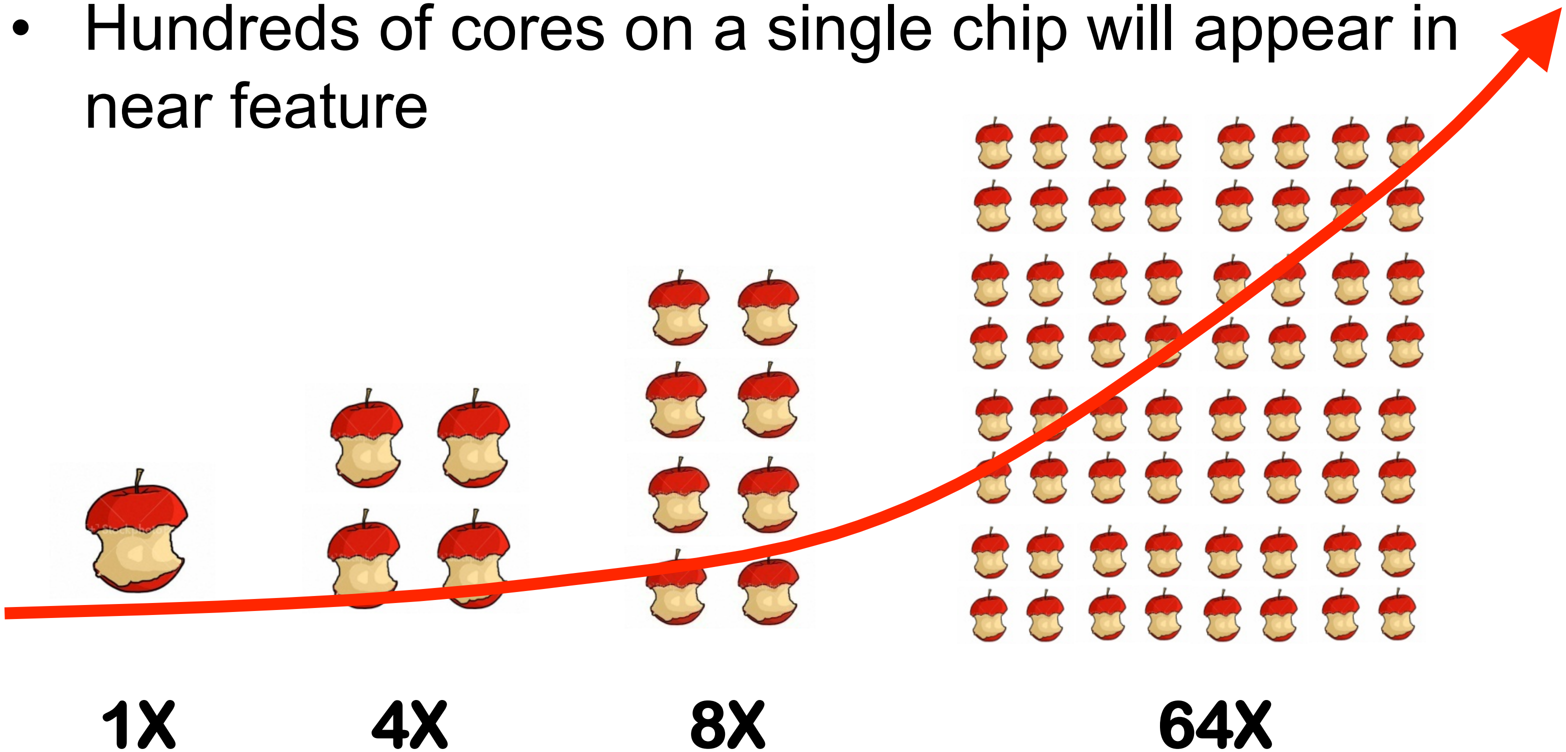
Optimizing the Performance and Scalability of MapReduce for Multicore

Haibo Chen
Parallel Processing Institute
Fudan University
http://ppi.fudan.edu.cn/haibo_chen

Multicore

Multicore is commercially prevalent recently

- Eight cores and Twelve cores on a chip are common,
- Hundreds of cores on a single chip will appear in near feature



Multicore: Challenges

How to fully harness the likely abundant cores?

- Data parallel applications fit well with multi-core system
  - processes data in private cache of cores
  - shares data within cores by main memory

- Issue#1: easy to use
  - Average programmers can use

- Issues#2: easy to scale
  - Can easily scale to a number of cores/nodes

Data-Parallel Application

Data-parallel applications emerge and rapidly increase in past 10 years

- Google processes about 24 petabytes of data per day in 2008
- The movie AVATAR takes over 1 petabyte of local storage for 3D rendering *
- …

* http://www.information-management.com/newsletters/avatar_data_processing-10016774-1.html

Data-parallel Programming Model

MapReduce: a simple programming model for data-parallel applications from Google

Data-parallel Programming Model

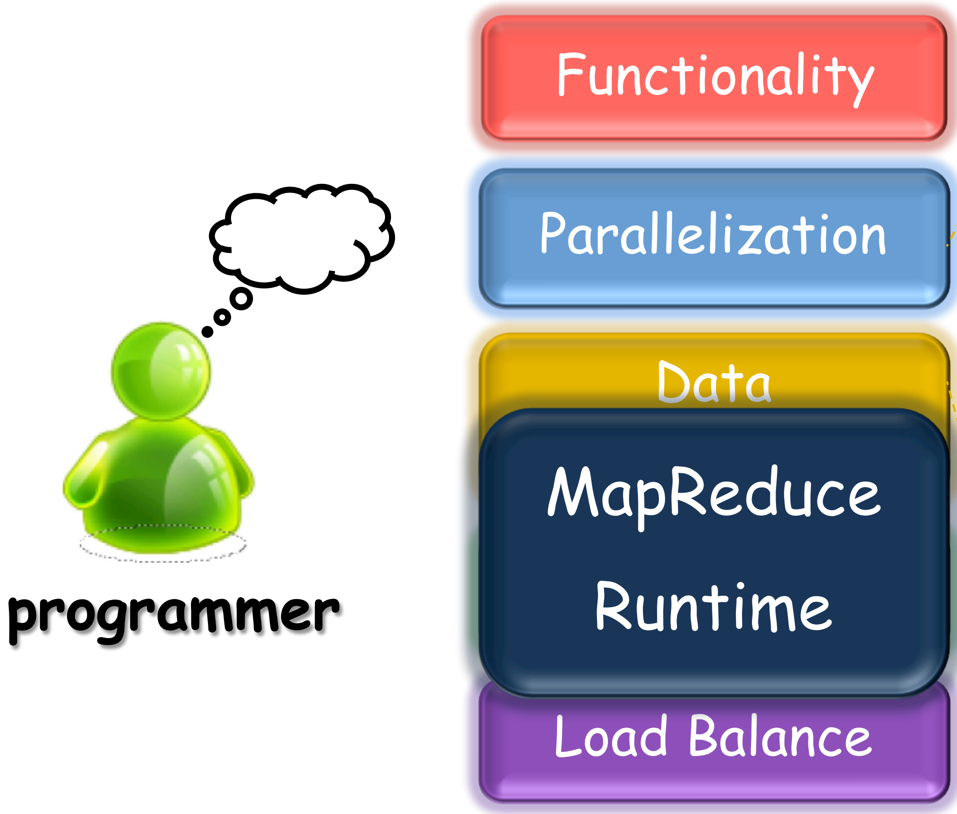
MapReduce: a simple programming model for data-parallel applications from Google

programmer

- Functionality
- Parallelism
- Data Distribution
- Fault Tolerance
- Load Balance

Data-parallel Programming Model

MapReduce: a simple programming model for data-parallel applications from Google Labs



Two Primitive:

Map (***input***)

Reduce (***key***, ***values***)

Data-parallel Programming Model

MapReduce: a simple programming model for data-parallel applications from Google Labs

programmer

Functionality

MapReduce Runtime

Two Primitive:

Map (***input***)
for each ***word*** in ***input***
emit (***word***, ***1***)

Reduce (***key***, ***values***)
int ***sum*** = 0;
for each ***value*** in ***values***
sum += ***value***;
emit (***word***, ***sum***)

State-of-the-Art *MapReduce Systems*

Hadoop an open-source alternative of Google's fairly secrete implementation

Phoenix a shared-memory implementation of MapReduce model for data-intensive processing tasks from *Stanford*

When MapReduce Meets Multicore

MapReduce: original developed for programming large clusters

Results:

- Little consideration of locality and parallelism on multiple cores on a single node
  - E.g., Hadoop uses a JVM-based runtime, which is really hard to exploit the multicore resource

- Aggressively parallelism for large clusters not directly fit multicore
  - Contentions on cache, memory and OS services
  - Simply adapting MapReduce to multicore is not optimal

--- Outline ---

Ostrich:

Optimizing MapReduce for a single machine with multiples core

Chadoop (Briefly):

Exploiting the Locality and Parallelism with Hierarchical MapReduce on the Cloud

MapReduce on Multicore

Phoenix [HPCA'07 IISWC'09]

A MapReduce runtime for shared-memory

> CMPs and SMPs

> NUMA

MapReduce on Multicore

Phoenix [HPCA'07 IISWC'09]

A MapReduce runtime for shared-memory

> CMPs and SMPs

> NUMA

Features

> Parallelism: *threads*

> Communication: *shared address space*

MapReduce on Multicore

Example: Phoenix [HPCA'07 IISWC'09]

A MapReduce runtime for shared-memory

> CMPs and SMPs

> NUMA

Features

> Parallelism: *threads*

> Communication: *shared address space*

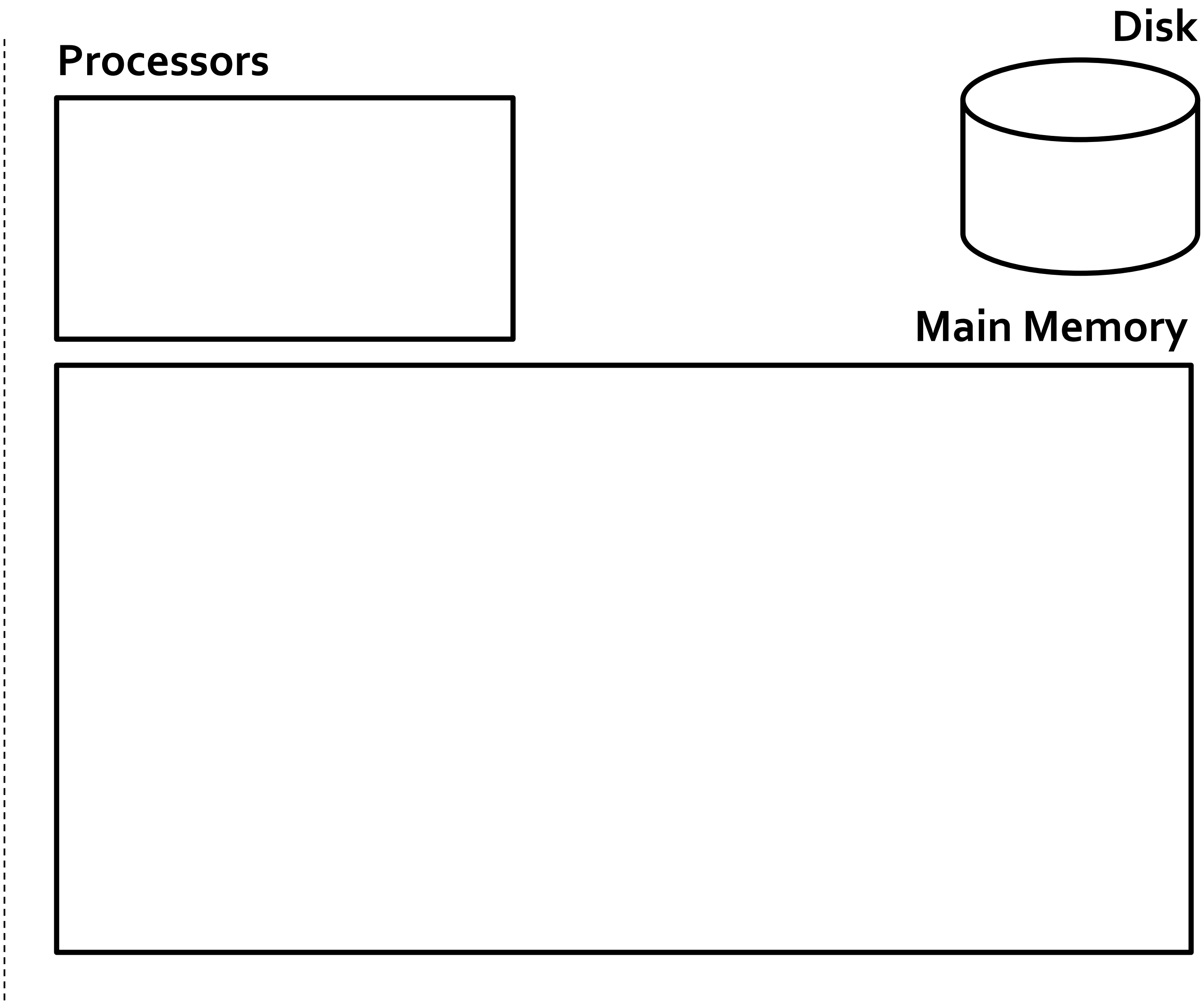
Heavily optimized runtime

> Runtime algorithm *e.g. locality-aware task distribution*

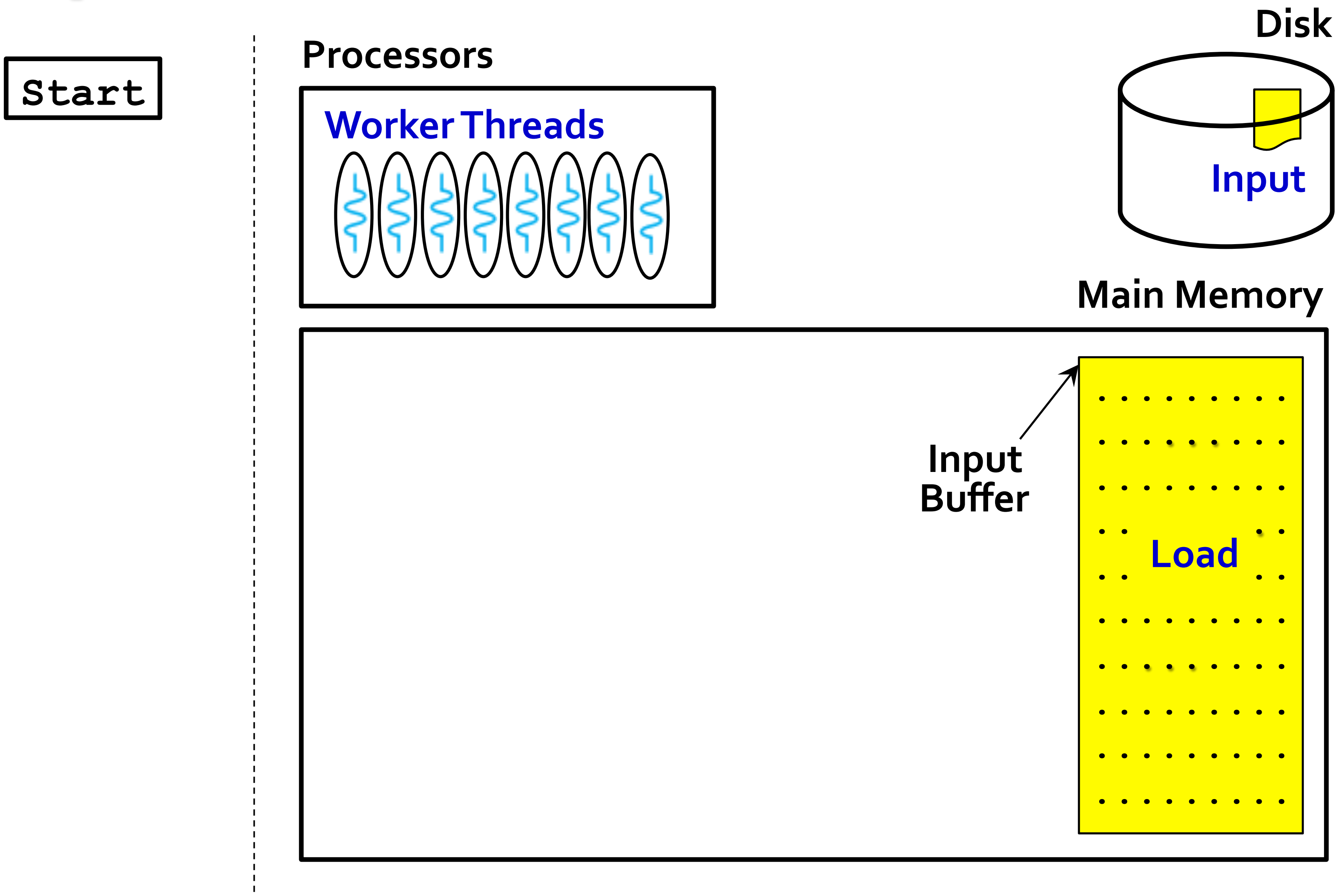
> Scalable data structure *e.g. hash table*

