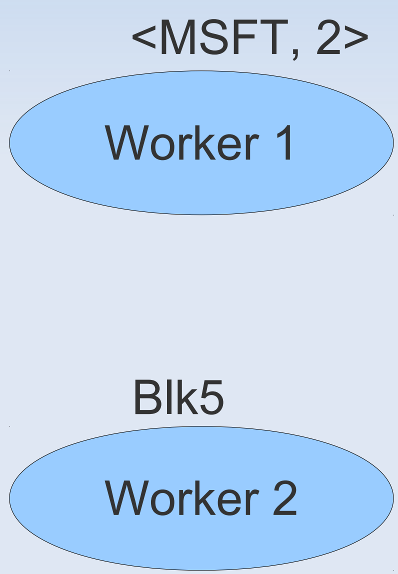
Assigns Blk5 to Worker2

Time t = 6

Modifications to MapReduce (Hyracks)



Blk1	MSFT AAPL	2 4
Blk2	ORCL	3
Blk3	AAPL	4



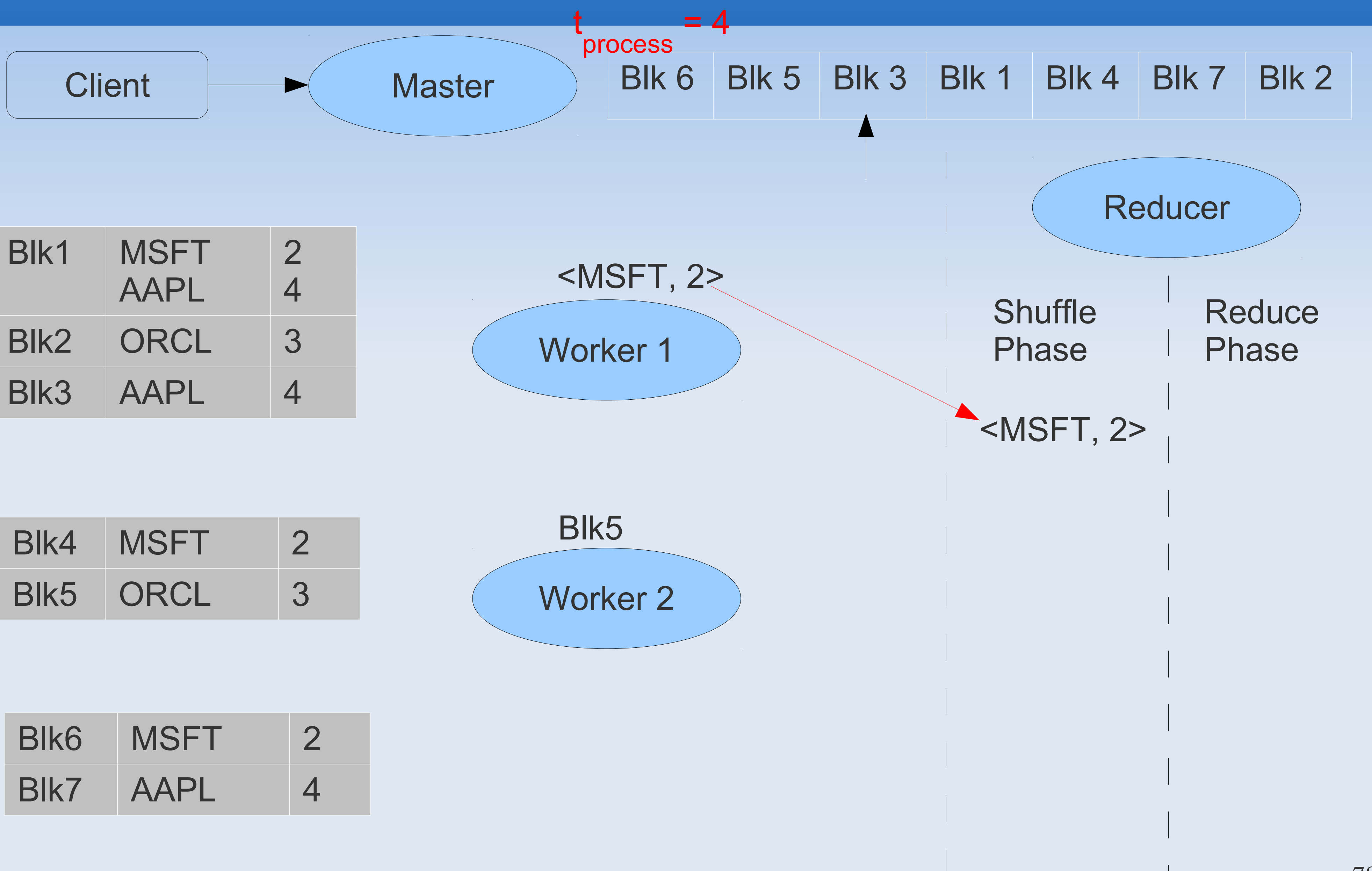
Worker1 does its map task

Blk4	MSFT	2
Blk5	ORCL	3

Blk6	MSFT	2
Blk7	AAPL	4

Time t = 7

Modifications to MapReduce (Hyracks)



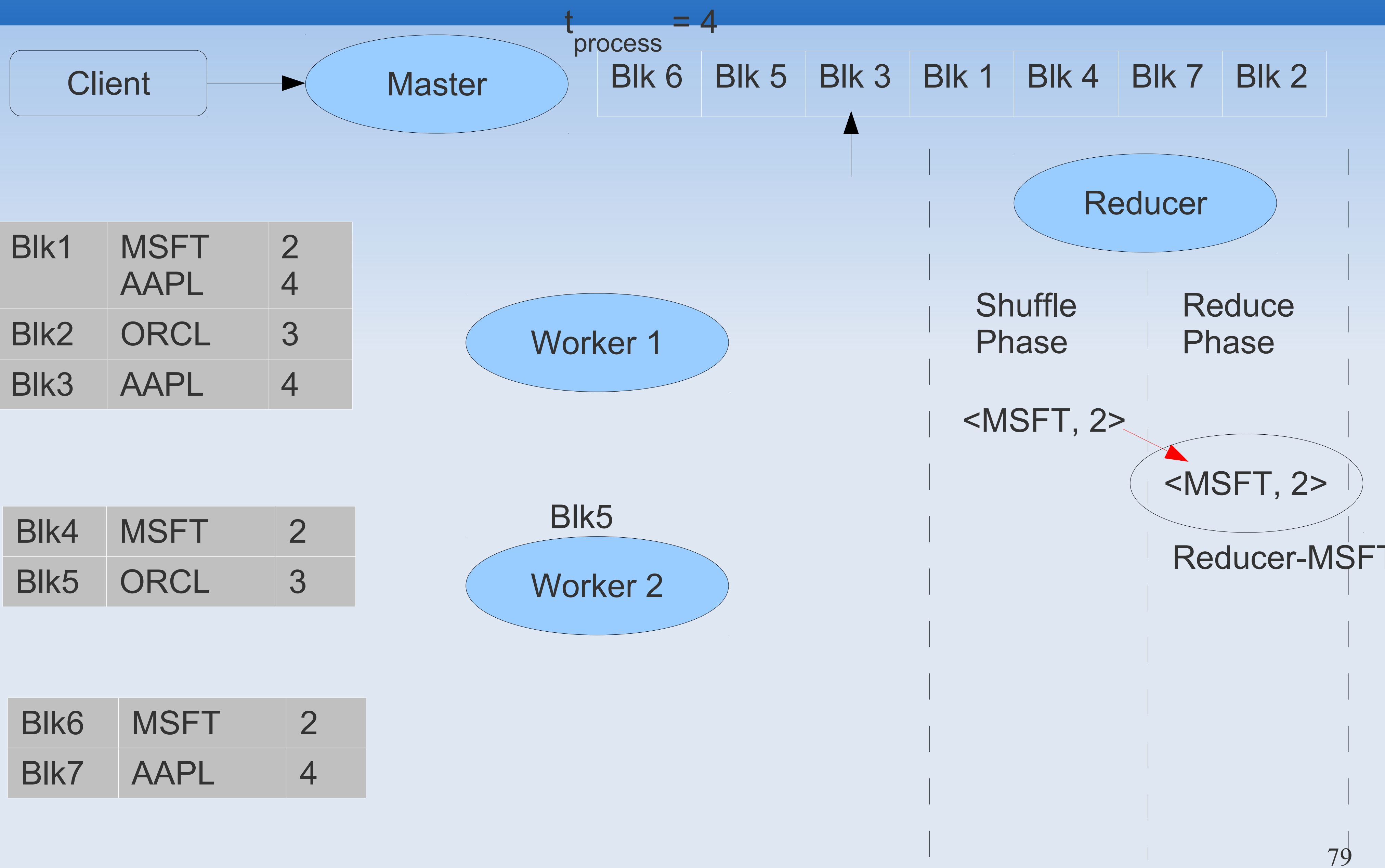
Blk1	MSFT AAPL	2 4
Blk2	ORCL	3
Blk3	AAPL	4

Blk4	MSFT	2
Blk5	ORCL	3

Blk6	MSFT	2
Blk7	AAPL	4

Time t = 8

Modifications to MapReduce (Hyracks)



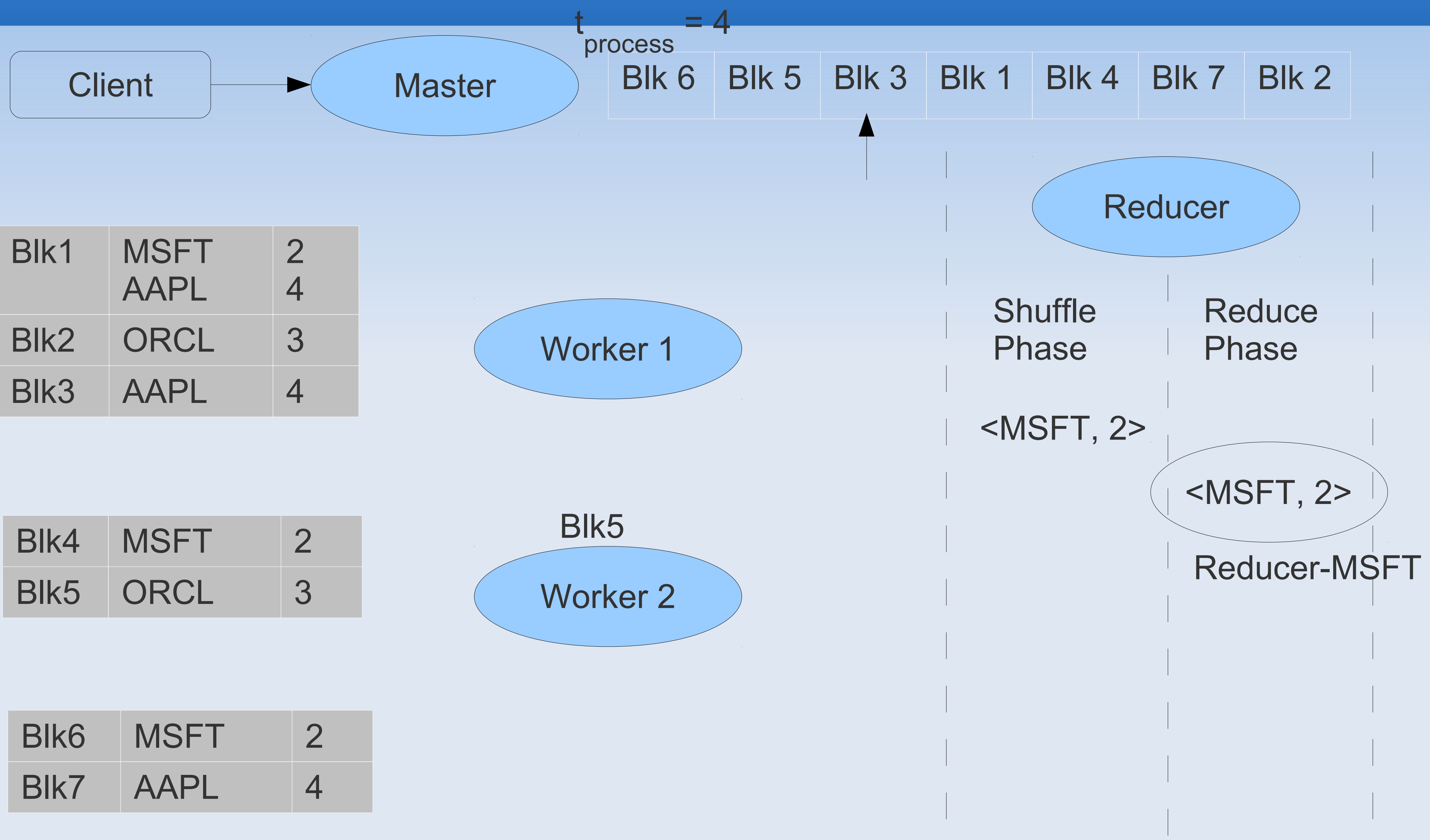
Blk1	MSFT AAPL	2 4
Blk2	ORCL	3
Blk3	AAPL	4

Blk4	MSFT	2
Blk5	ORCL	3

Blk6	MSFT	2
Blk7	AAPL	4

Time t = 9

Modifications to MapReduce (Hyracks)



Blk1	MSFT AAPL	2 4
Blk2	ORCL	3
Blk3	AAPL	4

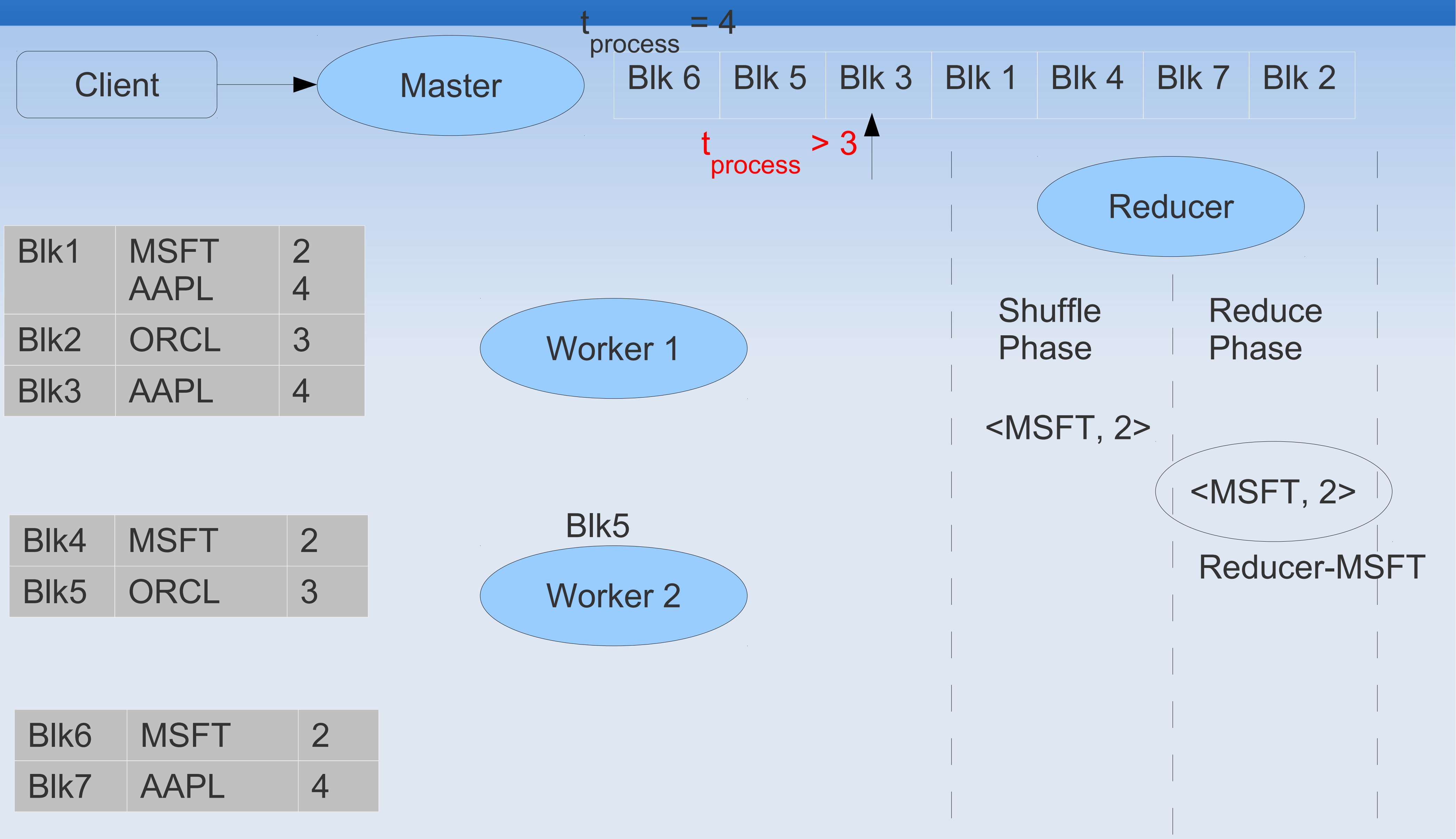
Blk4	MSFT	2
Blk5	ORCL	3

Blk6	MSFT	2
Blk7	AAPL	4

Random Time instance: Do estimation

Time t = 9

Modifications to MapReduce (Hyracks)