> OS Interaction *e.g. memory allocator, thread pool*

Implementation on Multicore

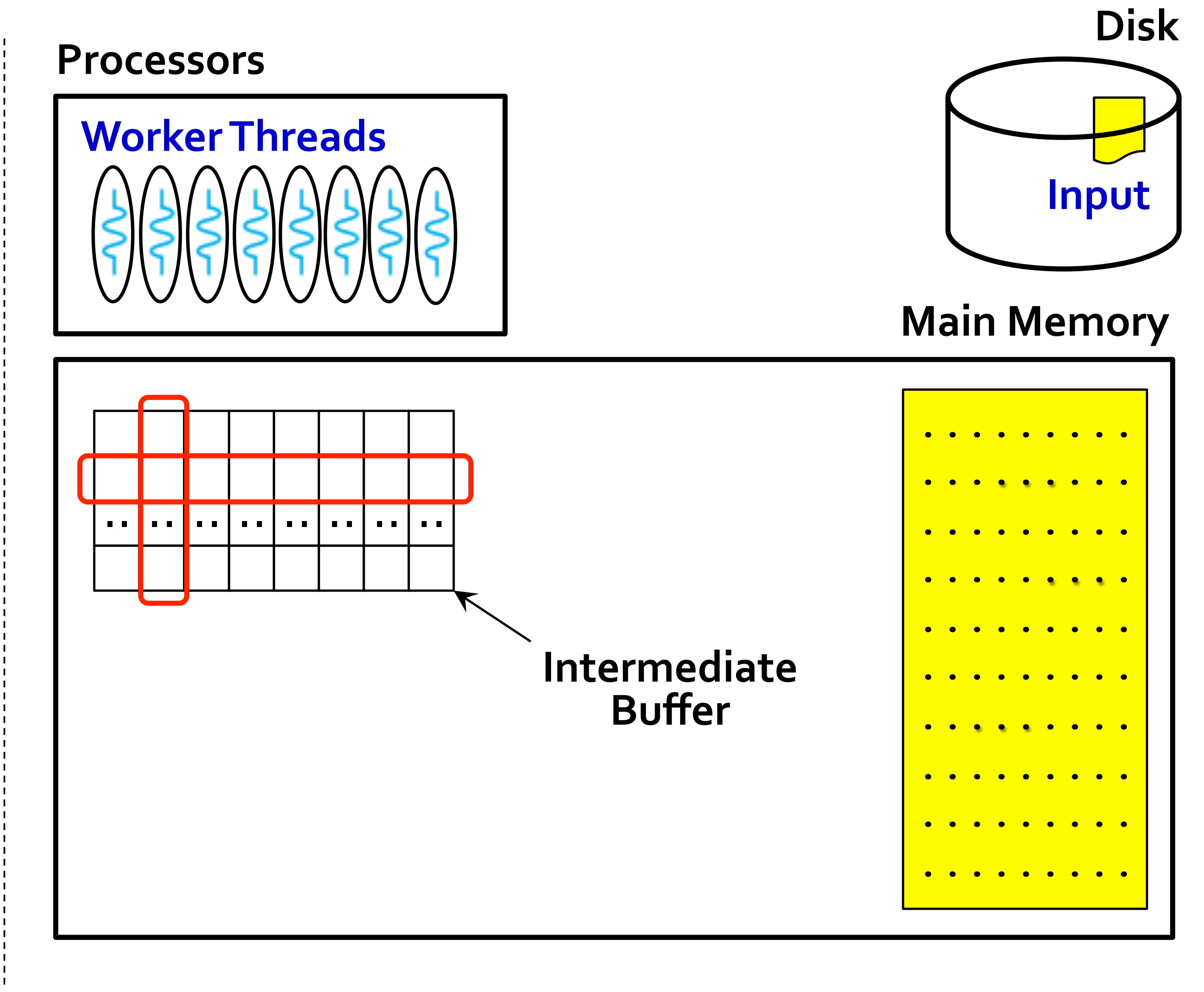


Implementation on Multicore

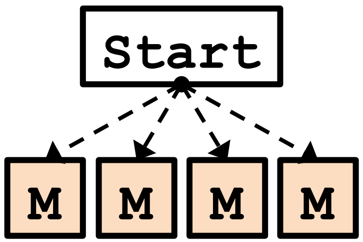


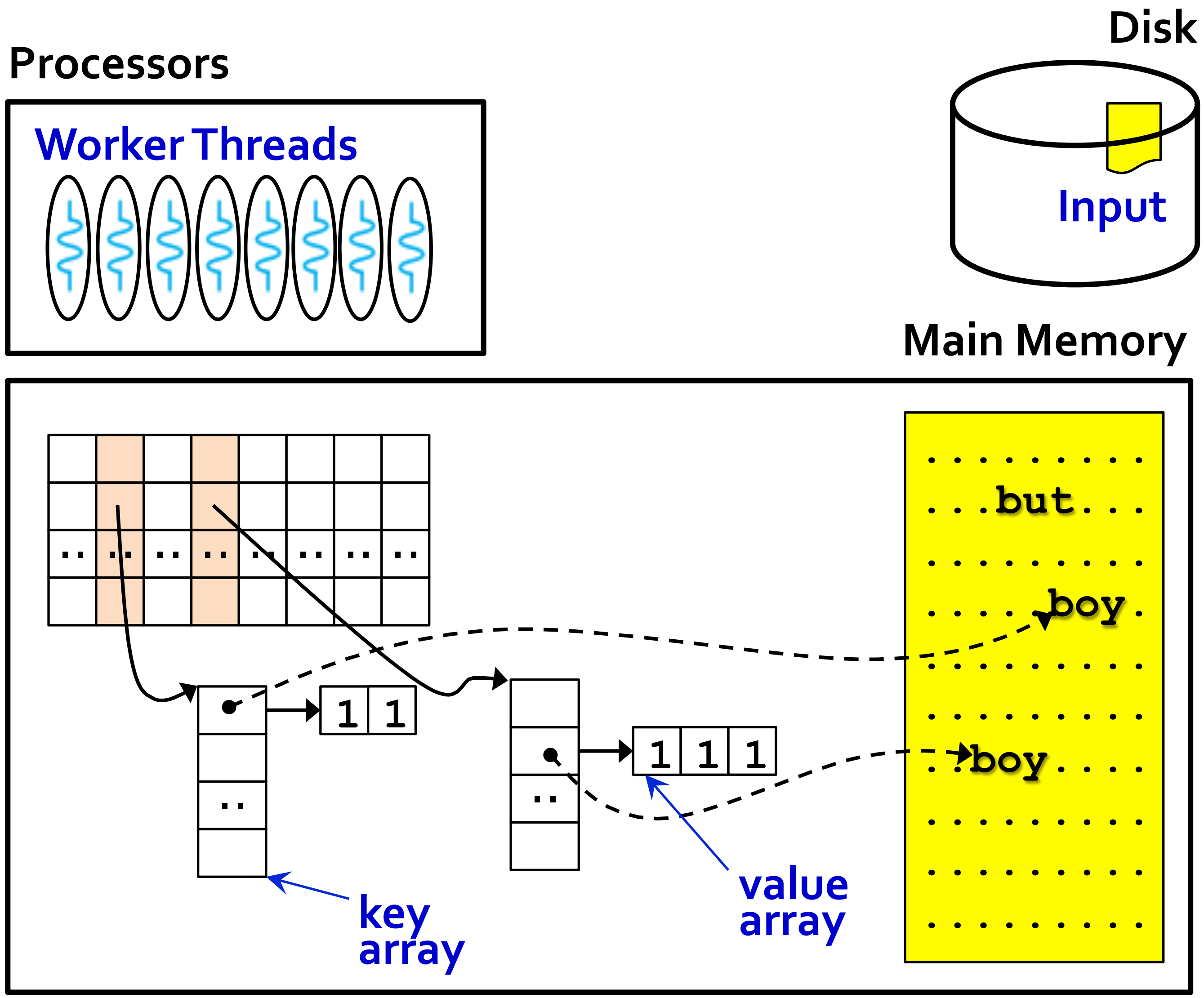
Implementation on Multicore

`Start`

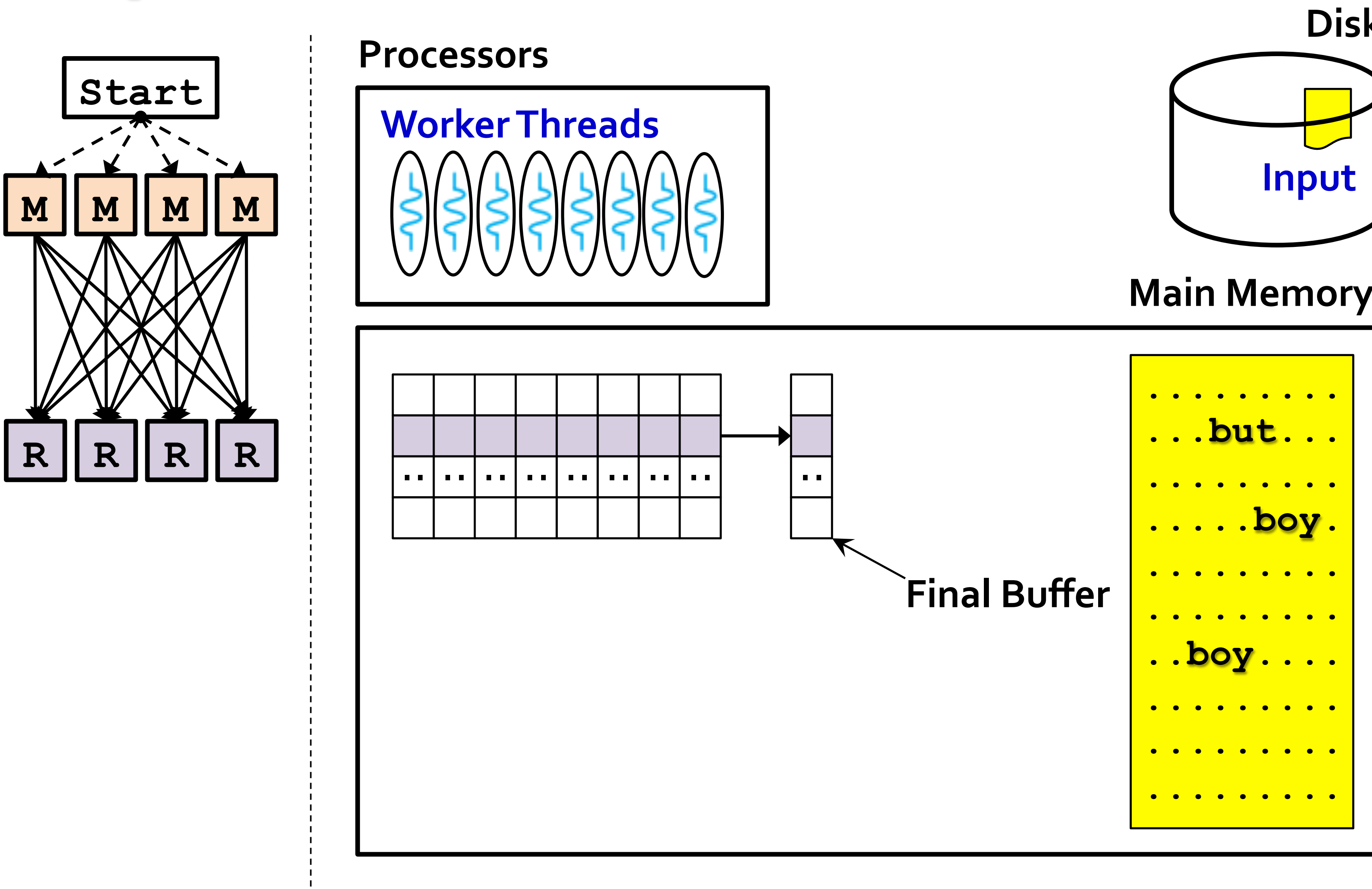


Implementation on Multicore

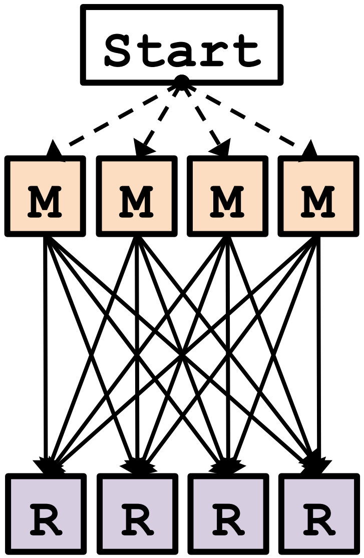


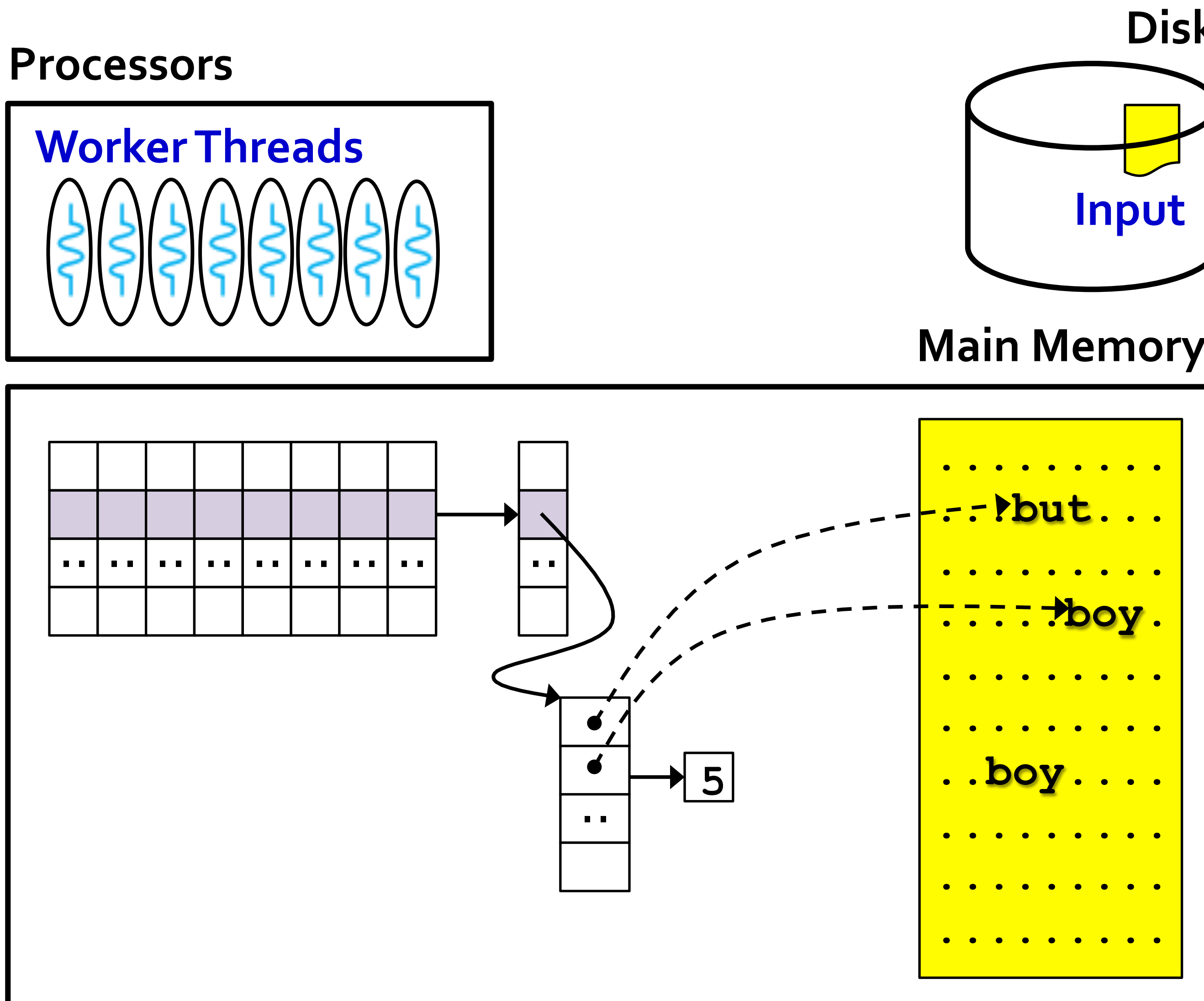


Implementation on Multicore

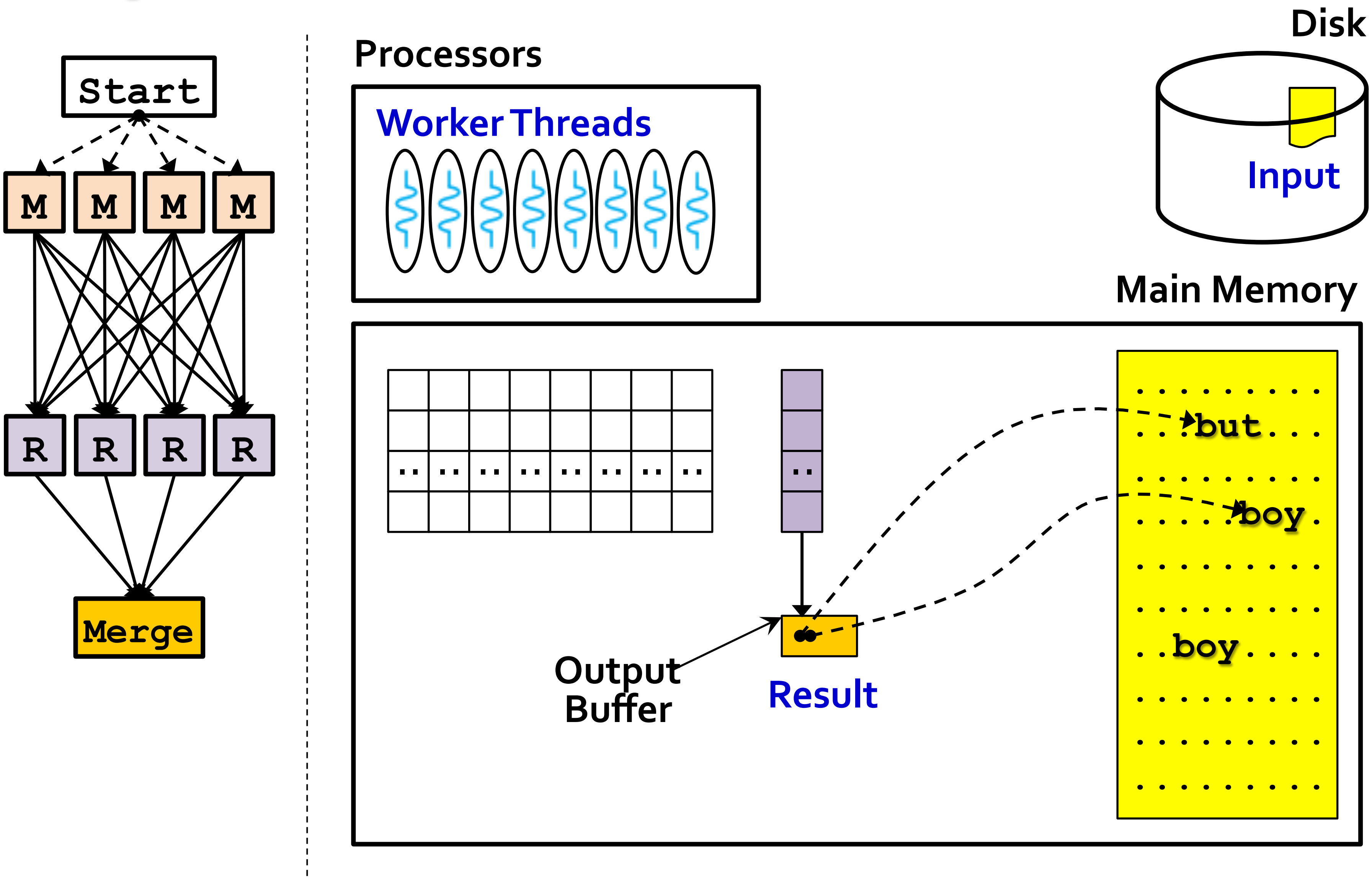


Implementation on Multicore

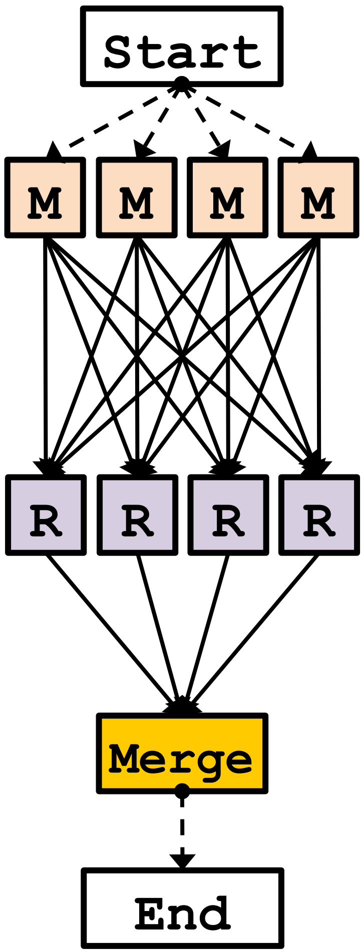


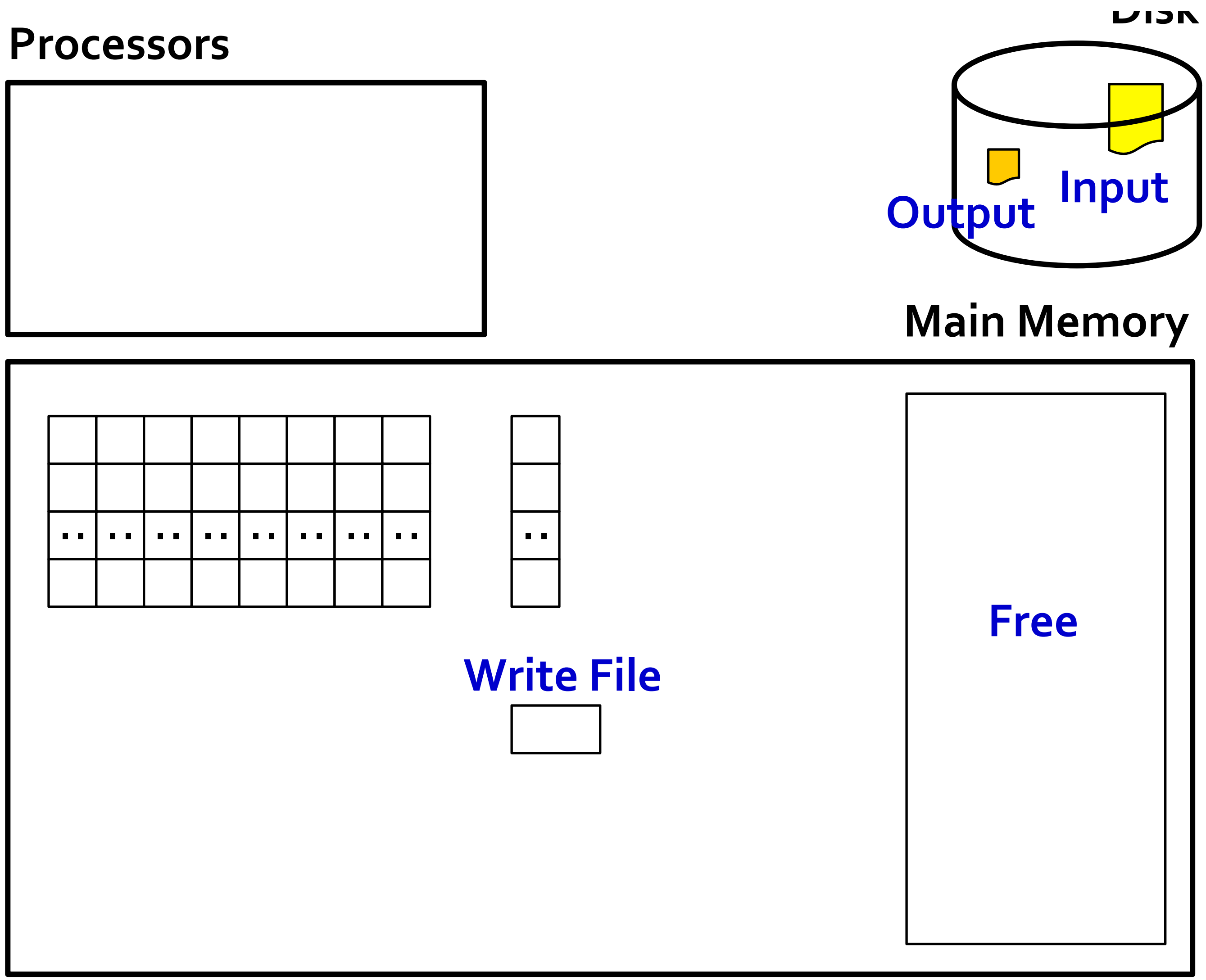


Implementation on Multicore



Implementation on Multicore





Deficiency of MapReduce on Multicore

Deficiency of MapReduce on Multicore

High memory usage

- Keep the whole input data in main memory all the time

  e.g. WordCount with 4GB input requires more than **4.3GB** memory on Phoenix (**93%** used by input data)