Blk1	MSFT AAPL	2 4
Blk2	ORCL	3
Blk3	AAPL	4

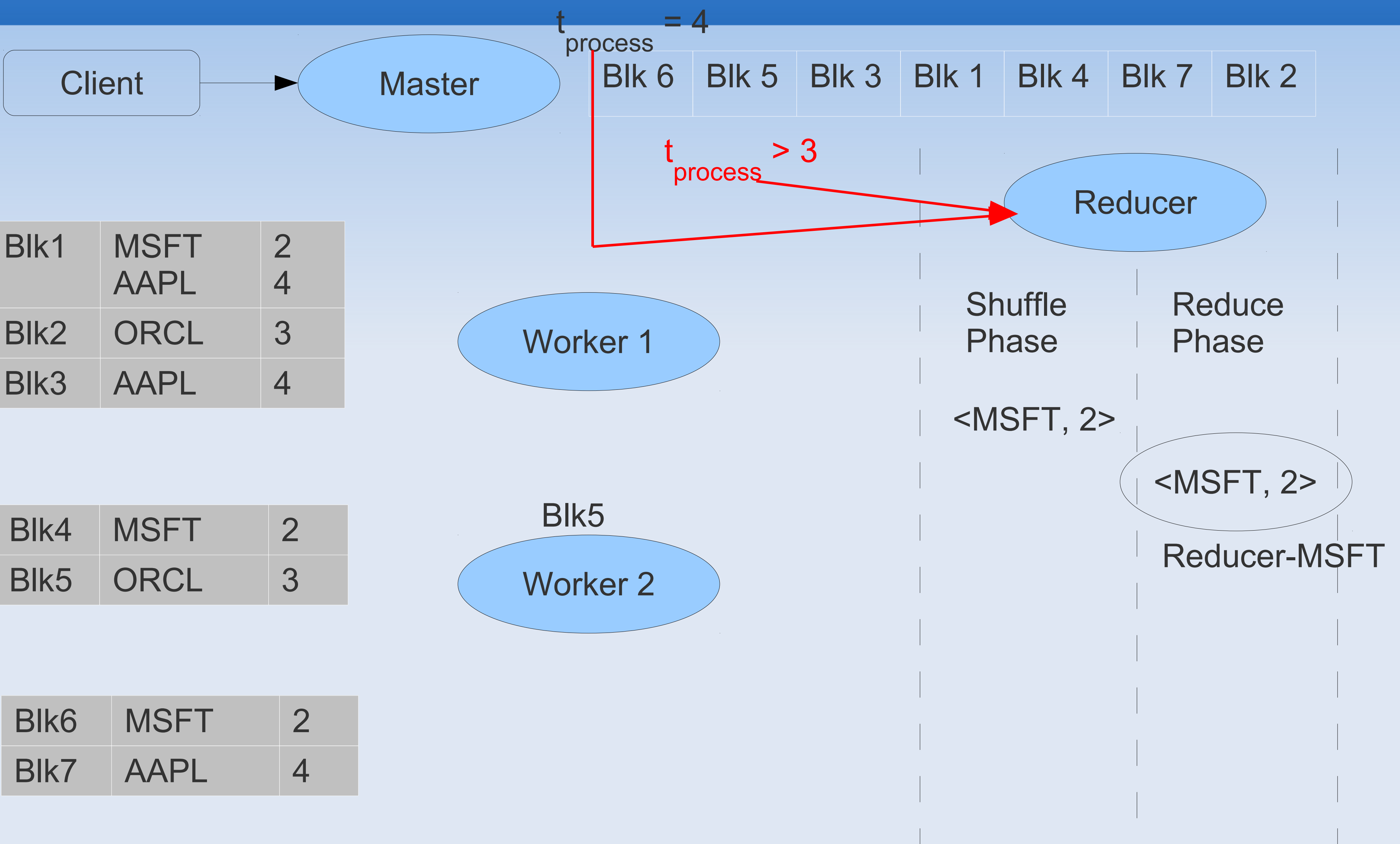
Blk4	MSFT	2
Blk5	ORCL	3

Blk6	MSFT	2
Blk7	AAPL	4

Random Time instance: Do estimation

Time t = 9

Modifications to MapReduce (Hyracks)



Blk1	MSFT AAPL	2 4
Blk2	ORCL	3
Blk3	AAPL	4

Blk4	MSFT	2
Blk5	ORCL	3

Blk6	MSFT	2
Blk7	AAPL	4

Random Time instance: Do estimation

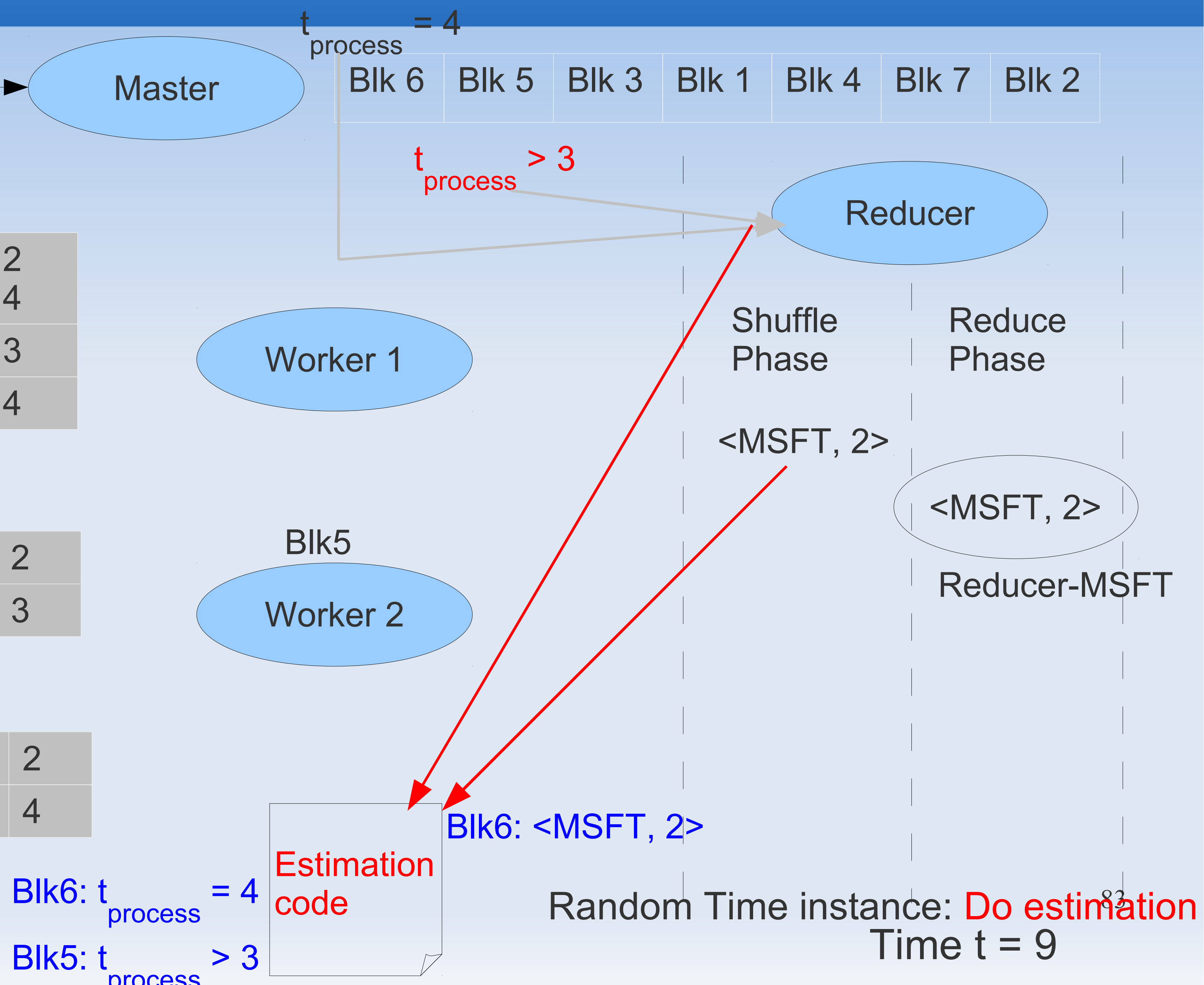
Time t = 9

Modifications to MapReduce (Hyracks)