Deficiency of MapReduce on Multicore

High memory usage

- Keep the whole input data in main memory all the time

  e.g. WordCount with 4GB input requires more than **4.3GB** memory on Phoenix (**93%** used by input data)

Poor data locality

- Process all input data at one time

  e.g. WordCount with 4GB input has about **25%** L2 cache miss rate

Deficiency of MapReduce on Multicore

High memory usage

- Keep the whole input data in main memory all the time

  e.g. WordCount with 4GB input requires more than **4.3GB** memory on Phoenix (**93%** used by input data)

Poor data locality

- Process all input data at one time

  e.g. WordCount with 4GB input has about **25%** L2 cache miss rate

Strict dependency barriers

- CPU idle at the exchange of phases

Deficiency of MapReduce on Multicore

High memory usage

- Keep the whole input data in main memory all the time

Poor data locality

Solution: Tiled-MapReduce

Stri… arriers

- CPU idle at the exchange of phases

Contribution

Tiled-MapReduce programming model

- Tiling strategy
- Fault tolerance *(in paper)*

Three optimizations for Tiled-MapReduce runtime

- Input Data Buffer Reuse
- NUCA/NUMA-aware Scheduler
- Software Pipeline

Outline

1. Tiled MapReduce
2. Optimization on TMR
3. Evaluation
4. Conclusion

Outline

1. Tiled MapReduce
2. Optimization on TMR
3. Evaluation
4. Conclusion

Tiled-MapReduce

"Tiling Strategy"

- Divide a large MapReduce job into a number of independent small sub-jobs
- Iteratively process one sub-job at a time

Tiled-MapReduce

"Tiling Strategy"

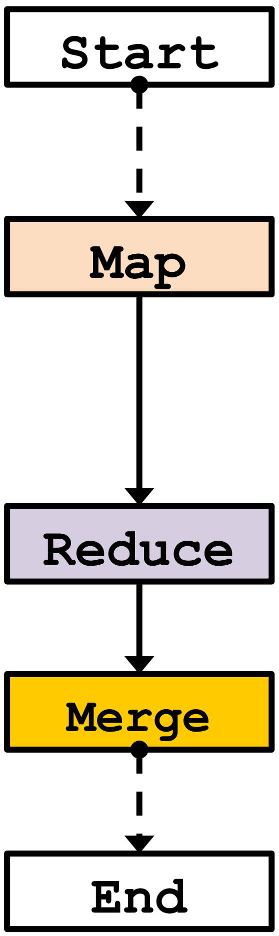
- Divide a large MapReduce job into a number of independent small sub-jobs
- Iteratively process one sub-job at a time

Requirement

- ***Reduce*** function must be ***Commutative*** and ***Associative***
  - all 26 applications in the test suit of *Phoenix* and *Hadoop* meet the requirement

Tiled-MapReduce

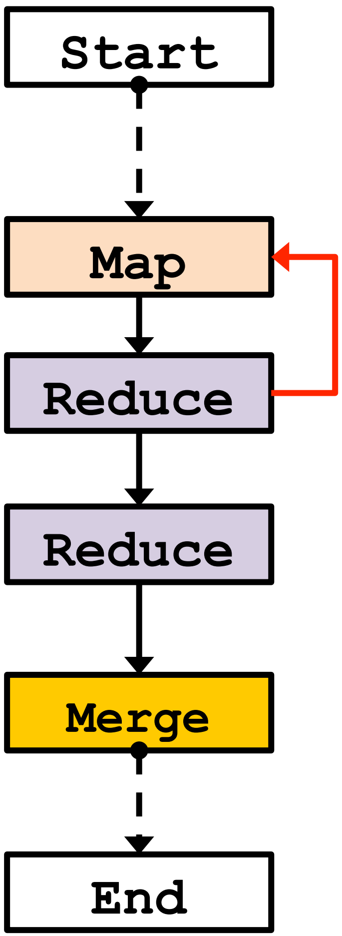
Extensions to MapReduce Model



Tiled-MapReduce

Extensions to MapReduce Model

1. Replace the Map phase with a loop of Map and Reduce phases

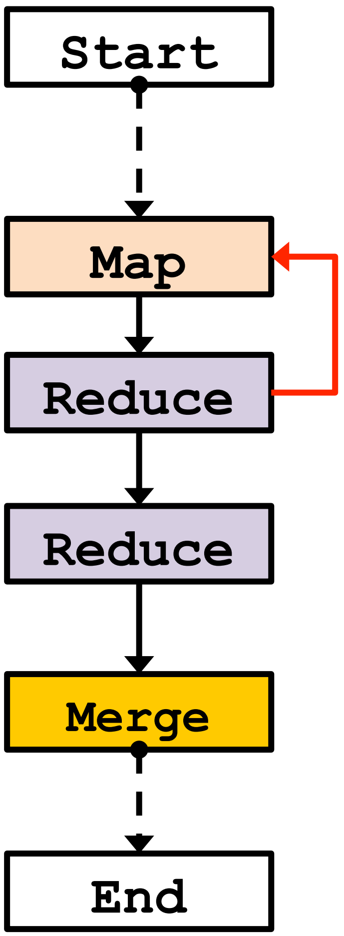


Tiled-MapReduce

Extensions to MapReduce Model

1. Replace the Map phase with a loop of Map and Reduce phases

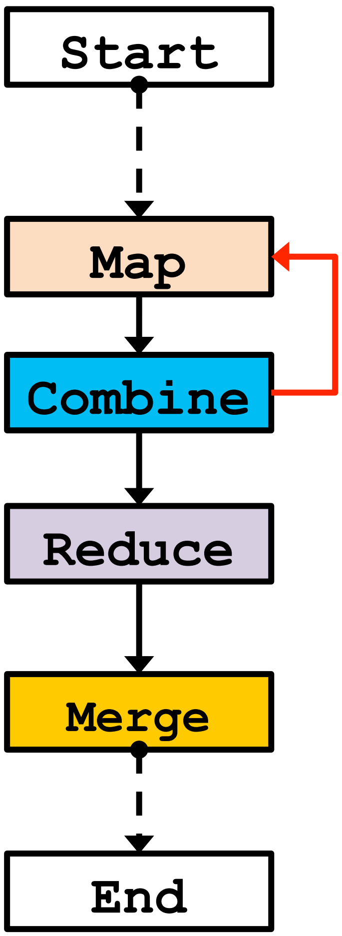
2. Process one sub-job in each iteration



Tiled-MapReduce

Extensions to MapReduce Model

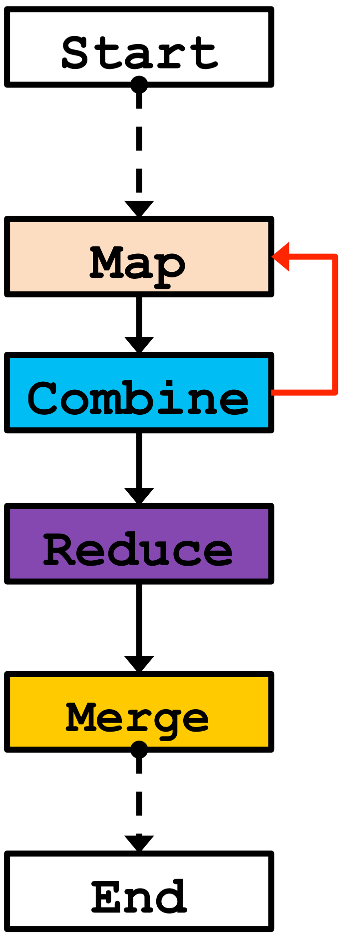
1. Replace the Map phase with a loop of Map and Reduce phases
2. Process one sub-job in each iteration
3. Rename the Reduce phase within loop to the Combine phase



Tiled-MapReduce

Extensions to MapReduce Model

1. Replace the Map phase with a loop of Map and Reduce phases
2. Process one sub-job in each iteration
3. Rename the Reduce phase within loop to the Combine phase
4. Modify the Reduce phase to process the partial results of all iterations



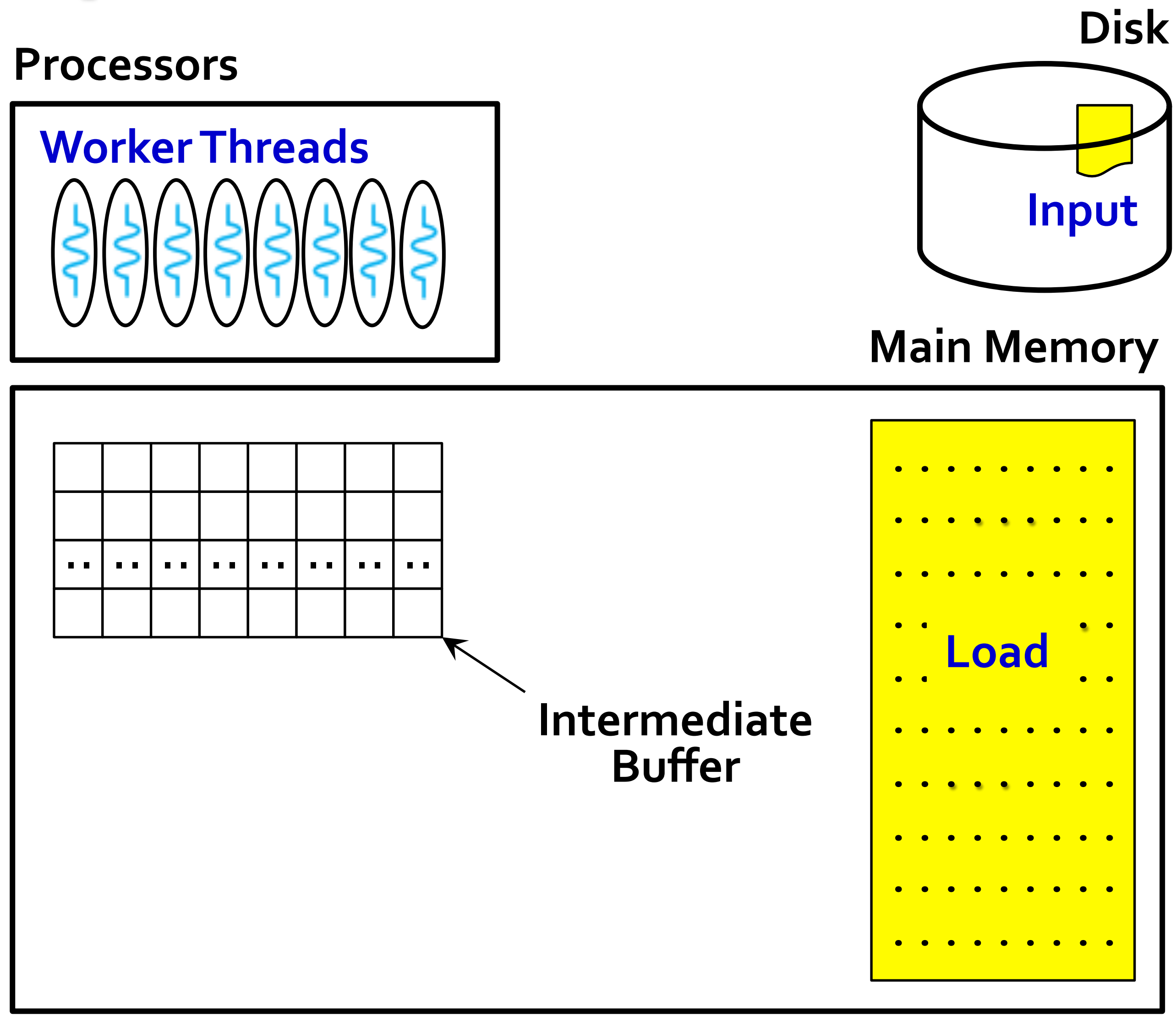
Prototype of Tiled-MapReduce

Ostrich: a prototype of Tiled-MapReduce programming model

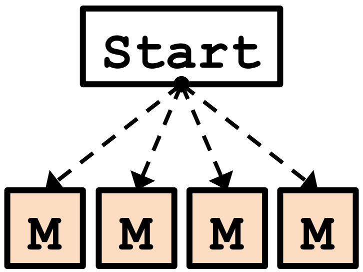
- Demonstrate the effectiveness of TMR programming model
- Base on *Phoenix* runtime
- Follow the *data structure* and *algorithms*

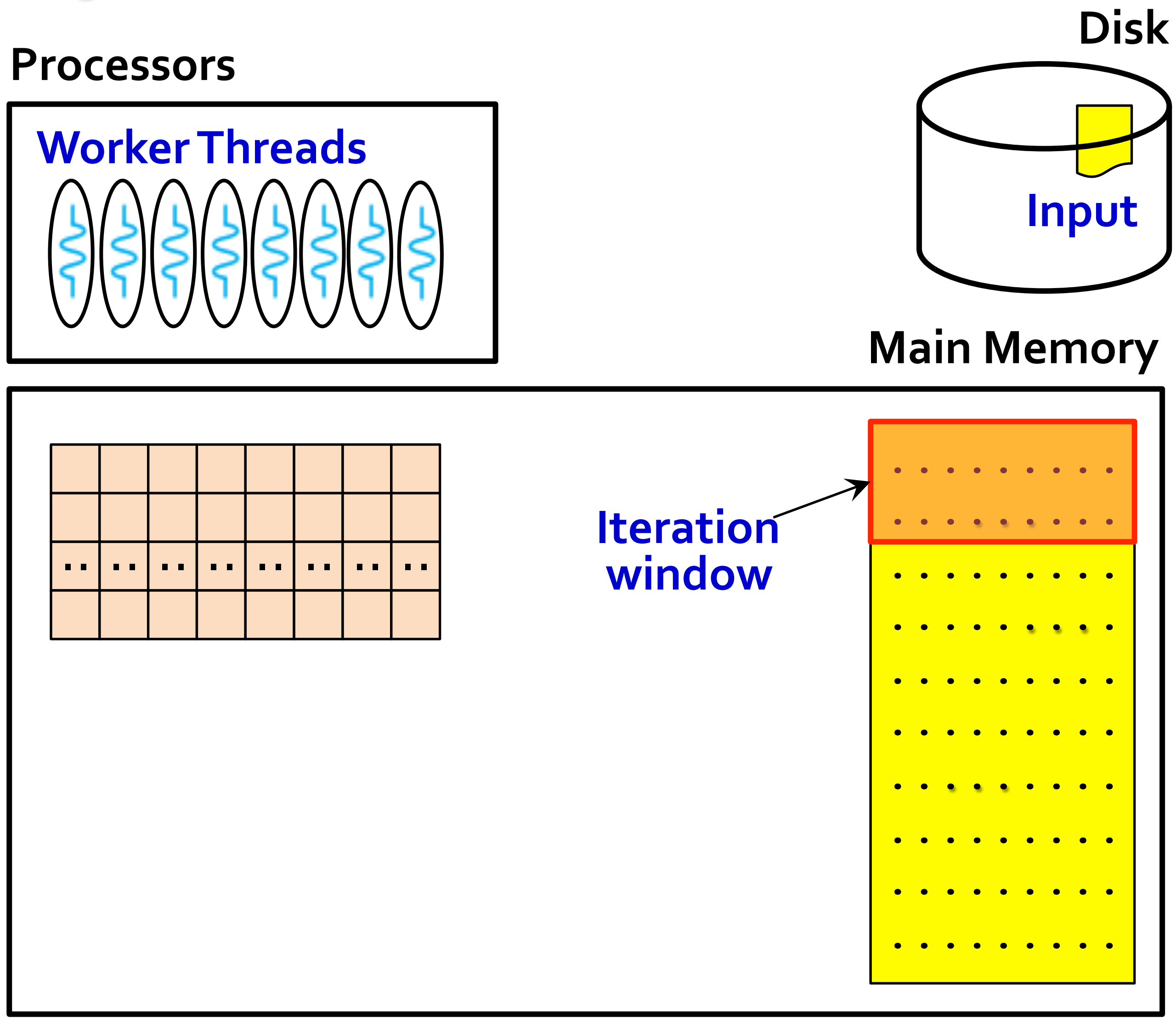
Ostrich Implementation

`Start`

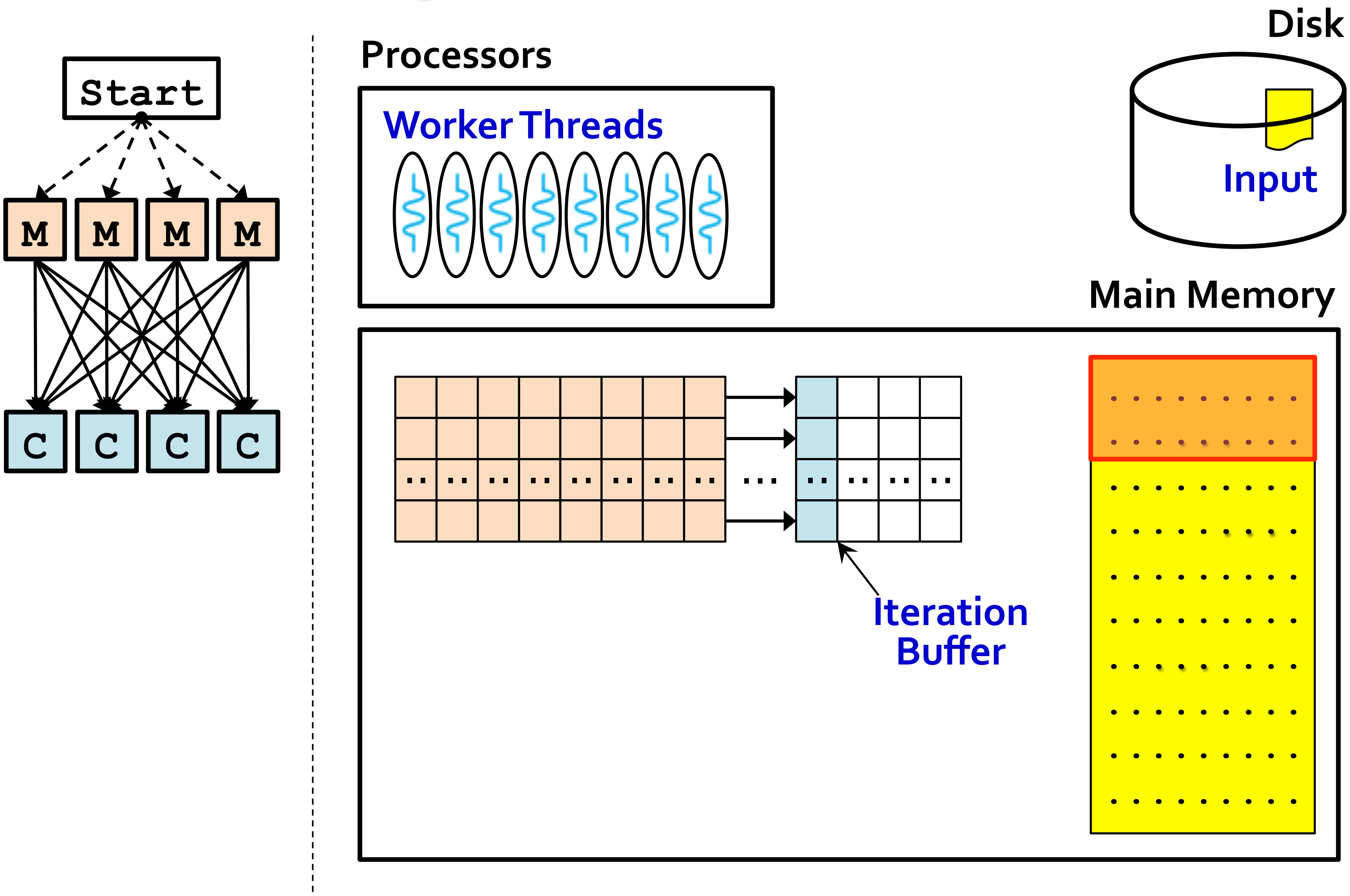


Ostrich Implementation

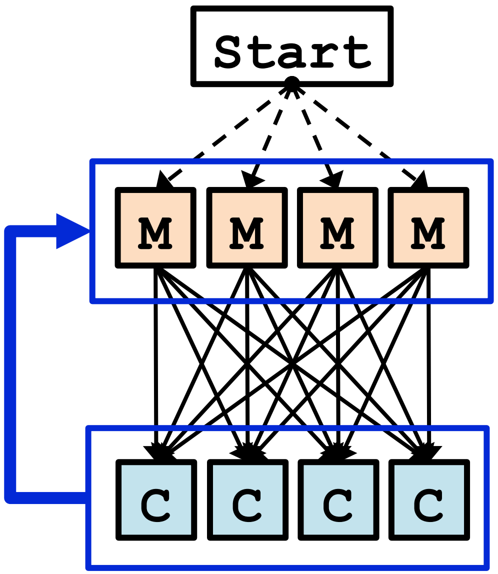


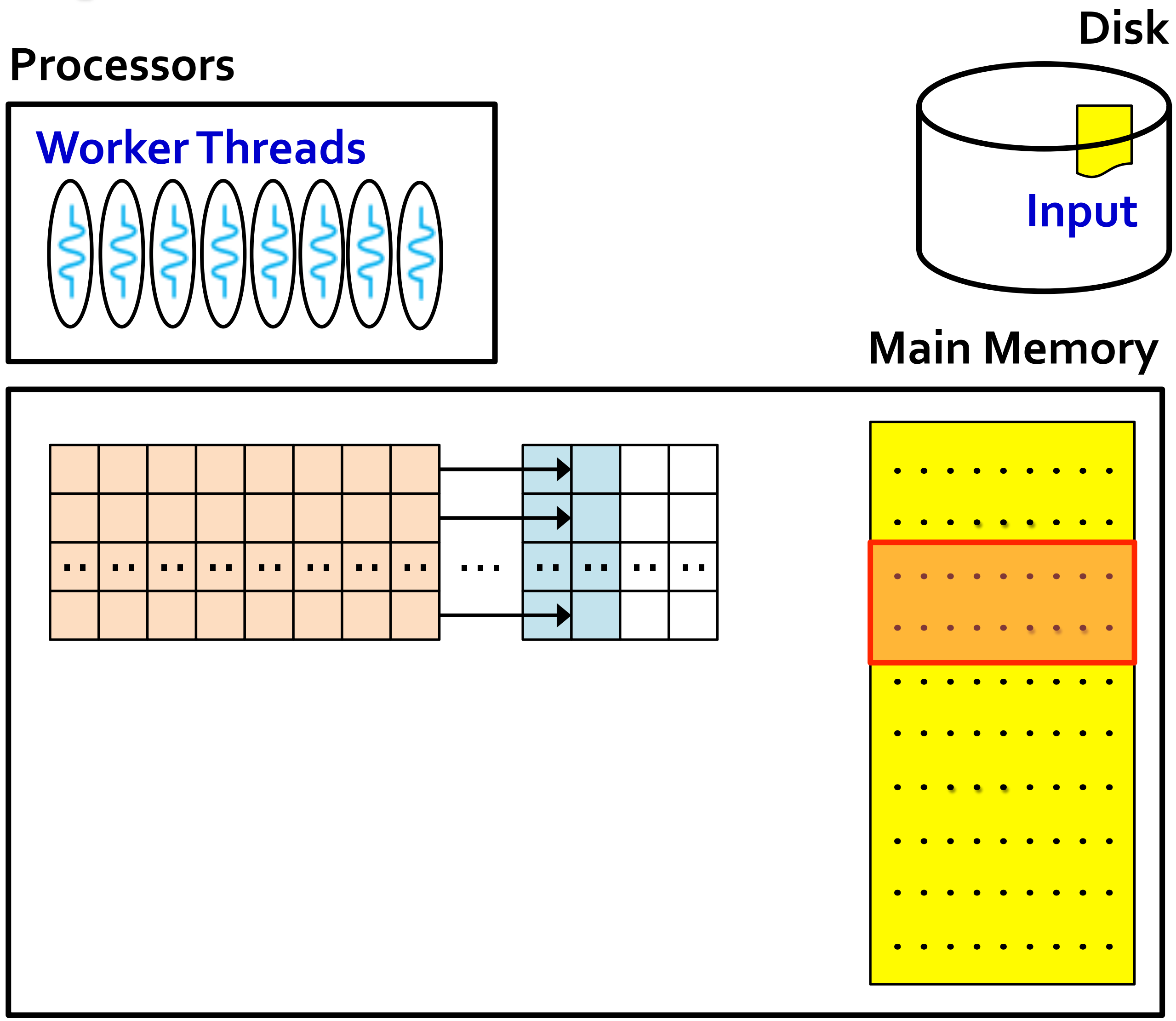


Ostrich Implementation

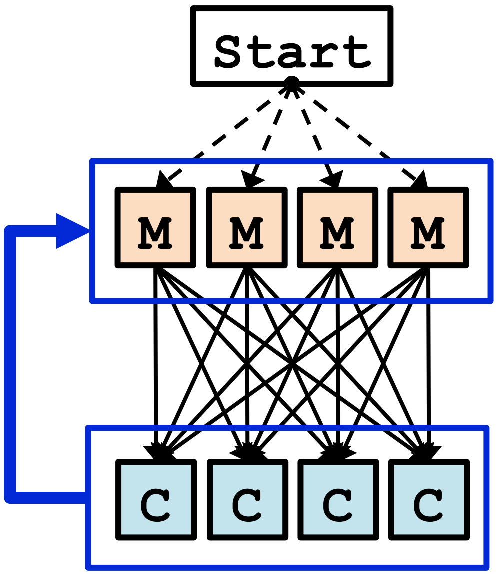


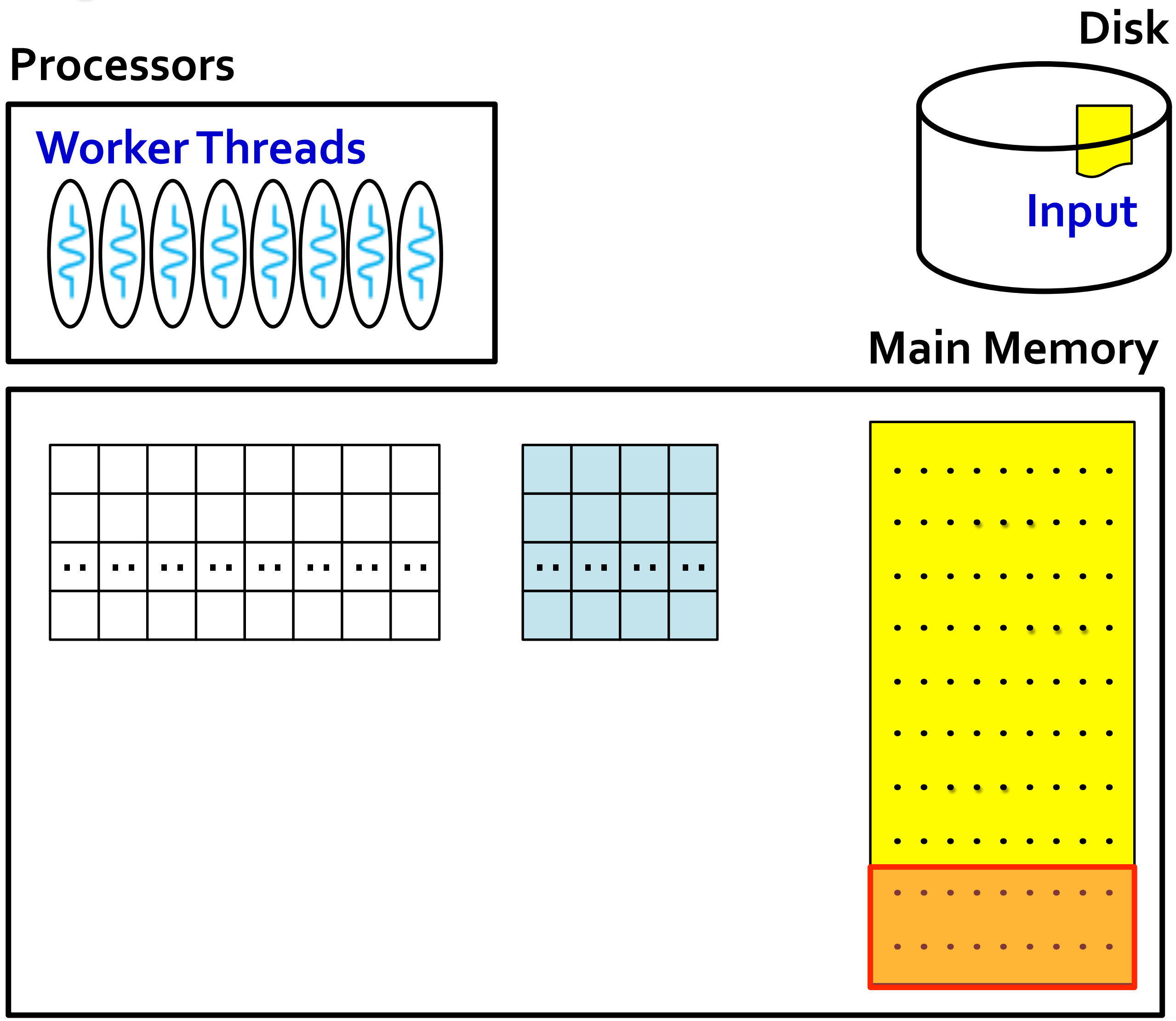
Ostrich Implementation



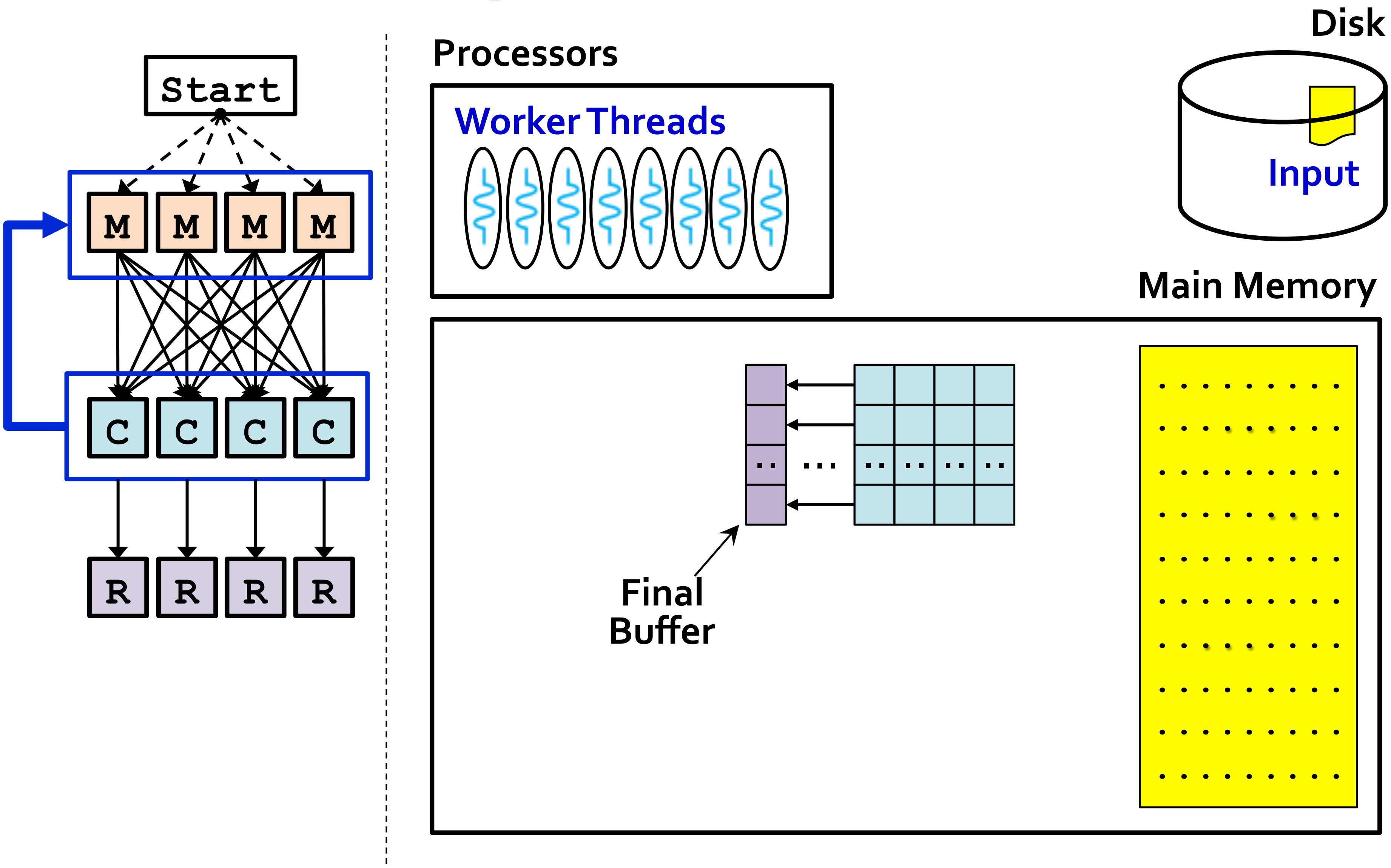


Ostrich Implementation

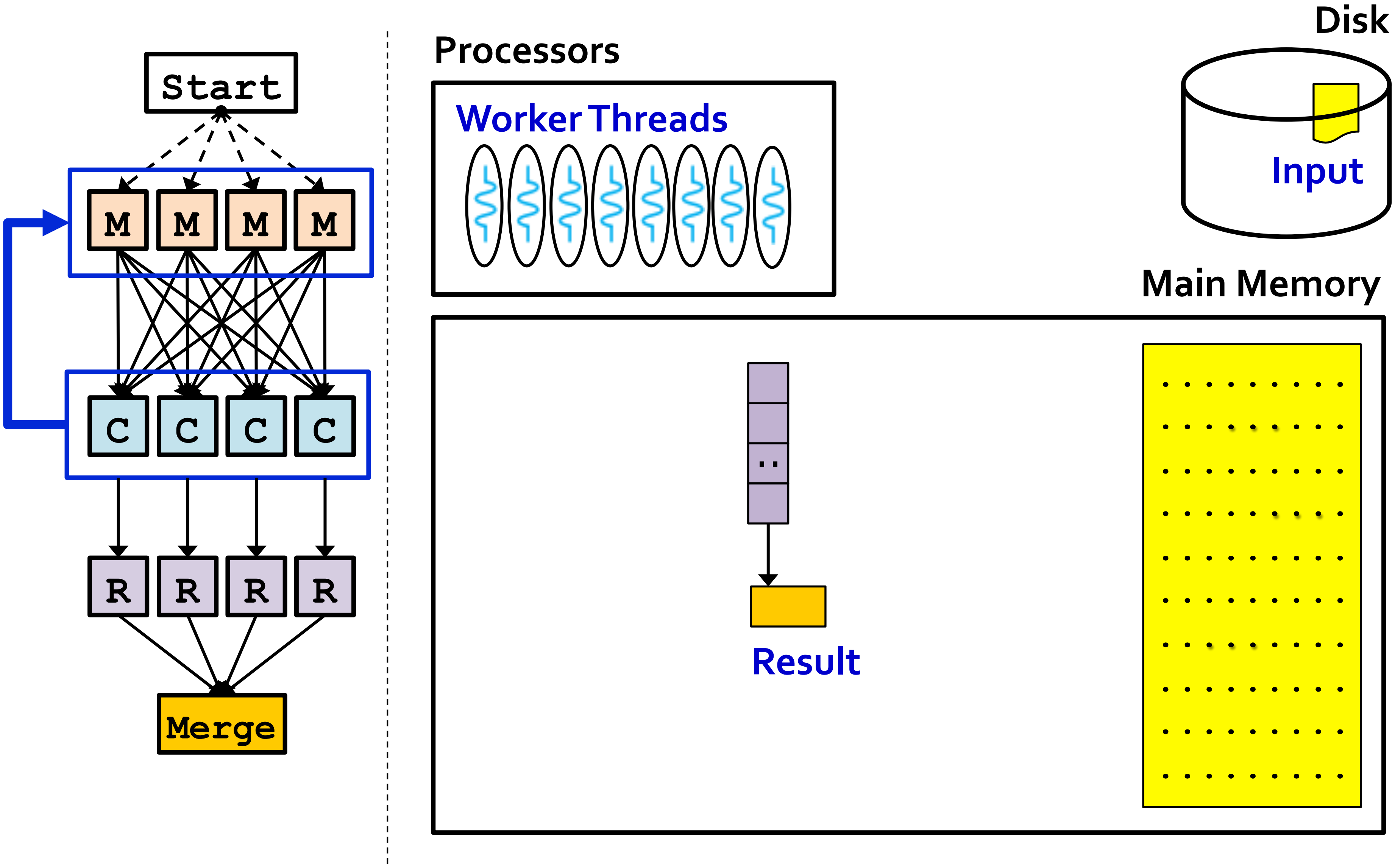




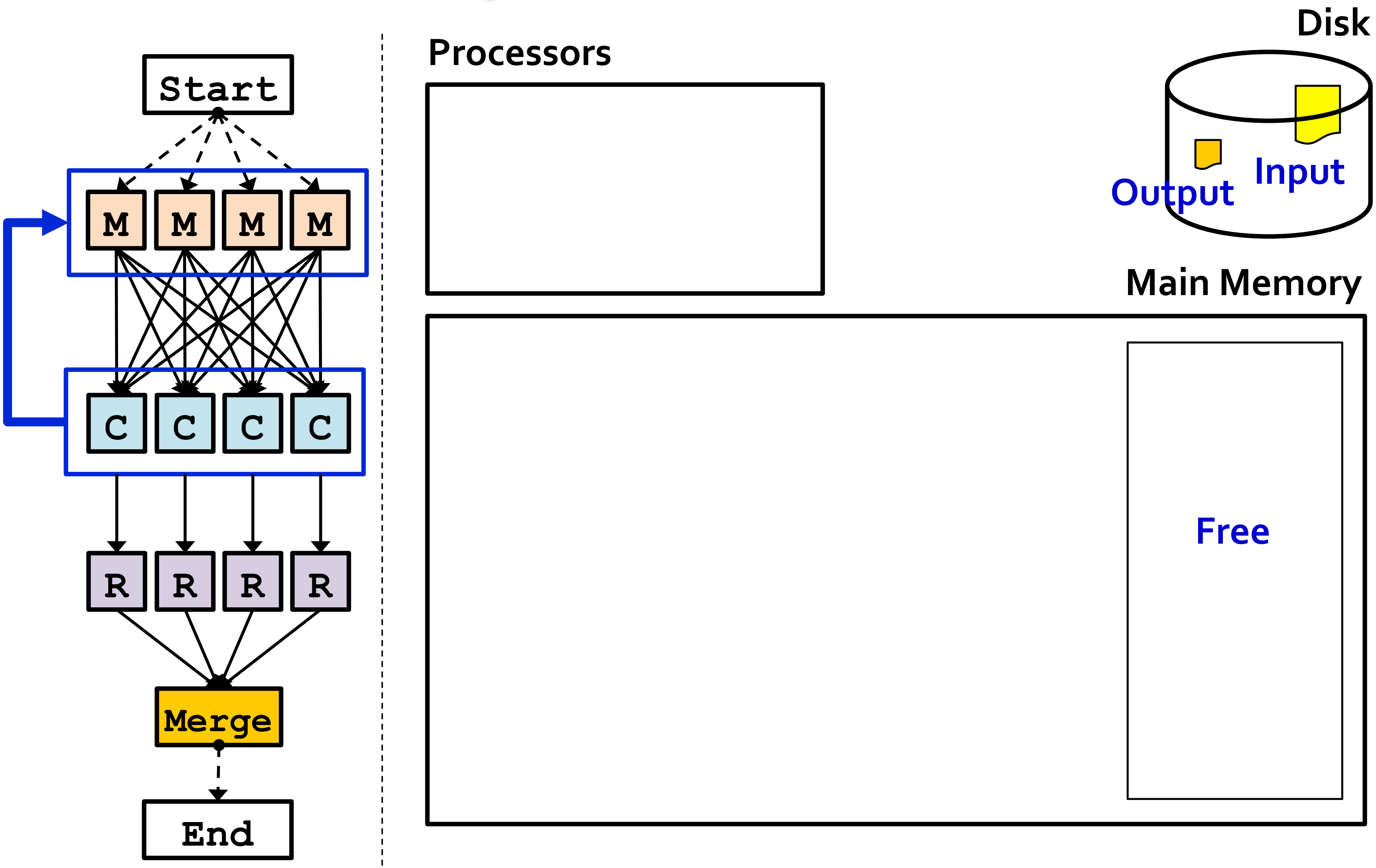
Ostrich Implementation



Ostrich Implementation



Ostrich Implementation



Outline

1. Tiled MapReduce
2. Optimization on TMR
3. Evaluation
4. Conclusion

OPT1: MEMORY REUSE

OPT1: Memory Reuse

High Memory Usage

- Keep the whole input data in memory during the entire lifecycle

OPT1: Memory Reuse

High Memory Usage

- Keep the whole input data in memory during the entire lifecycle

Observation

- Only few data in input data is necessary

  `e.g. WordCount: 1 copy for all duplicated words`

OPT1: Memory Reuse

High Memory Usage

- Keep the whole input data in memory during the entire lifecycle

Observation

- Only few data in input data is necessary

  `e.g. WordCount: 1 copy for all duplicated words`

- The aggregation of these data improves data locality

OPT1: Memory Reuse

Input Data Memory Reuse

- Copy necessary data to a new buffer in each *Combine* phase
- Only hold the input data of current sub-job in memory
- Reuse the Input Buffer among sub-jobs