Blk1	MSFT AAPL	2 4
Blk2	ORCL	3
Blk3	AAPL	4

Blk4	MSFT	2
Blk5	ORCL	3

Blk6	MSFT	2
Blk7	AAPL	4

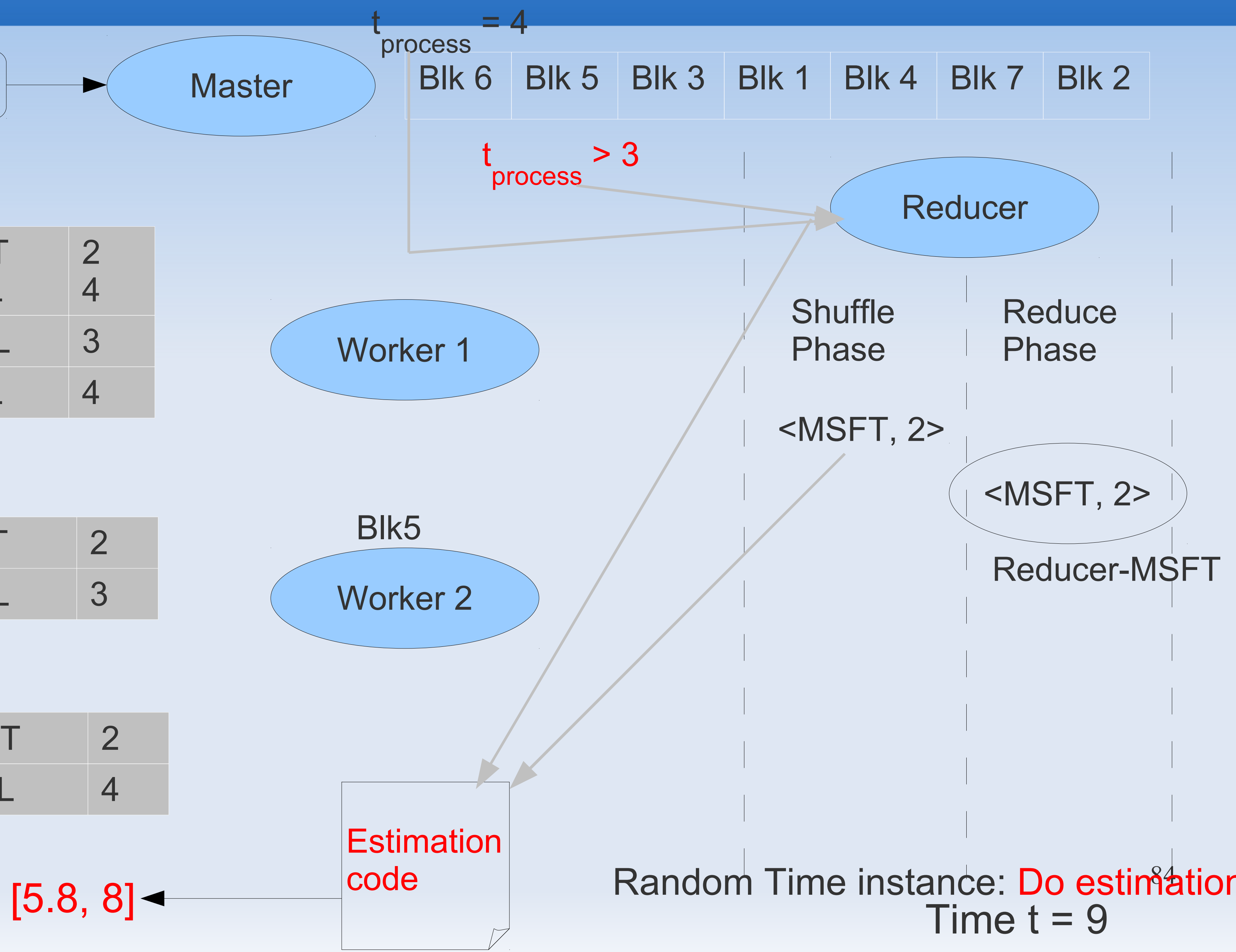


Modifications to MapReduce (Hyracks)

Blk1	MSFT AAPL	2 4
Blk2	ORCL	3
Blk3	AAPL	4

Blk4	MSFT	2
Blk5	ORCL	3

Blk6	MSFT	2
Blk7	AAPL	4



Implementation Overview

- Framework for distributed systems: MapReduce
  - Hadoop
    - Staged processing → ~~Online~~
  - Hyracks (developed at UC Irvine)
    - Pipelining → ”Online”
    - Architecture (and API) similar to Hadoop
    - http://code.google.com/p/hyracks/

- For estimates of ”Aggregation”,
  - 2 modifications to MapReduce (Hyracks)
  - Bayesian Estimator

Bayesian Estimator

- Why ? → To deal with Inspection Paradox

Bayesian Estimator

- Why ? → To deal with Inspection Paradox
- How ?
  - Allows for correlation between processing time and values
  - And also take into account the processing time of current block

Bayesian Estimator

- Why ? → To deal with Inspection Paradox
- How ?
  - Allows for correlation between processing time and values
  - And also take into account the processing time of current block
- Implementation:
  - C++ code using GNU Scientific Library and Minuit2
  - Input: Data file and Metadata file from Reducer
  - Output: Confidence Interval → Eg:[995, 1005] with 95% prob

Bayesian Estimator (Model)

- Parameterized model:
  - Timing Information: $T_{process}$, $T_{scheduling}$
  - Value: X

Bayesian Estimator (Model)

- Parameterized model:
  - Timing Information: $T_{process}$, $T_{scheduling}$
  - Value: X
- Underlying distribution
  - Classical sampling theory: f(X)

Bayesian Estimator (Model)

- Parameterized model:
  - Timing Information: $T_{process}$, $T_{scheduling}$
  - Value: X
- Underlying distribution
  - Classical sampling theory: $f(X)$
  - Our approach: $f(X, T_{process}, T_{scheduling})$

Bayesian Estimator (Model)

- Parameterized model:
  - Timing Information: $T_{process}$, $T_{scheduling}$
  - Value: X
- Underlying distribution
  - Classical sampling theory: $f(X)$
  - Our approach: $f(X, T_{process}, T_{scheduling})$

  - Correlation between X, $T_{process}$ and $T_{scheduling}$

Bayesian Estimator (Model)

- Parameterized model:
  - Timing Information: $T_{process}$, $T_{scheduling}$
  - Value: X
- Underlying distribution
  - Classical sampling theory: $f(X)$
  - Our approach: $f(X, T_{process}, T_{scheduling})$

  - Correlation between X, $T_{process}$ and $T_{scheduling}$

  - $f(X \mid T_{process} > 100000000, T_{scheduling} = 22) \neq f(X)$

Bayesian Estimator (Model)

- Parameterized model:
  - Timing Information: $T_{process}$, $T_{scheduling}$
  - Value: X
- Underlying distribution
  - Classical sampling theory: $f(X)$
  - Our approach: $f(X, T_{process}, T_{scheduling})$

    - Correlation between X, $T_{process}$ and $T_{scheduling}$

    - $f(X \mid T_{process} > 100000000, T_{scheduling} = 22) \neq f(X)$
- Estimation using Bayesian Machinery
  - Gibbs Sampler

    - Developed probability (or update) equations

Bayesian Estimator (Model)

- Parameterized model:
  - Timing Information:T T
  - Valu
- Underly
  - Clas
  - Our $T_{scheduling}$)

    - Co duling

    - f(X) ≠ f(X)

Detailed discussion in the paper

- Estimat
  - Gibbs Sampler

    - Developed probability (or update) equations

Outline

- Motivation
- Implementation
- Experiments
- Conclusion

Experiments

- Hypothesis:
  - Randomized Queue required
  - Allow correlation between processing time and value
  - Convergence of estimates
- Experiment 1: (Real dataset)
  - select sum(page_count) from wikipedia_log group by language
  - 6 months Wikipedia log (220 GB compressed, 3960 blocks)
  - 11 node cluster (4 disks, 4 cores, 12GB RAM)
  - Uniform configuration: Machines, Blocks
  - 80 mappers and 10 reducer