OPT1: Memory Reuse

Extension of Interface

- Provide 2 optional interfaces

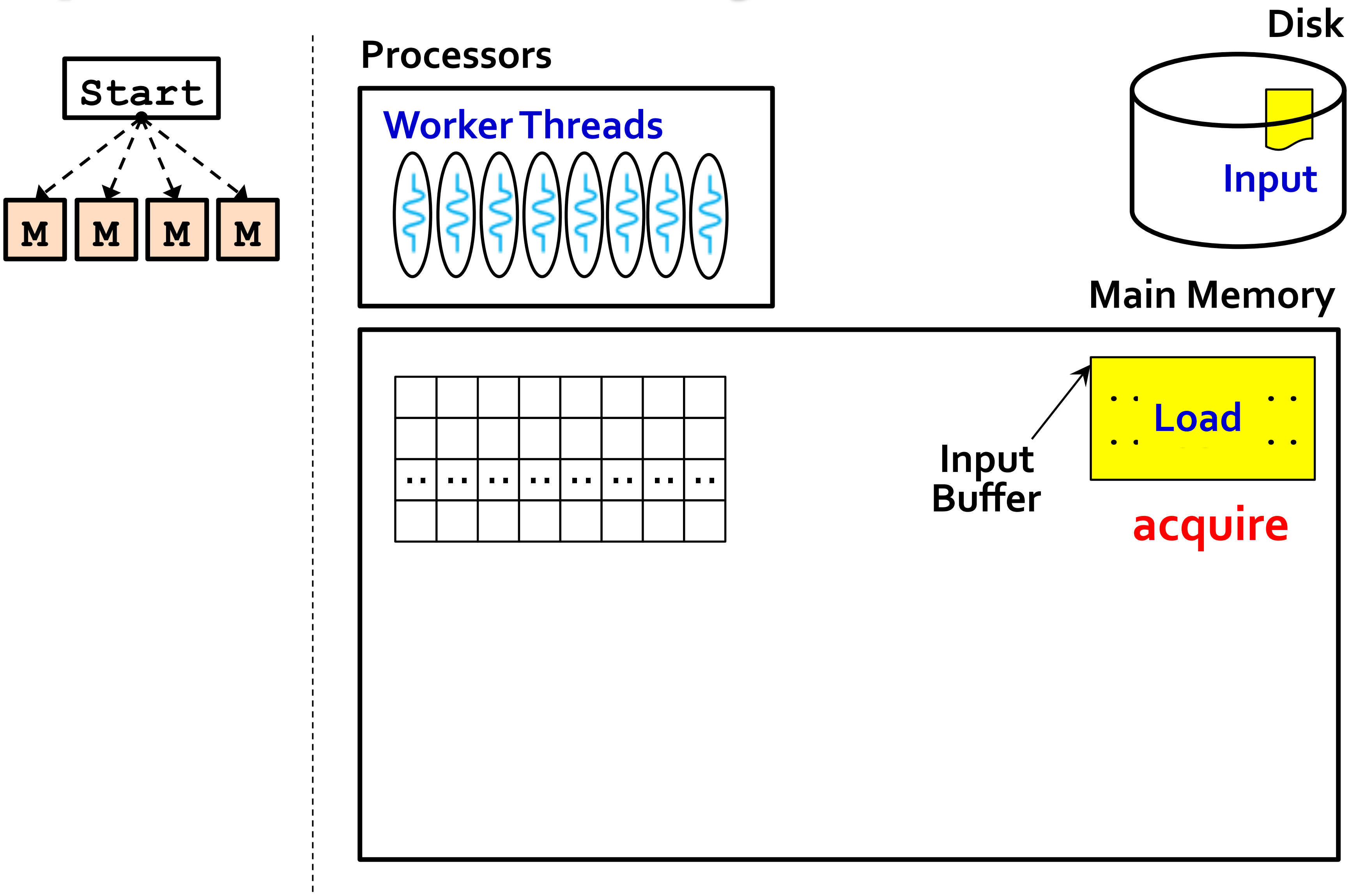
  Acquire: `load input data to memory`

  Release: `free input data from memory`

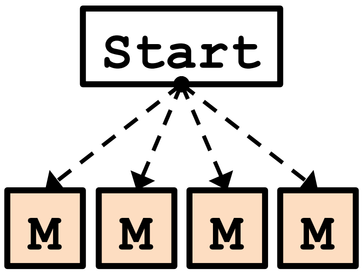
- The counterparts in other runtimes

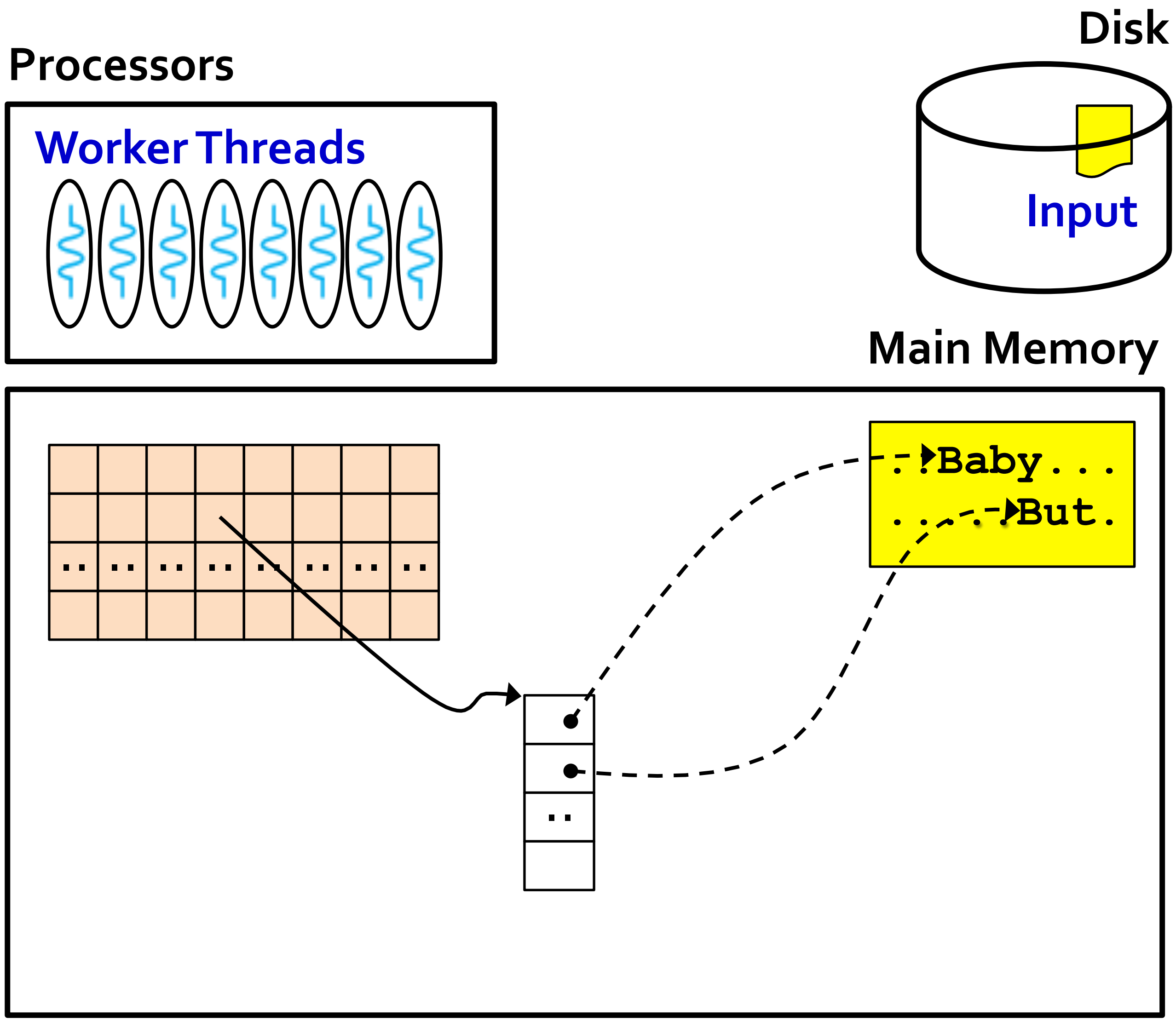
Runtime	Interface	
Ostrich	acquire	release
Google MapReduce	reader	writer
Hadoop	constructor	close

Input Data Memory Reuse

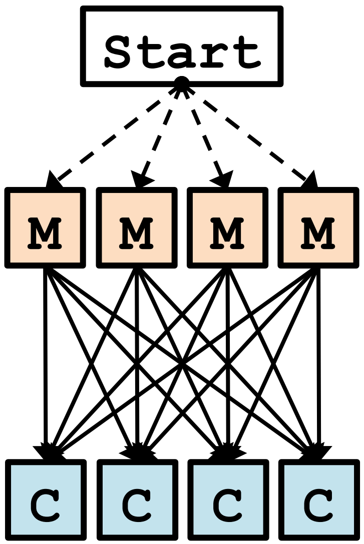


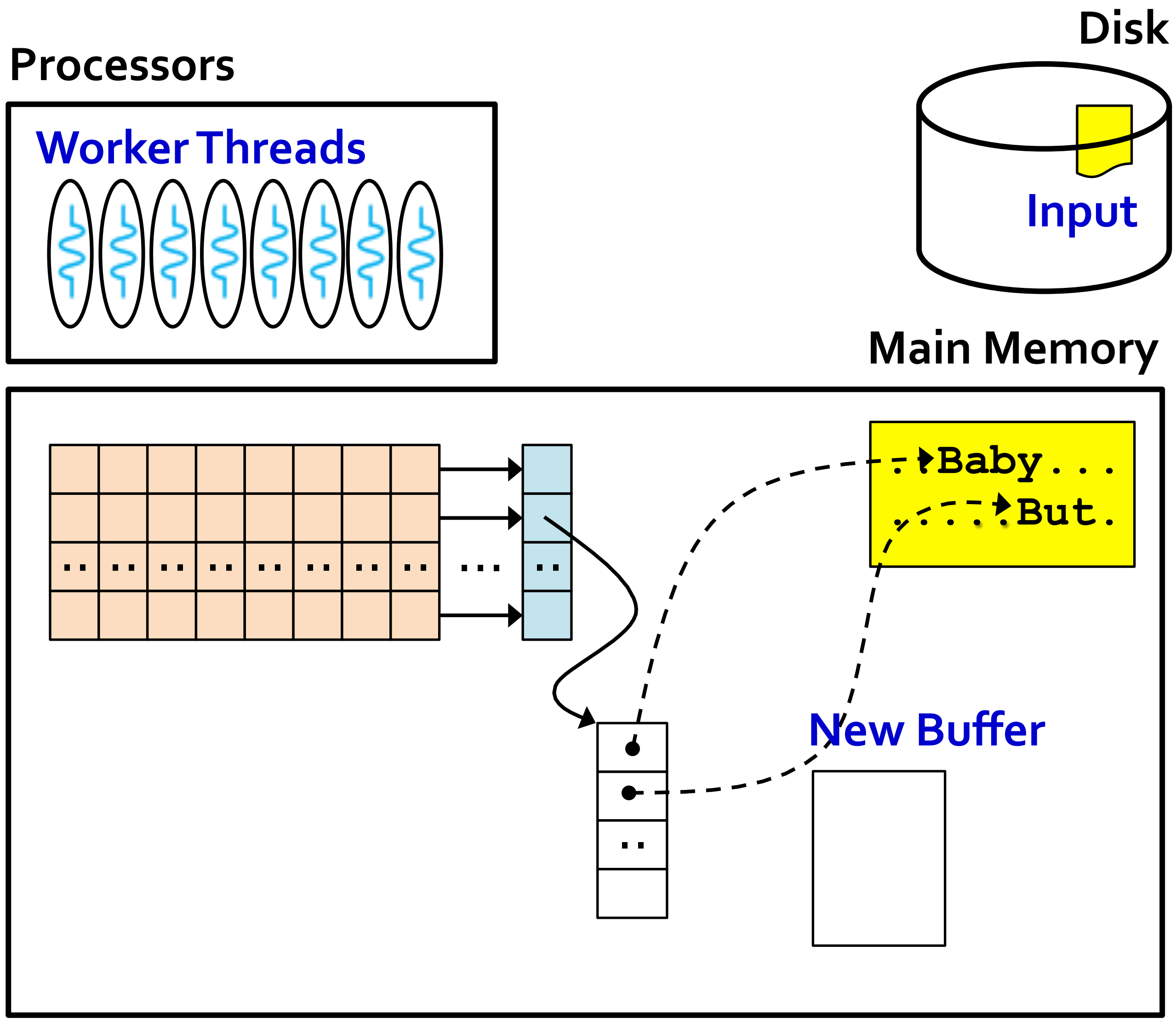
Input Data Reuse



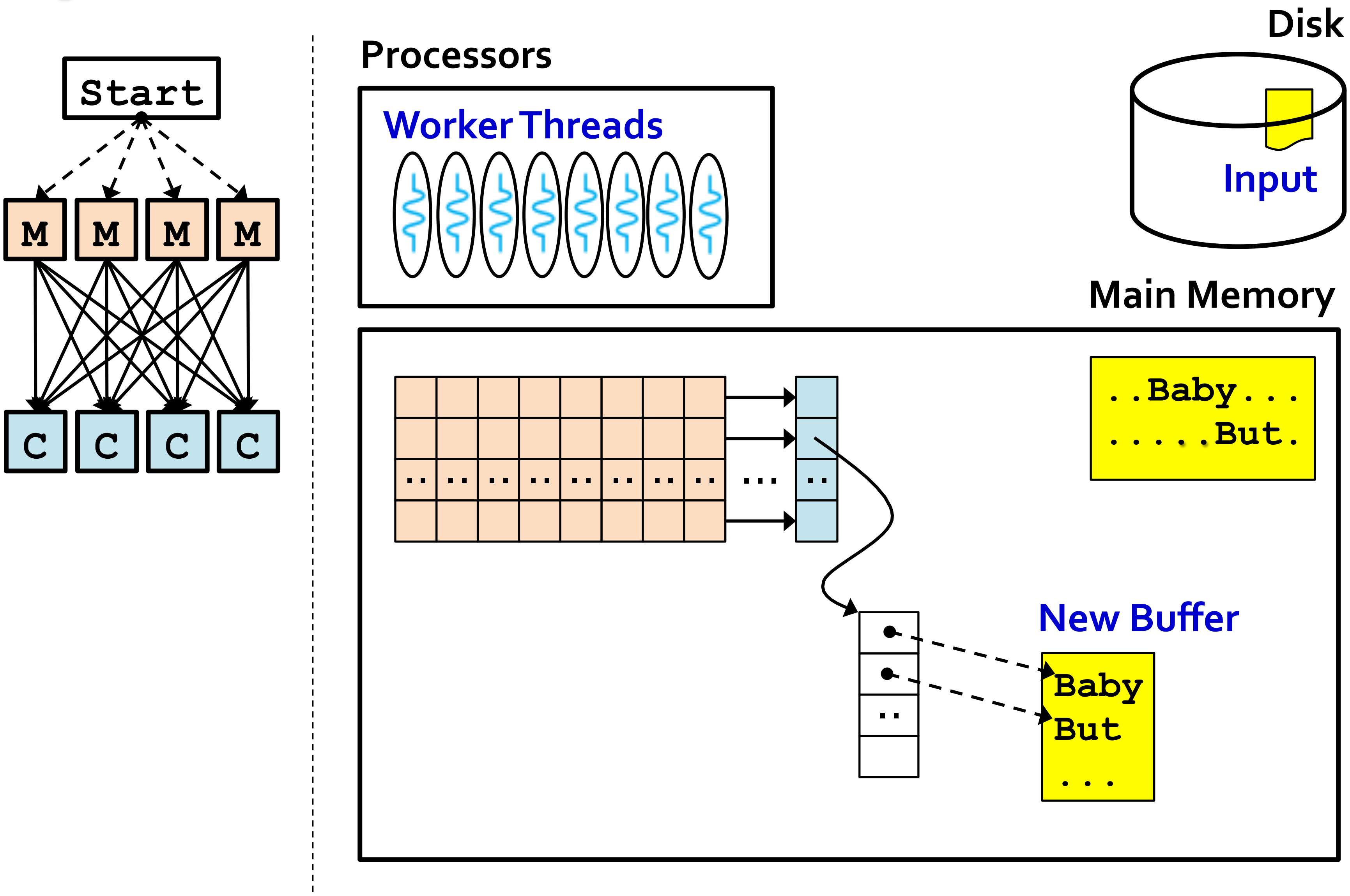


Input Data Reuse

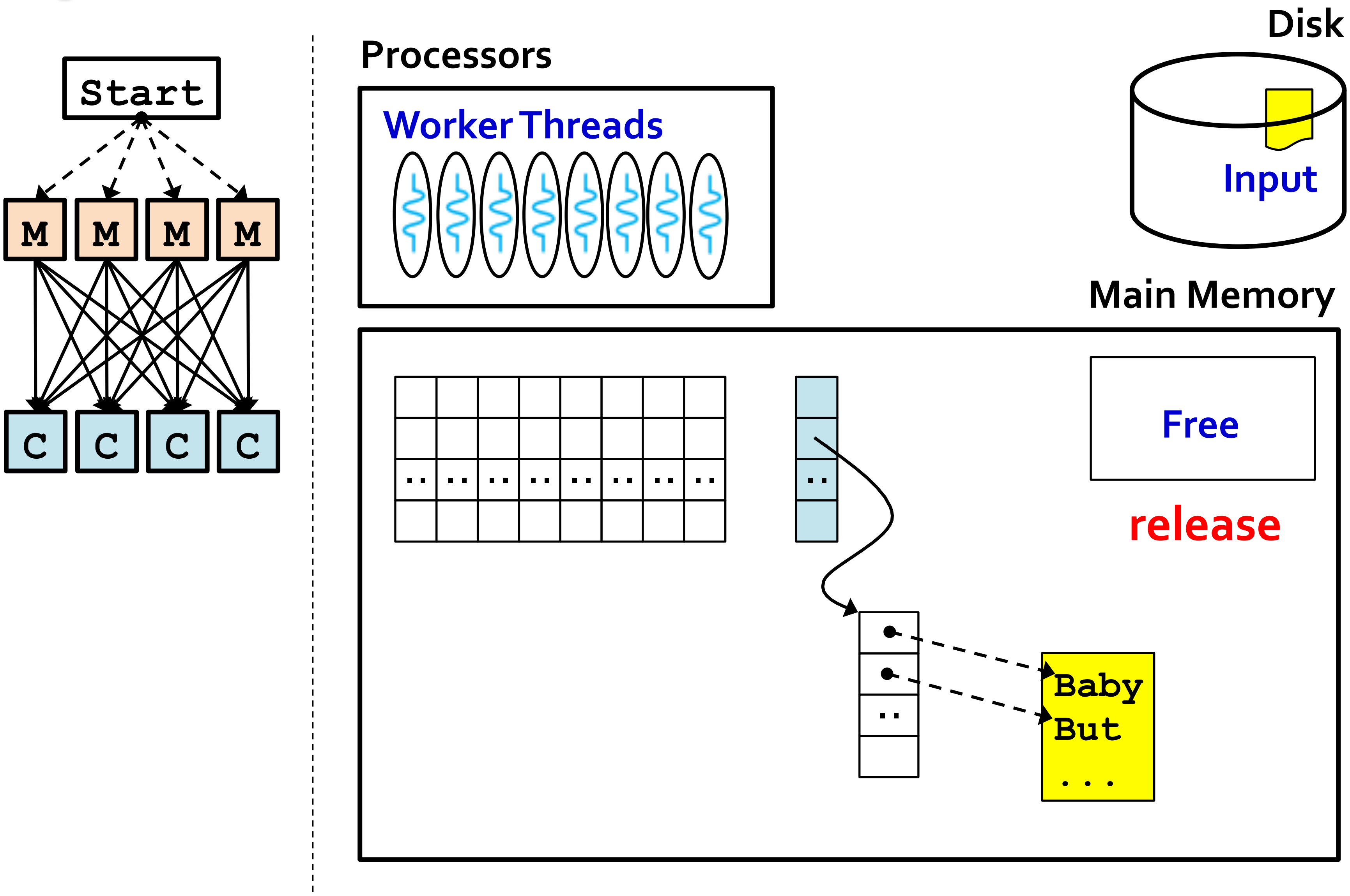




Input Data Reuse



Input Data Reuse



OPT2: LOCALITY OPTIMIZATION

OPT2: Locality Optimization

Poor Data Locality of MapReduce runtime on Multicore

- Process all input data in one time

OPT2: Locality Optimization

Poor Data Locality of MapReduce runtime on Multicore

- Process all input data in one time

Tiled-MapReduce improves data locality

- Make the working set of each sub-job fit into the last level cache
- Aggregate partial results in *Combine* phase (in OPT1)

OPT2: Locality Optimization

Memory Hierarchy

- Multicore hardware usually organizes caches in a non-uniform cache access (NUCA) way
- The cross-chip operations are expensive*

`e.g. Local/Remote L2 cache: 14/110 cycles*`

* Intel 16-Core Machine with 4 Xeon 1.6GHz Quad-cores chips

OPT2: Locality Optimization

Memory Hierarchy

- Multicore hardware usually organizes caches in a non-uniform cache access (NUCA) way
- The cross-chip operations are expensive*

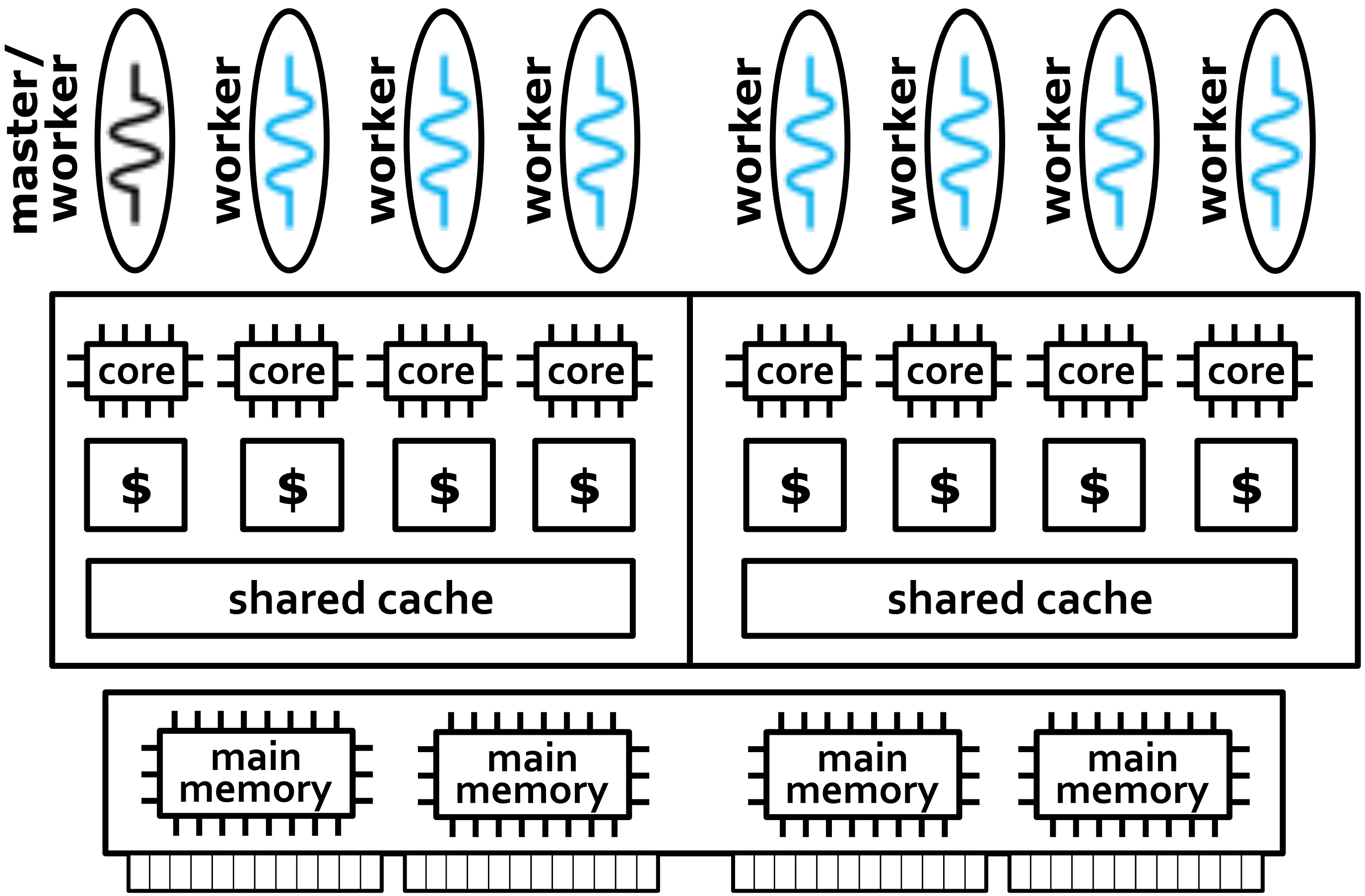
`e.g. Local/Remote L2 cache: 14/110 cycles*`

NUCA/NUMA-aware scheduler

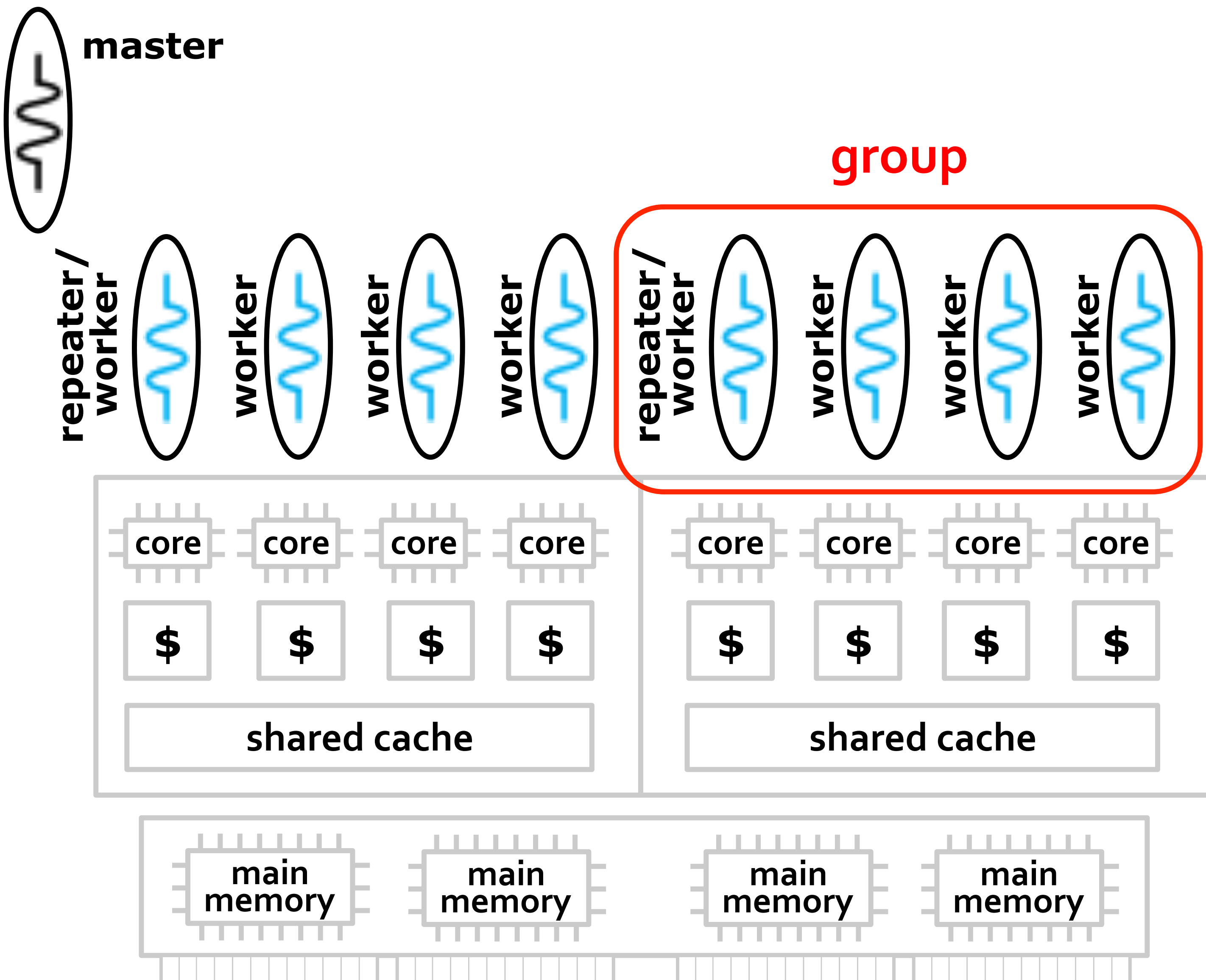
- Eliminate remote cache and memory access
- Run each sub-job on a single chip

* Intel 16-Core Machine with 4 Xeon 1.6GHz Quad-cores chips

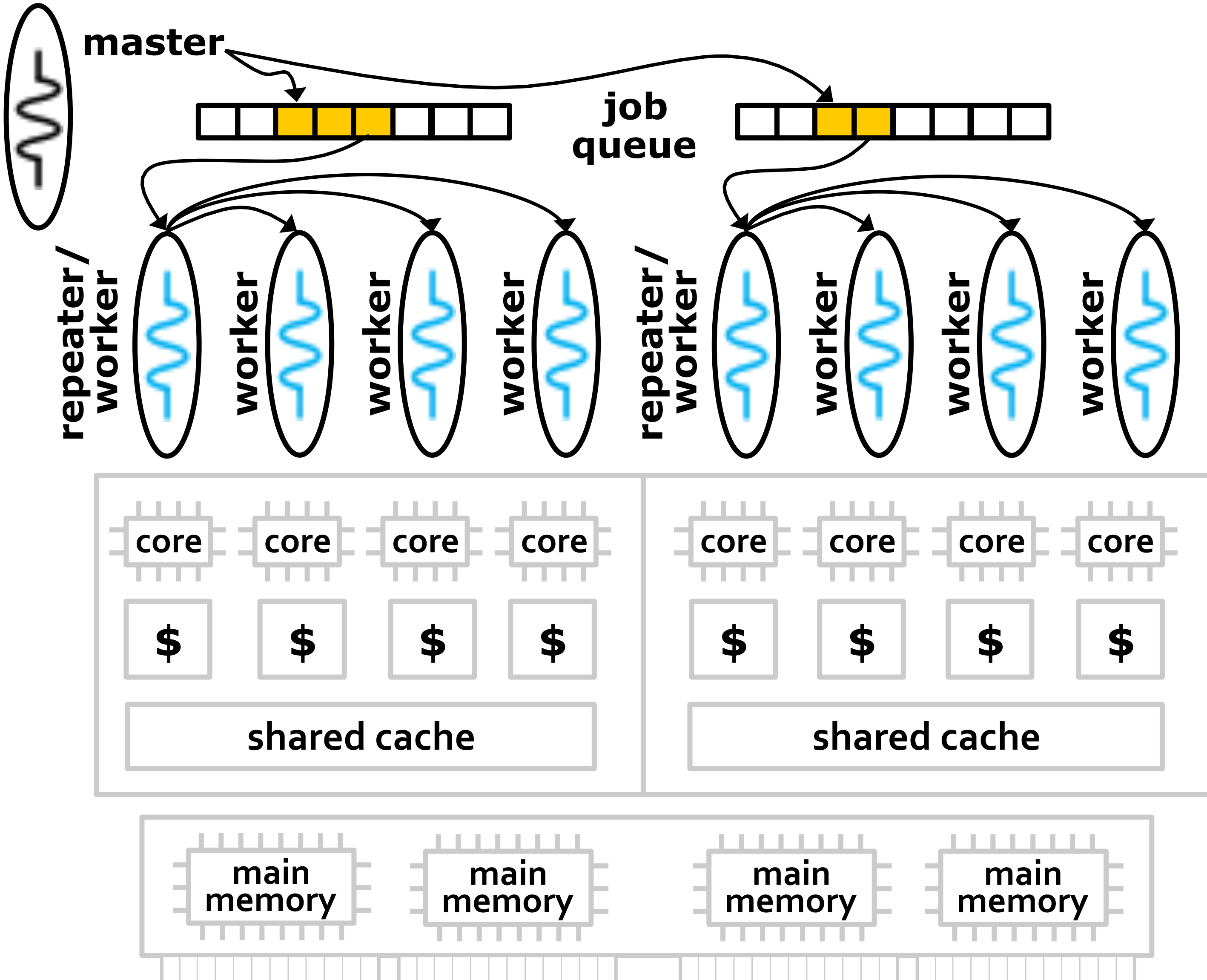
NUCA/NUMA-Aware Scheduler



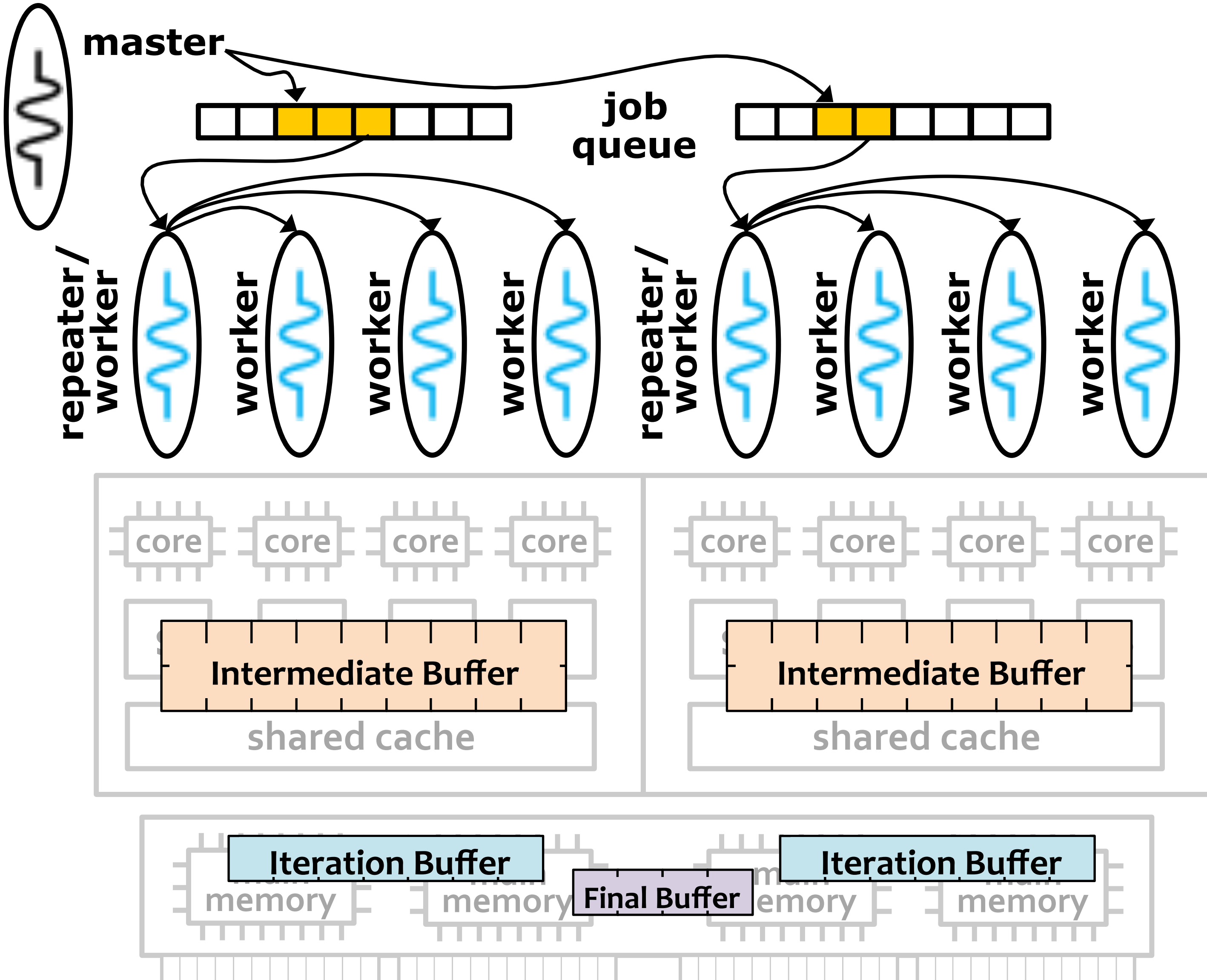
NUCA/NUMA-Aware Scheduler



NUCA/NUMA-Aware Scheduler



NUCA/NUMA-Aware Scheduler



OPT3: CPU OPTIMIZATION

OPT3: CPU Optimization

Data Dependency

- Strict barrier after map and reduce phase
- The execution time of a job is determined by the slowest worker in each phase

OPT3: CPU Optimization

Data Dependency

- Strict barrier after map and reduce phase
- The execution time of a job is determined by the slowest worker in each phase

Observation

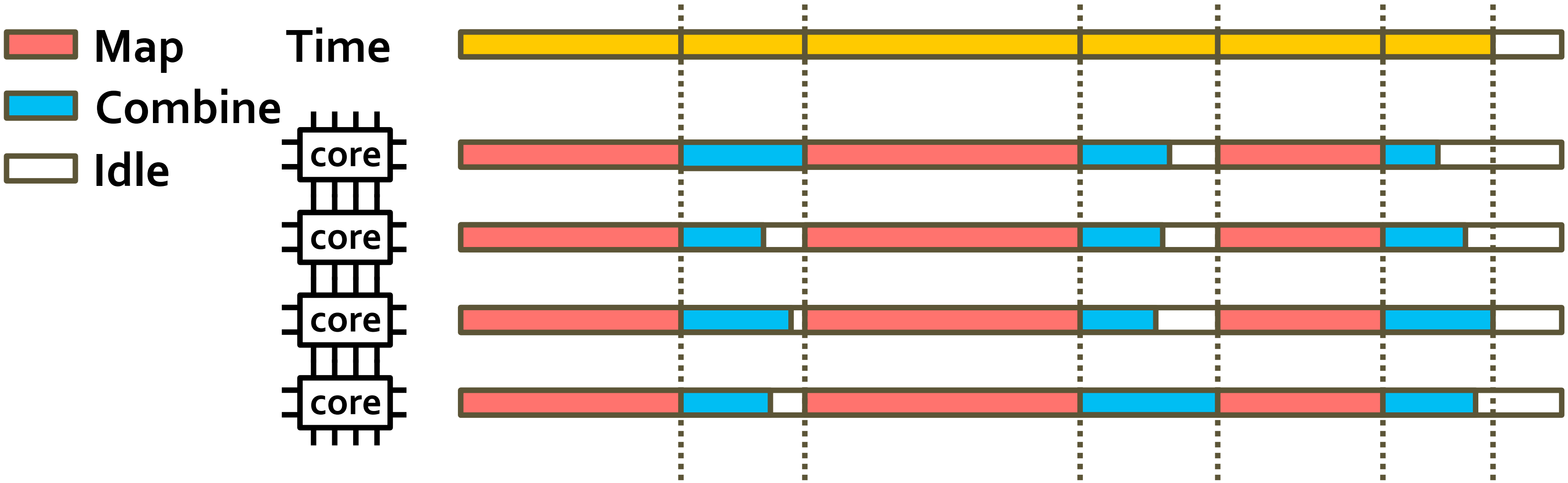
- No data dependency between one sub-job's *Combine* phase and its *successor's* *Map* phase