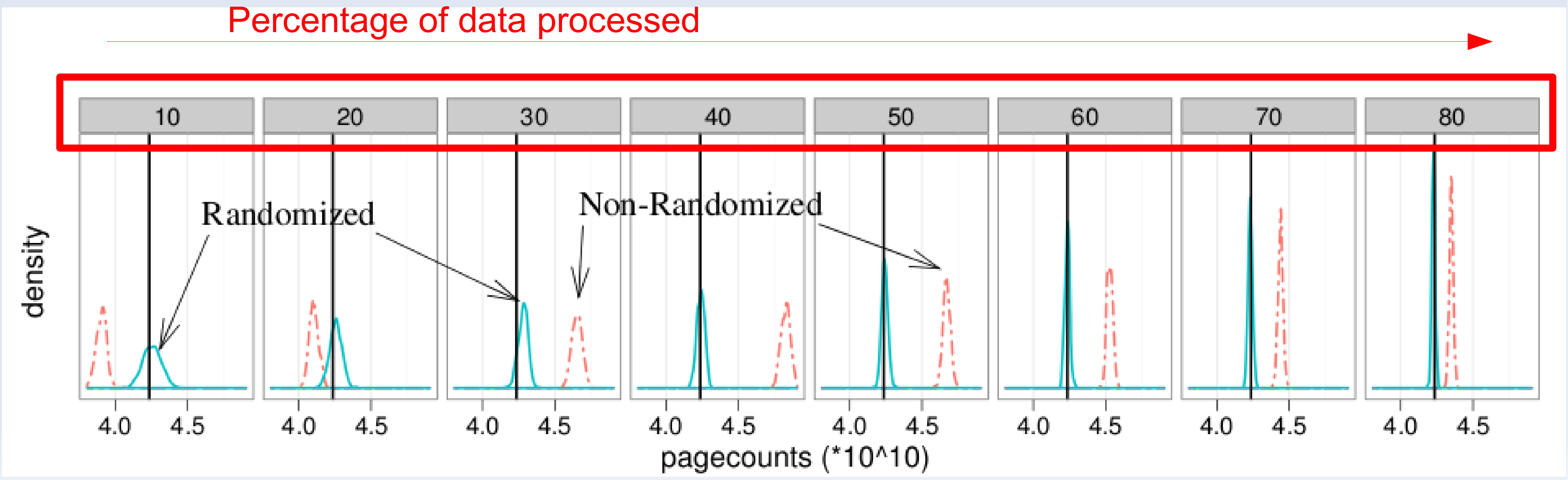
Experiments

- Hypothesis:
  - Randomized Queue required
  - Allow correlation between processing time and value
  - Convergence of estimates
- Experiment 1: (Real dataset)
  - select sum(page_count) from wikipedia_log group by language
  - 6 months Wikipedia log (220 GB compressed, 3960 blocks)
  - 11 node cluster (4 disks, 4 cores, 12GB RAM)
  - Uniform configuration: Machines, Blocks
  - 80 mappers and 10 reducer

Experiments

- Hypothesis:
  - Randomized Queue re
  - Allow correlation between [...]sing time and value
  - Convergence of estim[...]ates
- Experiment 1: (Real dataset)

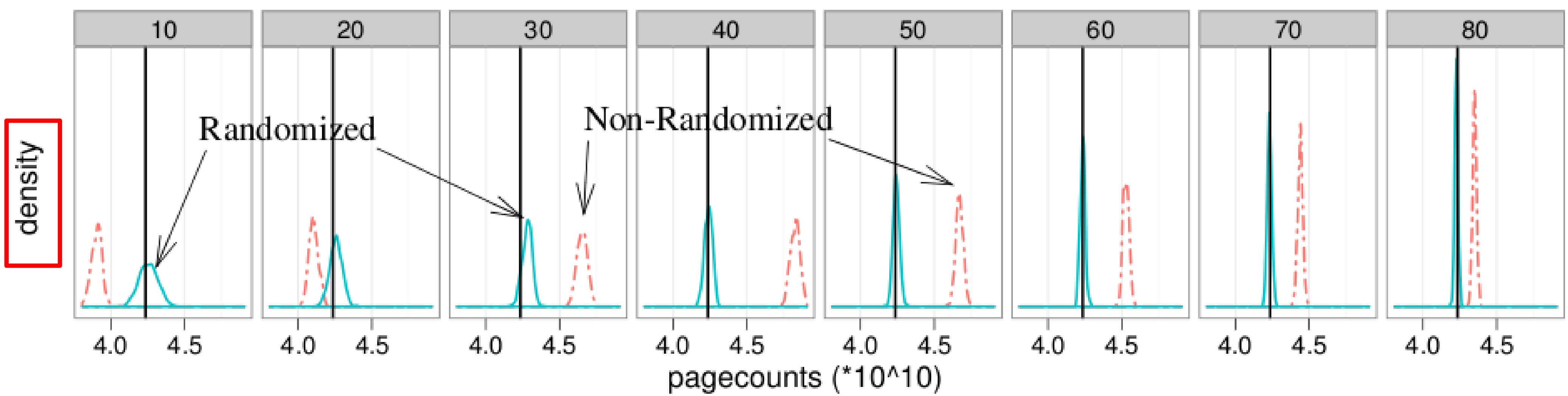
Reading the figures



Experiments

- Hypothesis:
  - Randomized Queue re
  - Allow correlation between processing time and value
  - Convergence of estimates
- Experiment 1: (Real dataset)

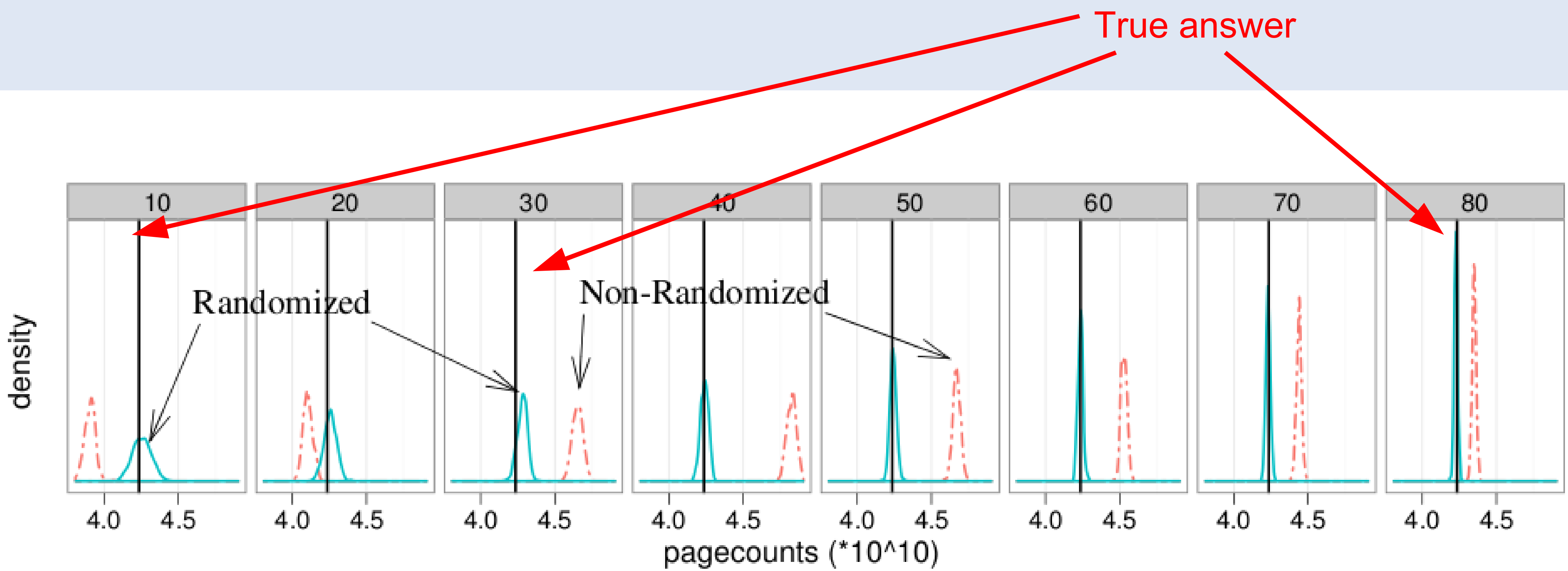
Reading the figures



Experiments

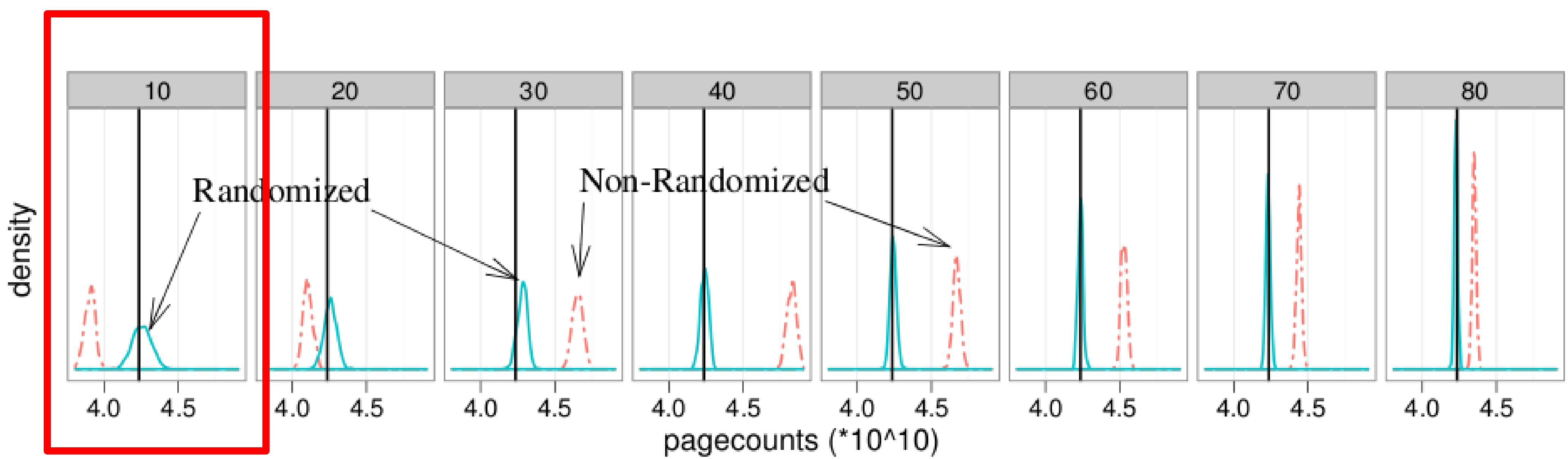
- Hypothesis:
  - Randomized Queue re
  - Allow correlation between processing time and value
  - Convergence of estimates
- Experiment 1: (Real dataset)

Reading the figures



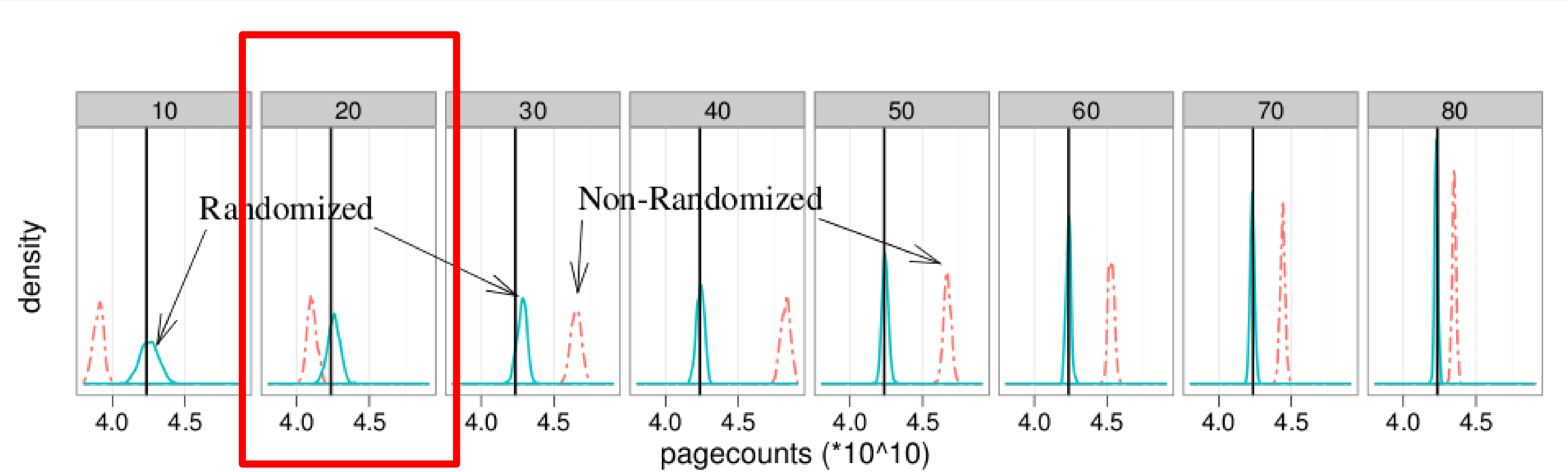
Experiments

- Hypothesis:
  - Randomized Queue required
  - Allow correlation between processing time and value
  - Convergence of estimates
- Experiment 1: (Real dataset)
  - 10% of data processed → Non-randomized: Inaccurate estimate



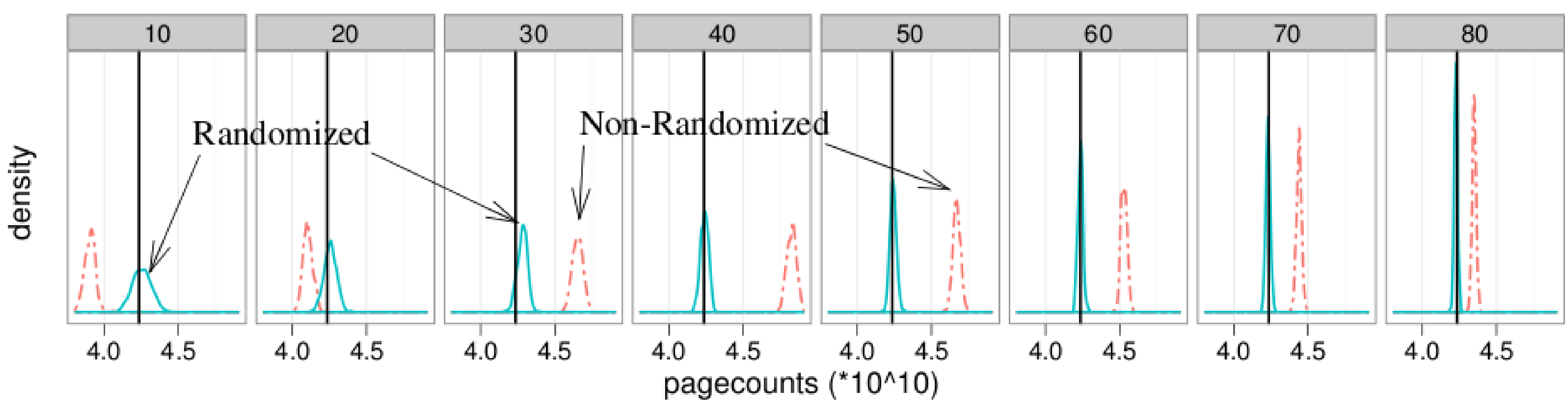
Experiments

- Hypothesis:
  - Randomized Queue required
  - Allow correlation between processing time and value
  - Convergence of estimates
- Experiment 1: (Real dataset)
  - 20% of data processed → Non-randomized: Inaccurate estimate