OPT3: CPU Optimization

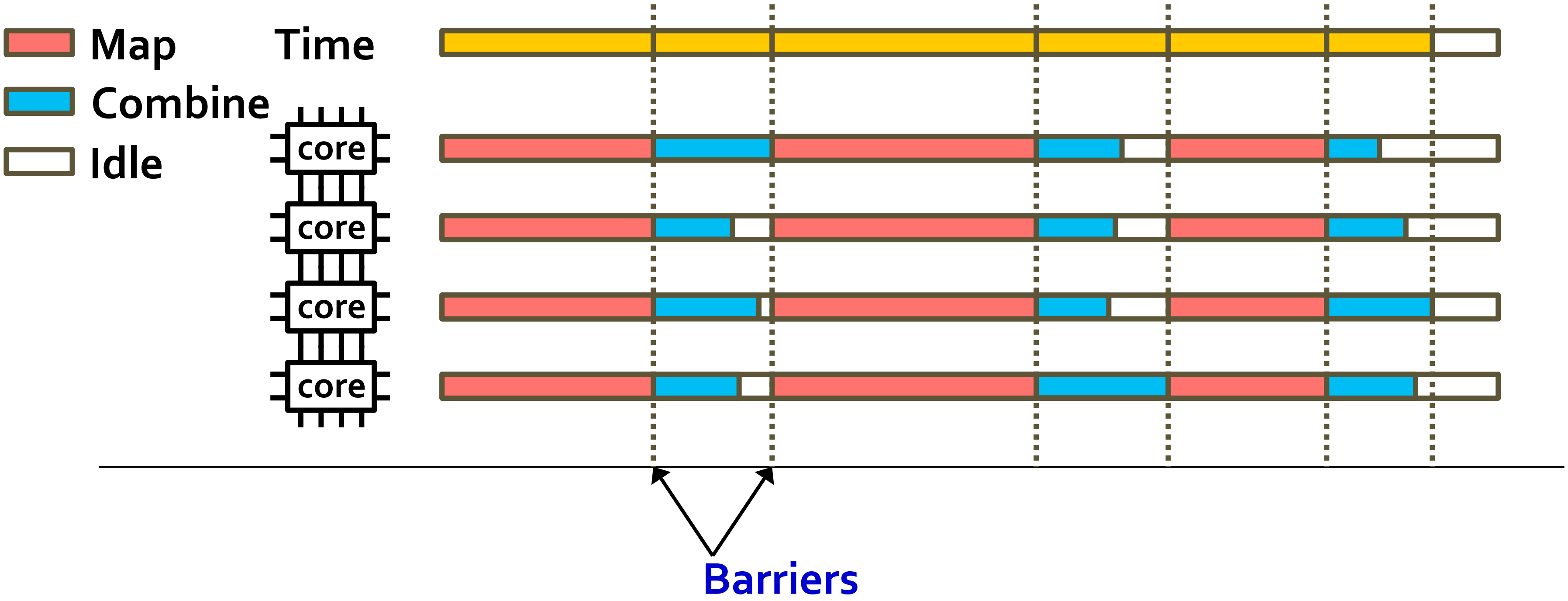
Software Pipeline

- Overlap the ***Combine*** phase of the current sub-job and the ***Map*** phase of its successor

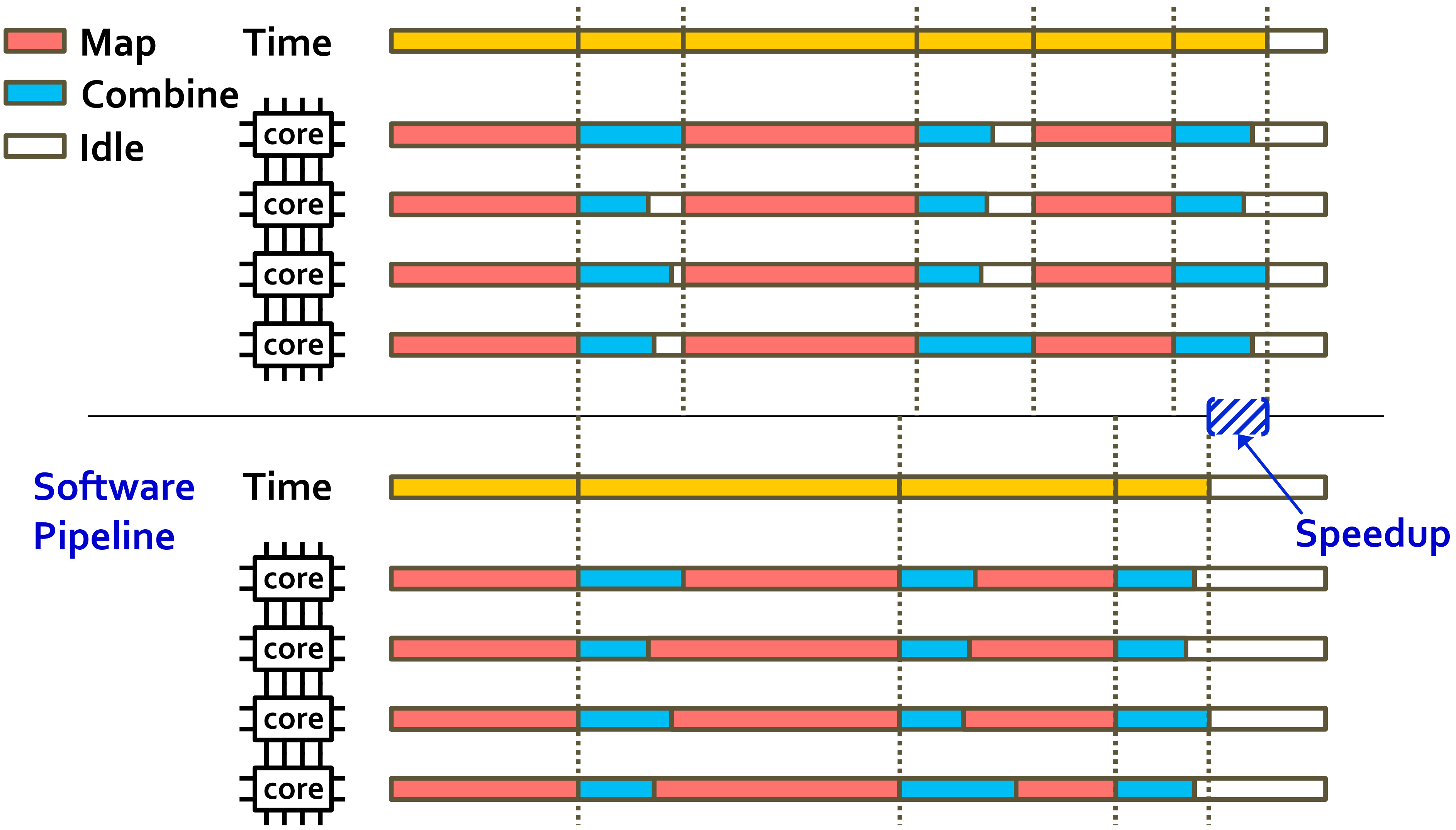
Software Pipeline



Software Pipeline



Software Pipeline



Outline

1. Tiled MapReduce
2. Optimization on TMR
3. Evaluation
4. Conclusion

Configuration

Platform

Intel 16-Core machine (4 Quad-cores chips)

32GB Main Memory

Debian Linux with kernel v2.6.24

Systems:

Phoenix-2 with streamflow *

Ostrich with streamflow

* **Scalable locality-conscious multithreaded memory allocation - ISMM'06**

Configuration

Applications

Applications	Key	Duplicate
WordCount (WC)	many	many
Distributed Sort (DS)	many	no
Log Statistics (LS)	few	many
Inverted Index (II)	one	few

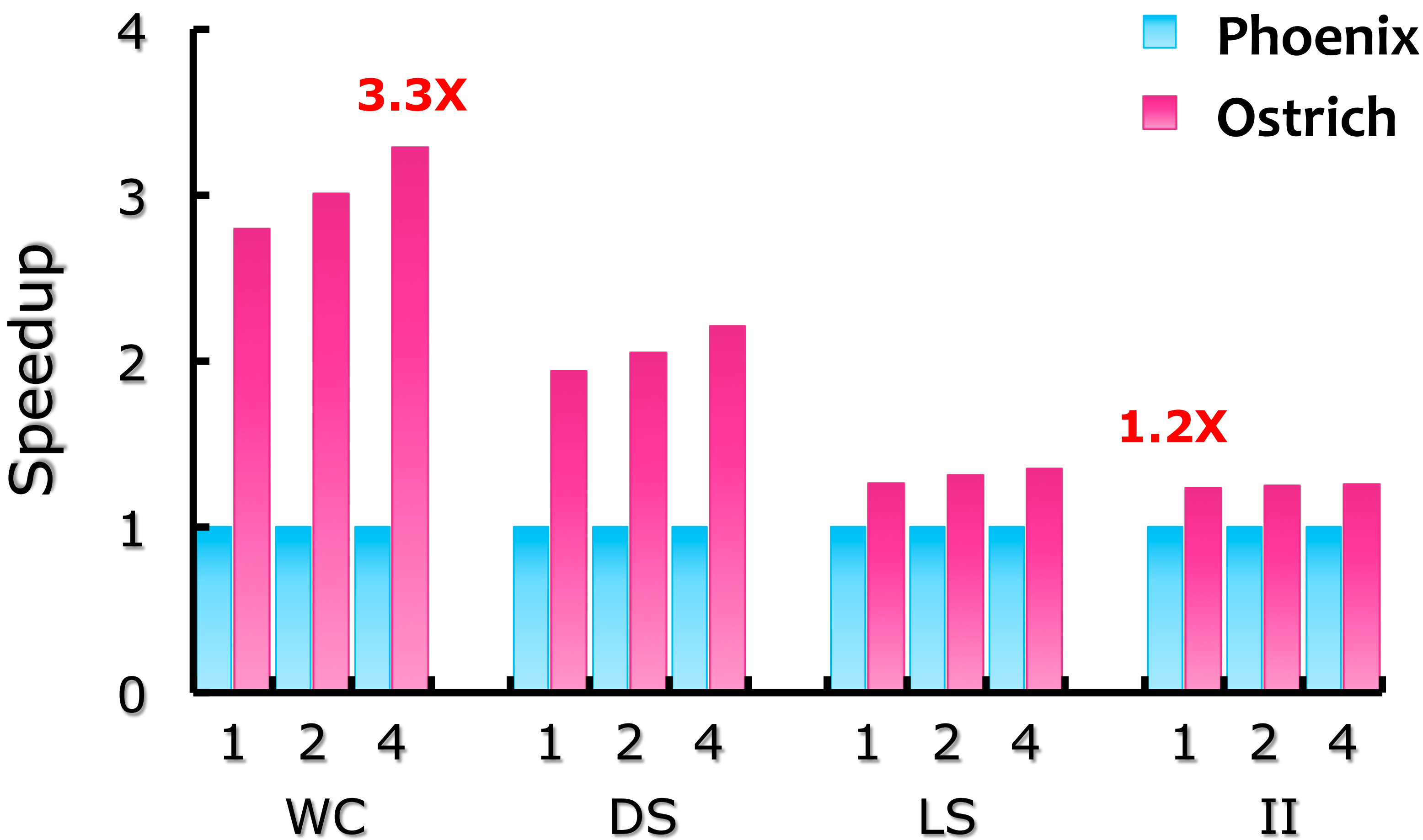
Burden of Programmer

Code Modification

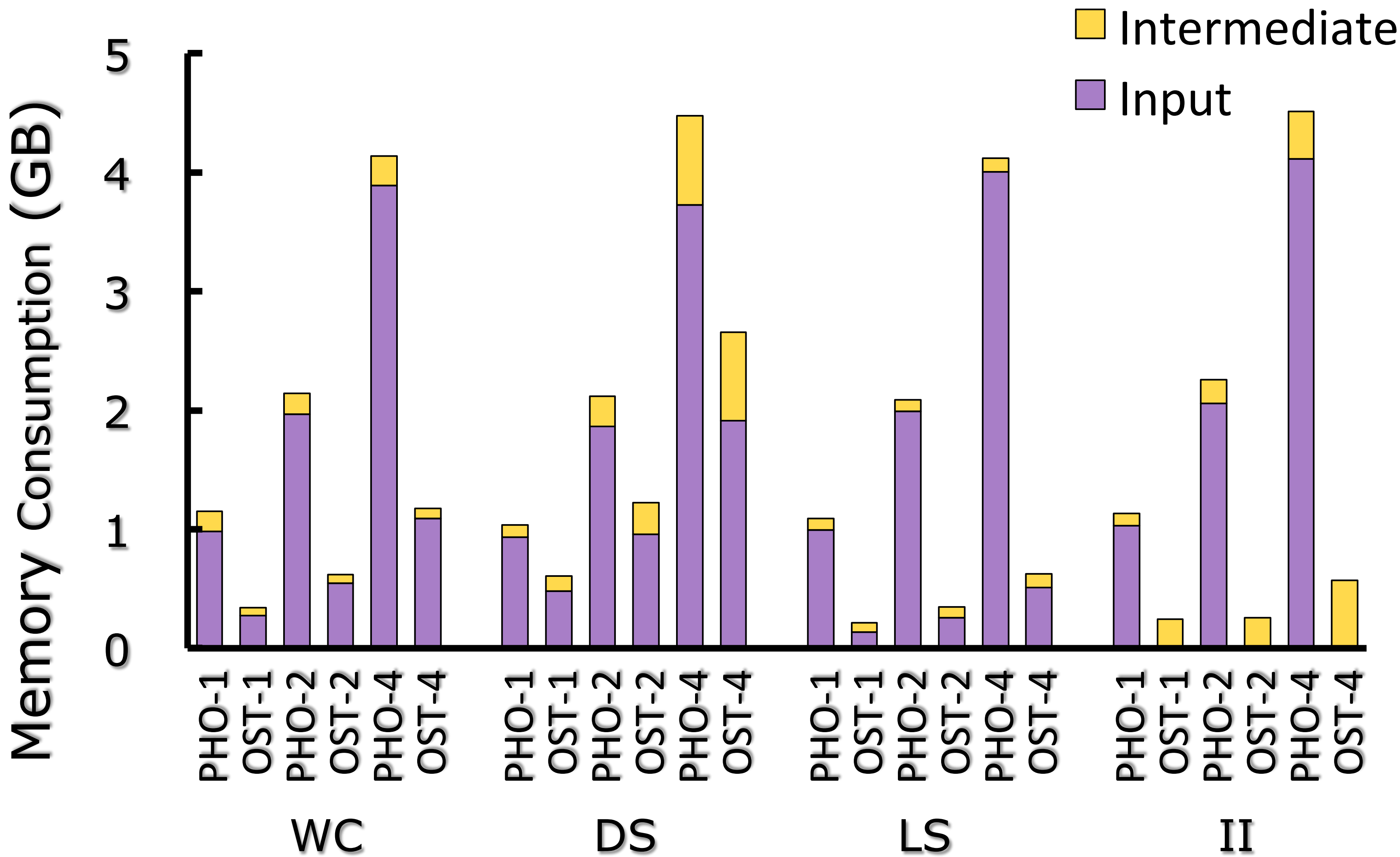
- Support input data memory reuse

Applications	Acquire	Release
WordCount (WC)	11	3
Distributed Sort (DS)	Default	Default
Log Statistics (LS)	Default	Default
Inverted Index (II)	11	3