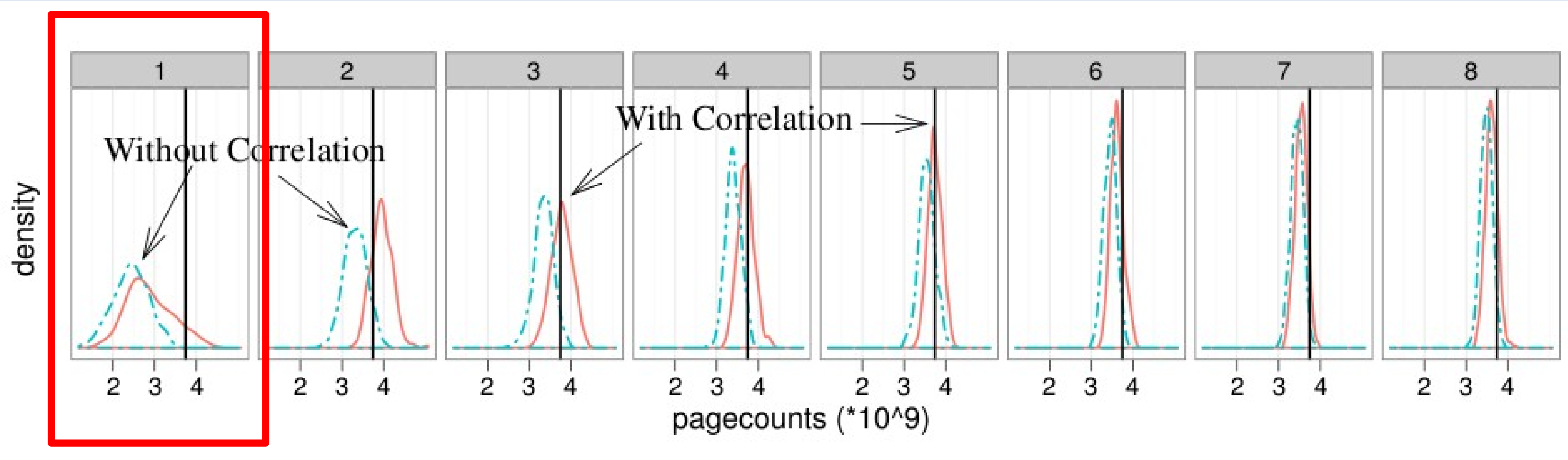
Experiments

- Hypothesis:
  - Randomized Queue required
  - Allow correlation between processing time and value
  - Convergence of estimates
- Experiment 1: (Real dataset)
  - Non-randomized → Inaccurate estimates



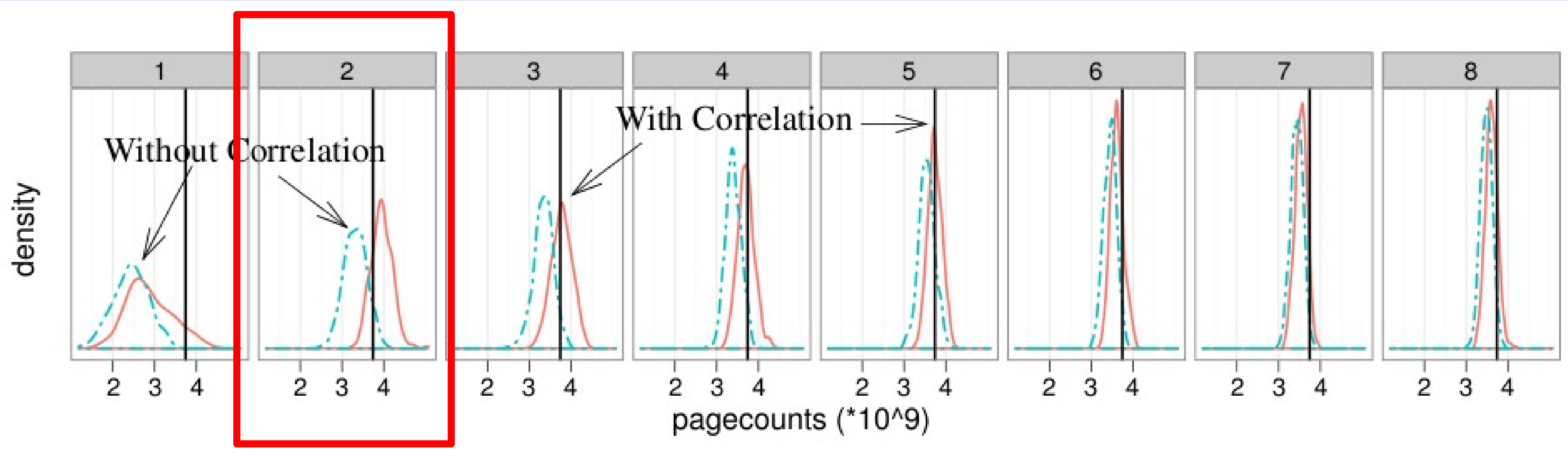
Experiments

- Hypothesis:
  - Randomized Queue required
  - Allow correlation between processing time and value
  - Convergence of estimates
- Experiment 1: (Real dataset)
  - Processing large block → no correlation detected



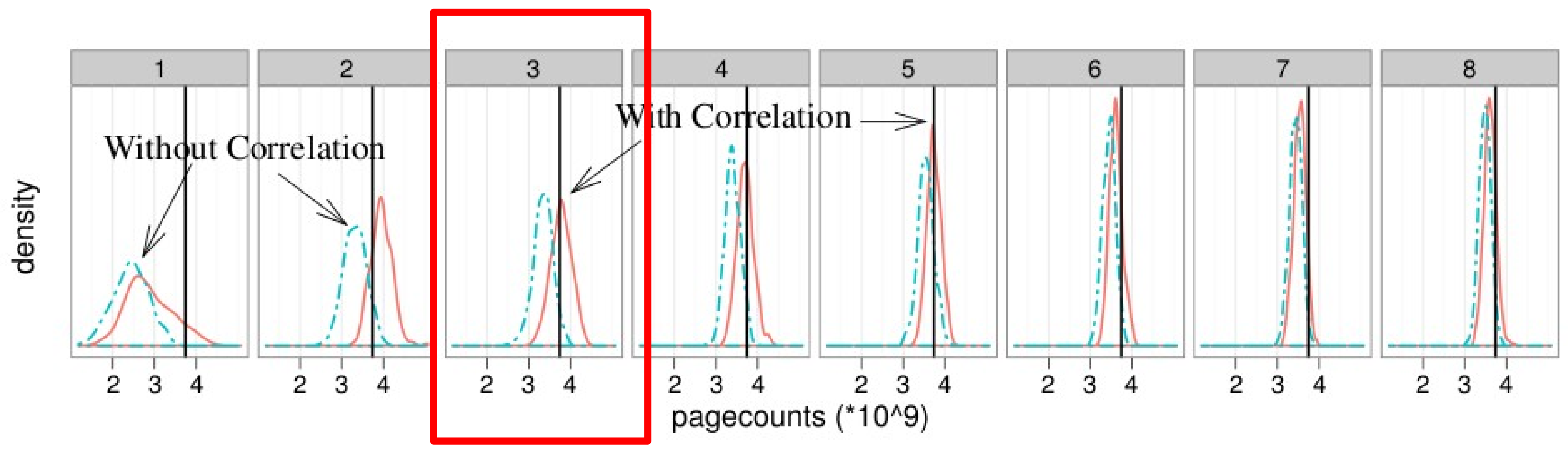
Experiments

- Hypothesis:
  - Randomized Queue required
  - Allow correlation between processing time and value
  - Convergence of estimates
- Experiment 1: (Real dataset)
  - Correlation detected → With correlation: Slightly more accurate



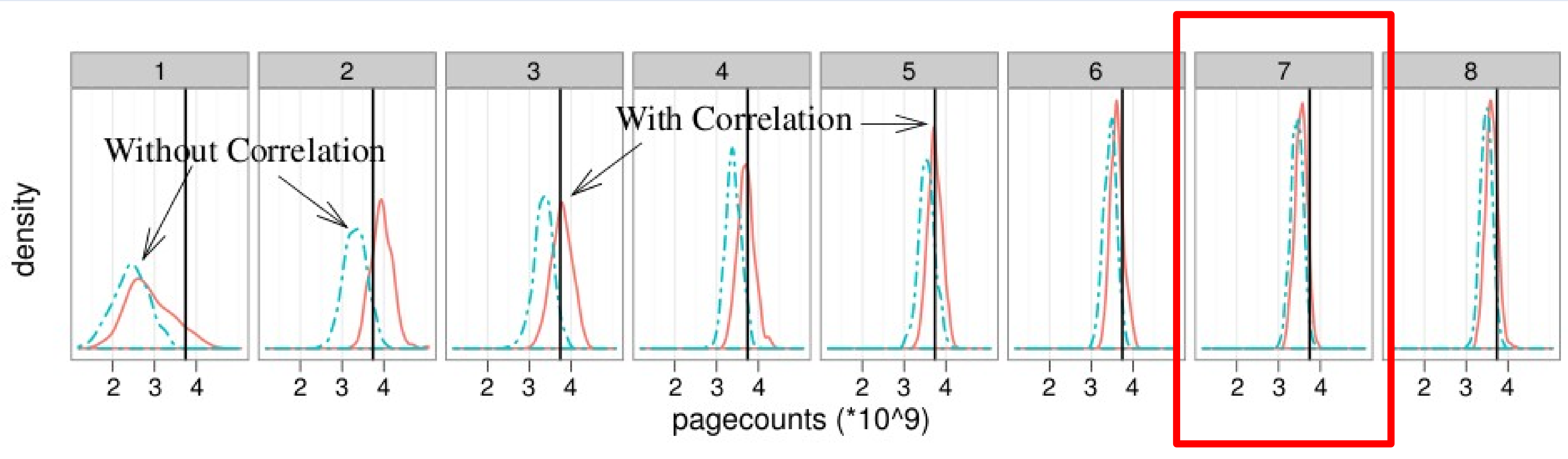
Experiments

- Hypothesis:
  - Randomized Queue required
  - Allow correlation between processing time and value
  - Convergence of estimates
- Experiment 1: (Real dataset)
  - Correlation detected → With correlation: Unbiased



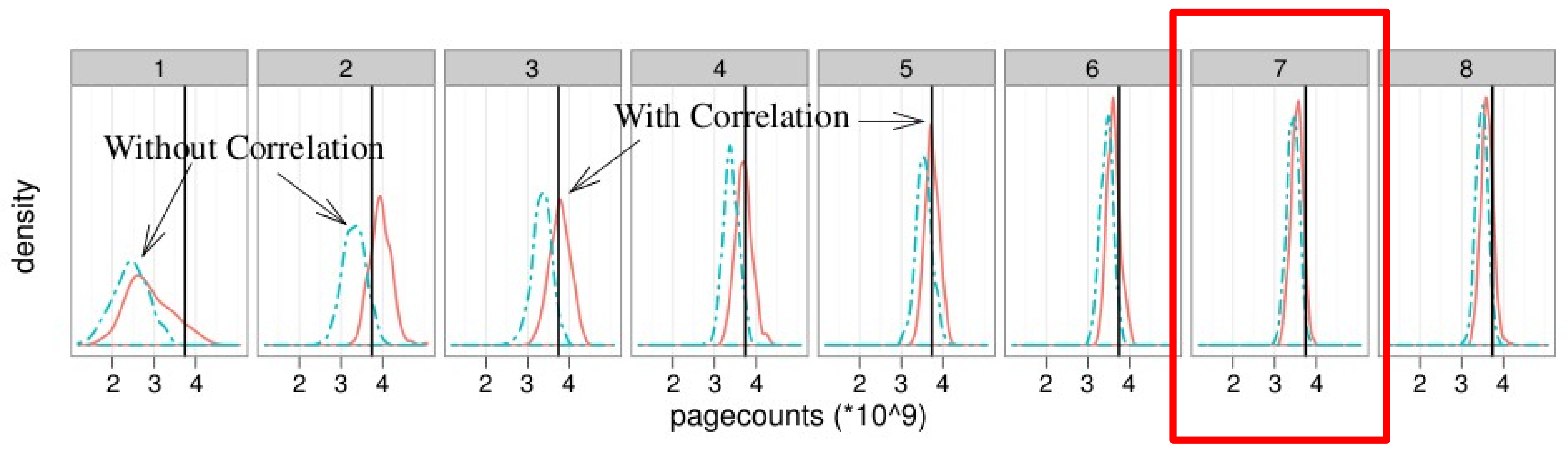
Experiments

- Hypothesis:
  - Randomized Queue required
  - Allow correlation between processing time and value
  - Convergence of estimates
- Experiment 1: (Real dataset) → Uniform Configuration (low correlation)



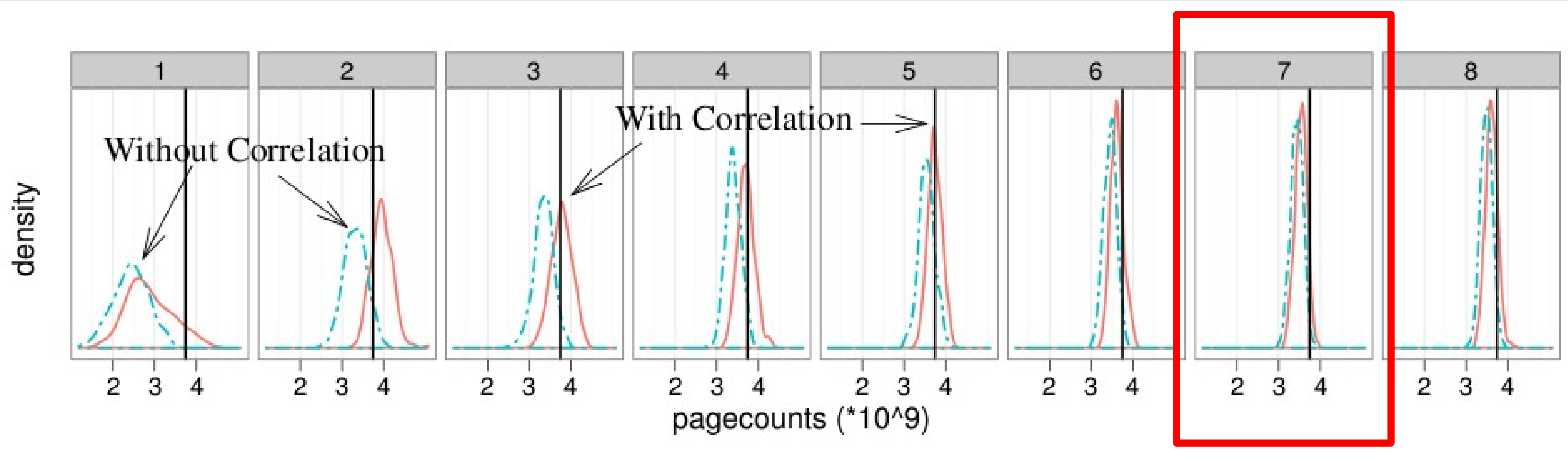
Experiments

- Hypothesis:
  - Randomized Queue required
  - Allow correlation between processing time and value
  - Convergence of estimates
- Experiment 1: (Real dataset) → Uniform Configuration (low correlation) + As ↑ data, likelihood takes over



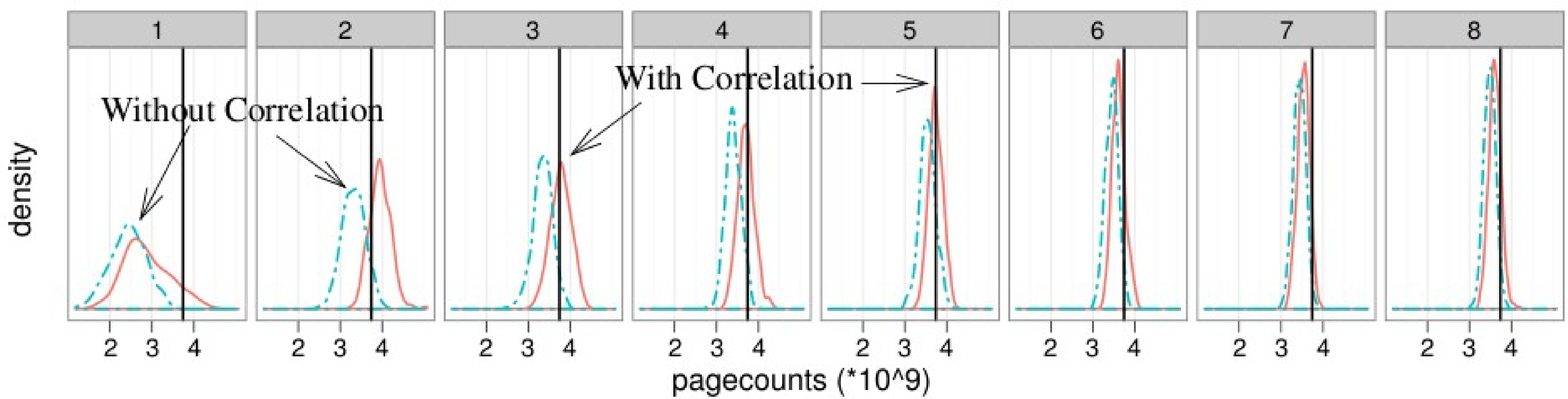
Experiments

- Hypothesis:
  - Randomized Queue required
  - Allow correlation between processing time and value
  - Convergence of estimates
- Experiment 1: (Real dataset) → Uniform Configuration (low correlation) + As ↑ data, likelihood takes over → estimates similar



Experiments

- Hypothesis:
  - Randomized Queue required
  - Allow correlation between processing time and value
  - Convergence of estimates
- Experiment 1: (Real dataset)



Outline

- Motivation
- Implementation
- Experiments
- Conclusion

Conclusion

- OLA over MapReduce
  - Statistically robust estimates
- Model that accounts for biases that can arise in distributed environment
- Little modification to existing MapReduce architecture

Thanks for your time and attention

Questions ?