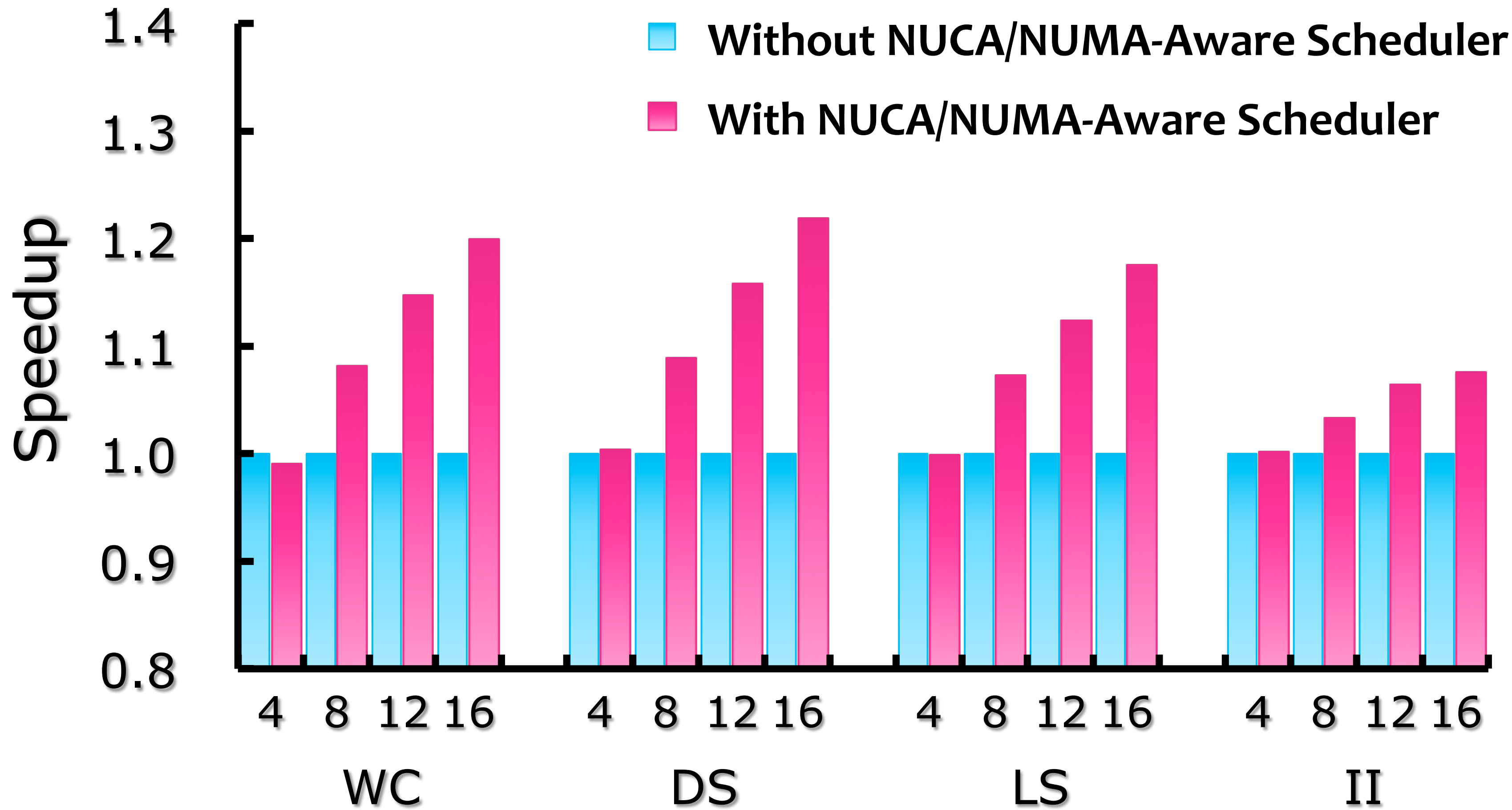
Overall Performance



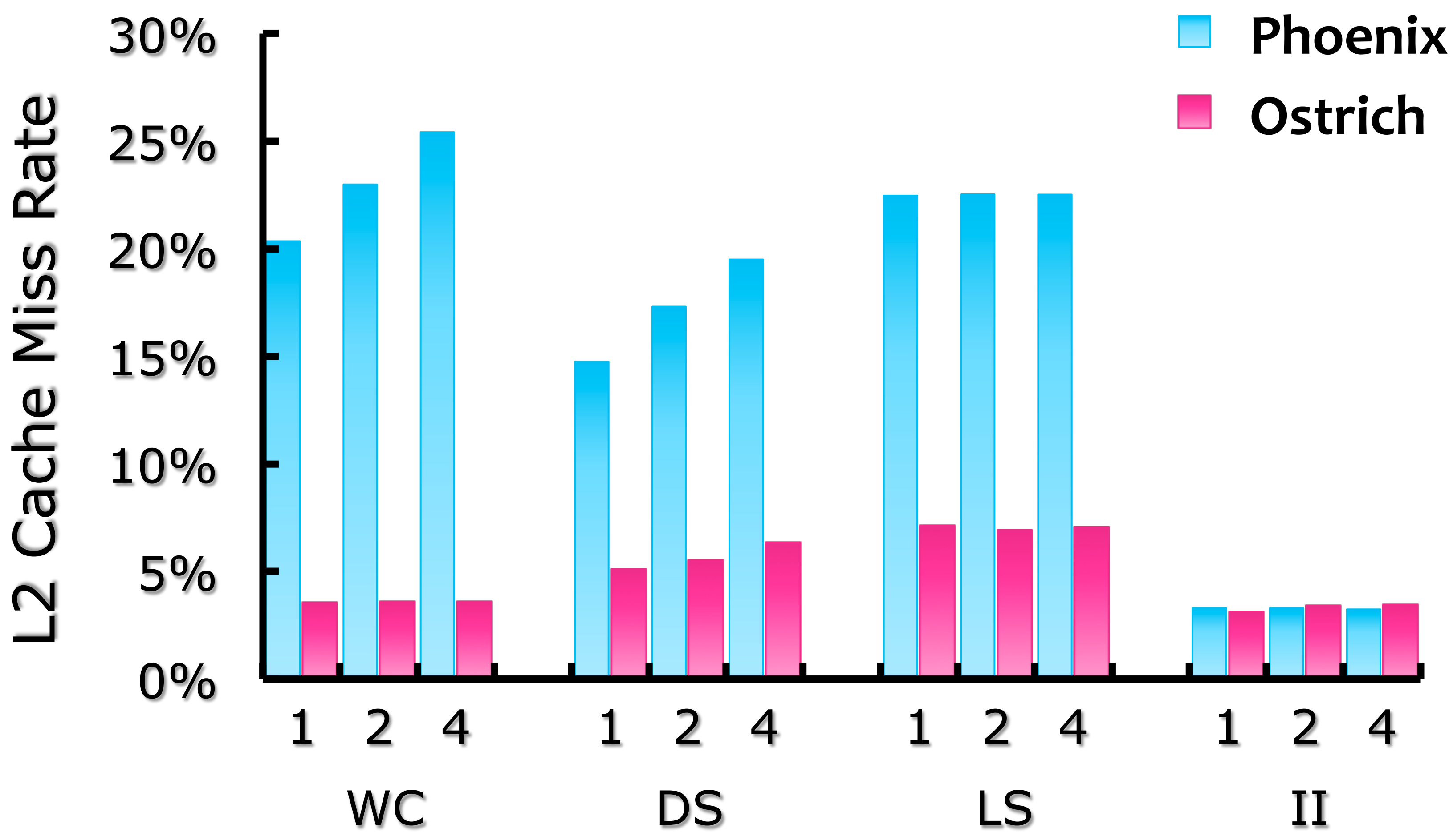
Memory Consumption



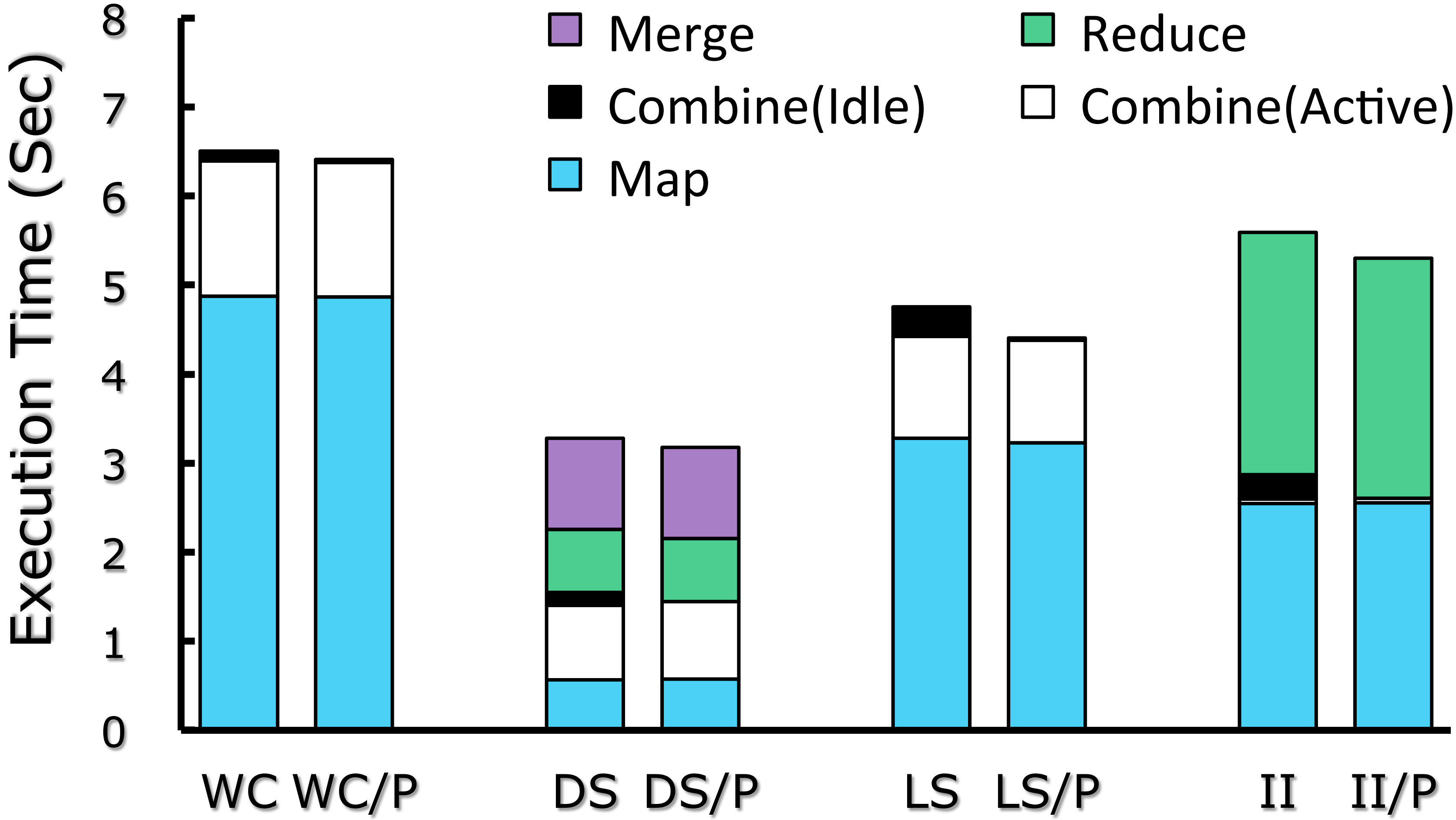
NUCA/NUMA-Aware Scheduler



Exploit Locality

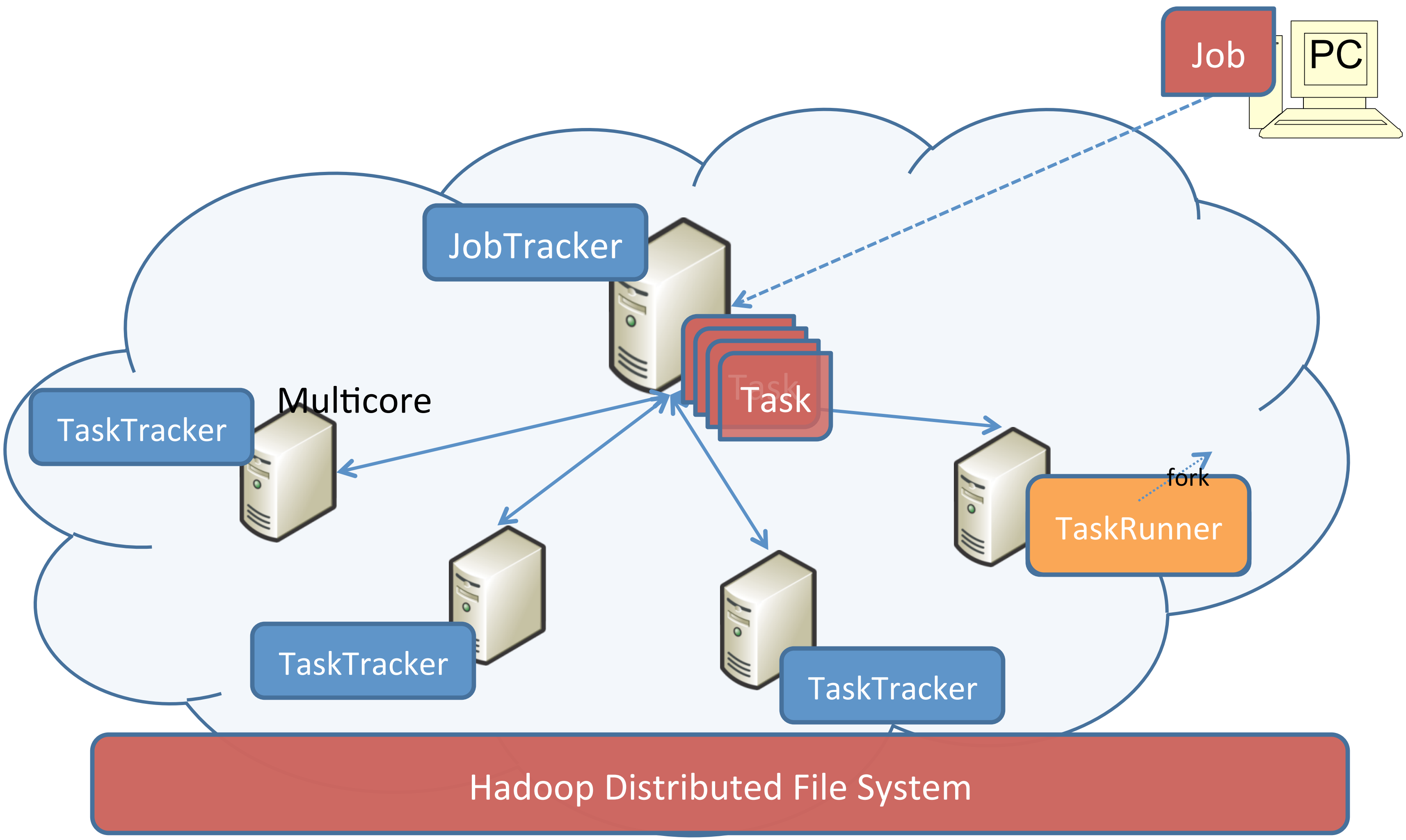


Software Pipeline



Exploiting Locality and Parallelism of MapReduce with Hierarchical MapReduce

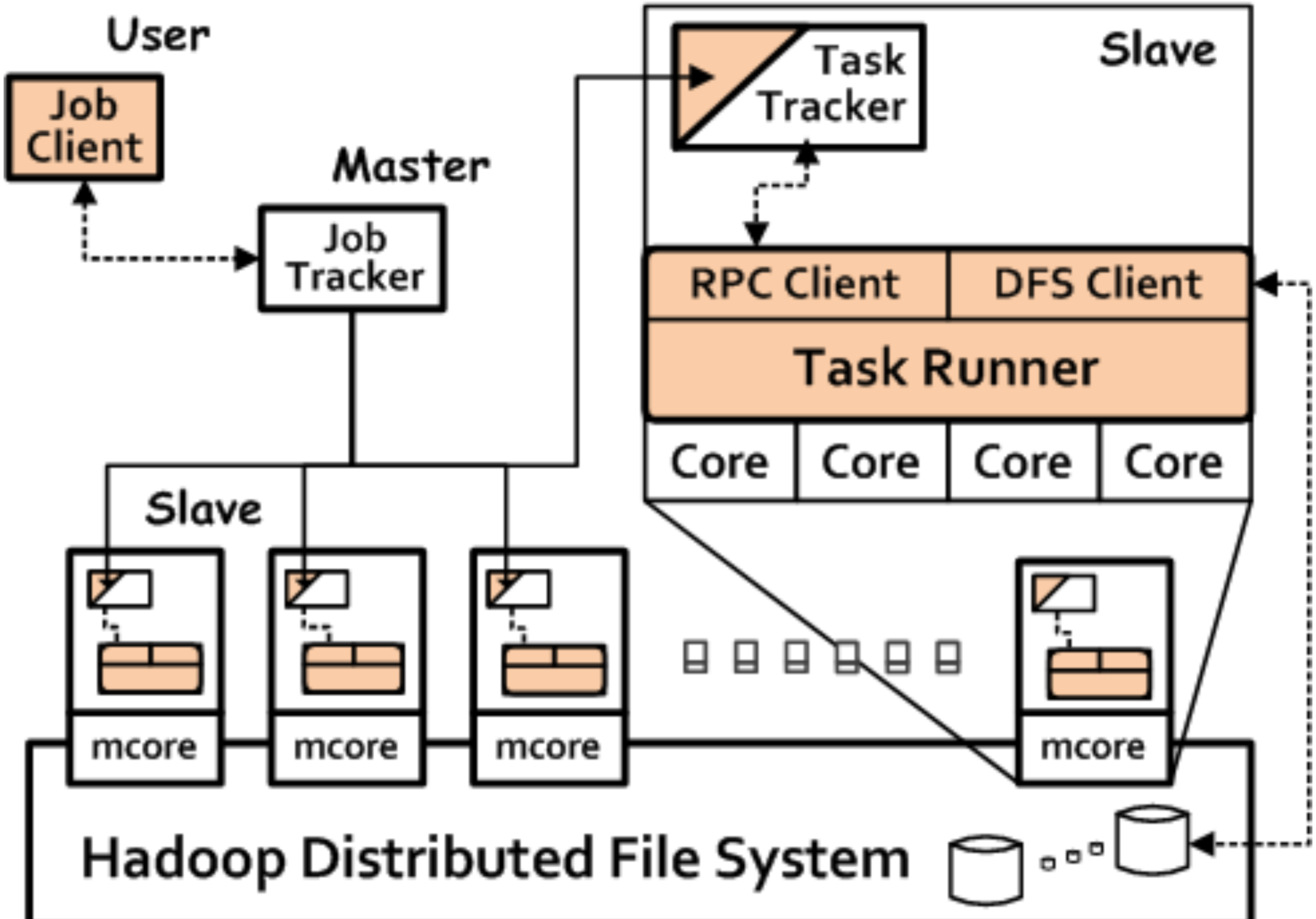
Hadoop: MapReduce on Clusters



Motivation

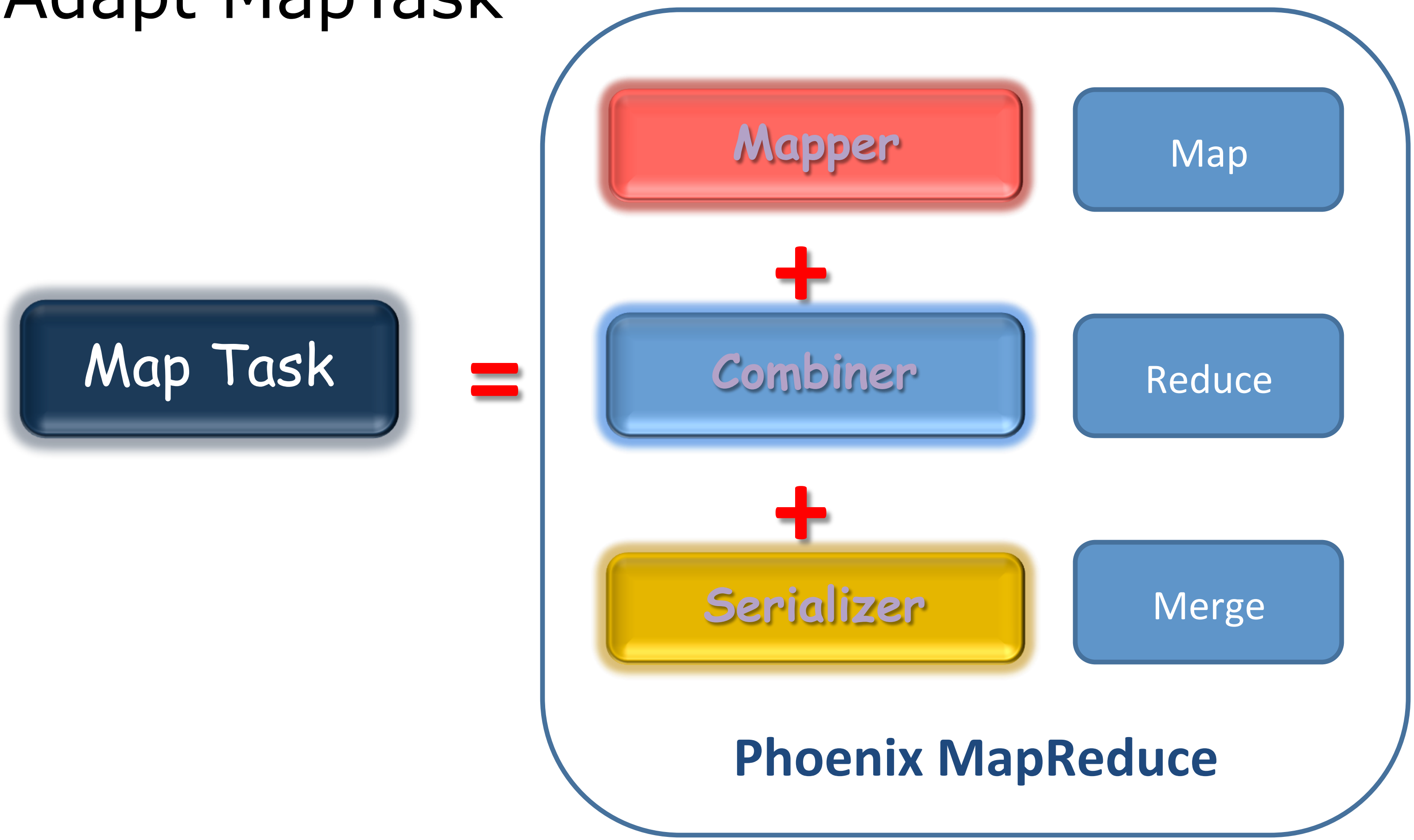
- Two level of parallelism on typical clusters
  – Multi-core based parallel architecture on a single node
  – Cluster-level parallelism among nodes
- Multiple levels of data locality
  – Cache locality
  – Data locality among storage and network
- Both Hadoop and Ostrich are not good at exploiting these parallelism and locality
- Chadoop: Hierarchical MapReduce
  – Based on Hadoop and Ostich
  – Fine-grained control on system resources with C language based runtime

Chadoop Architecture



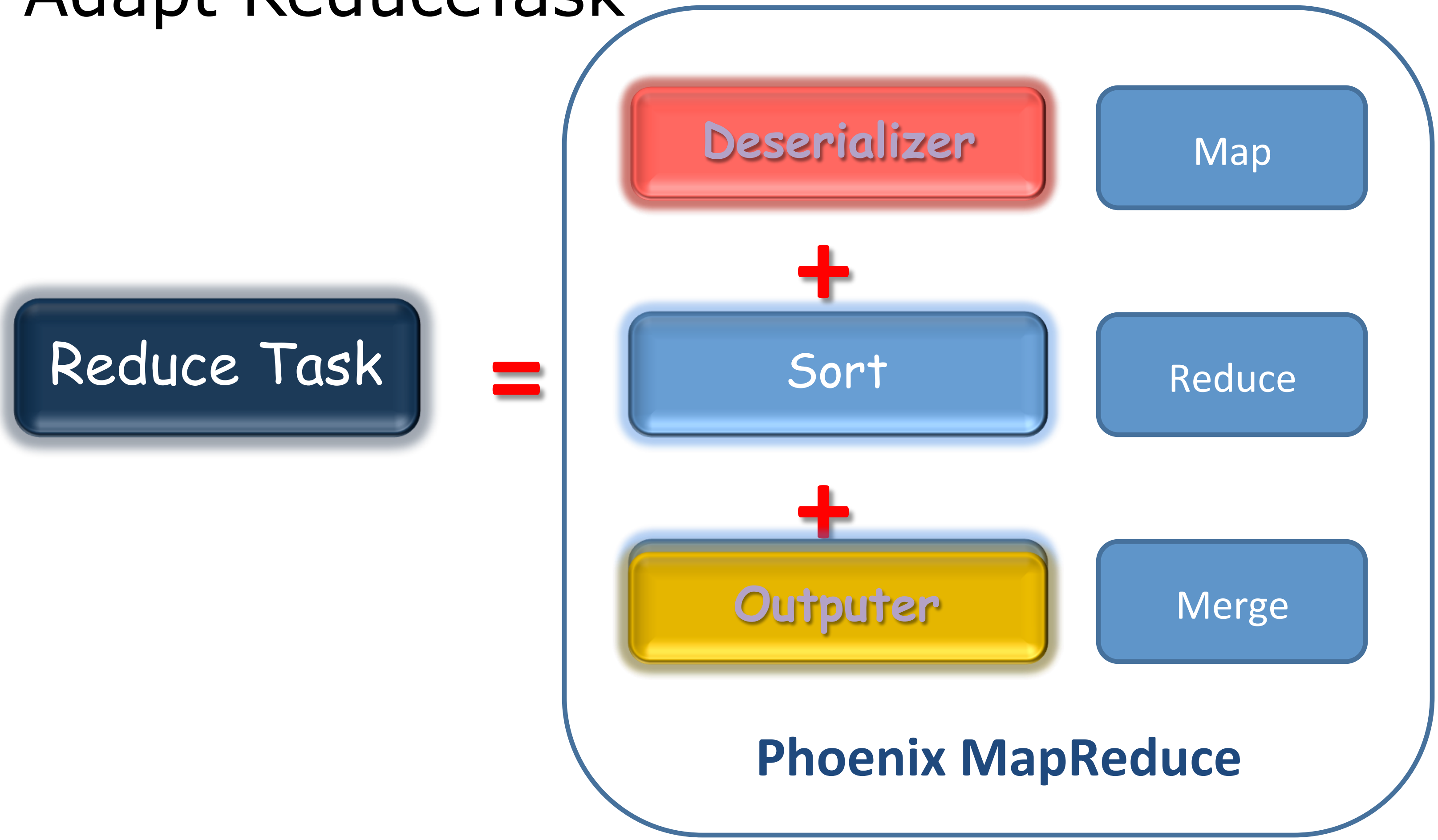
Adapt Ostrich to Hadoop (1)

- Adapt MapTask



Adapt Ostrich to Hadoop (2)

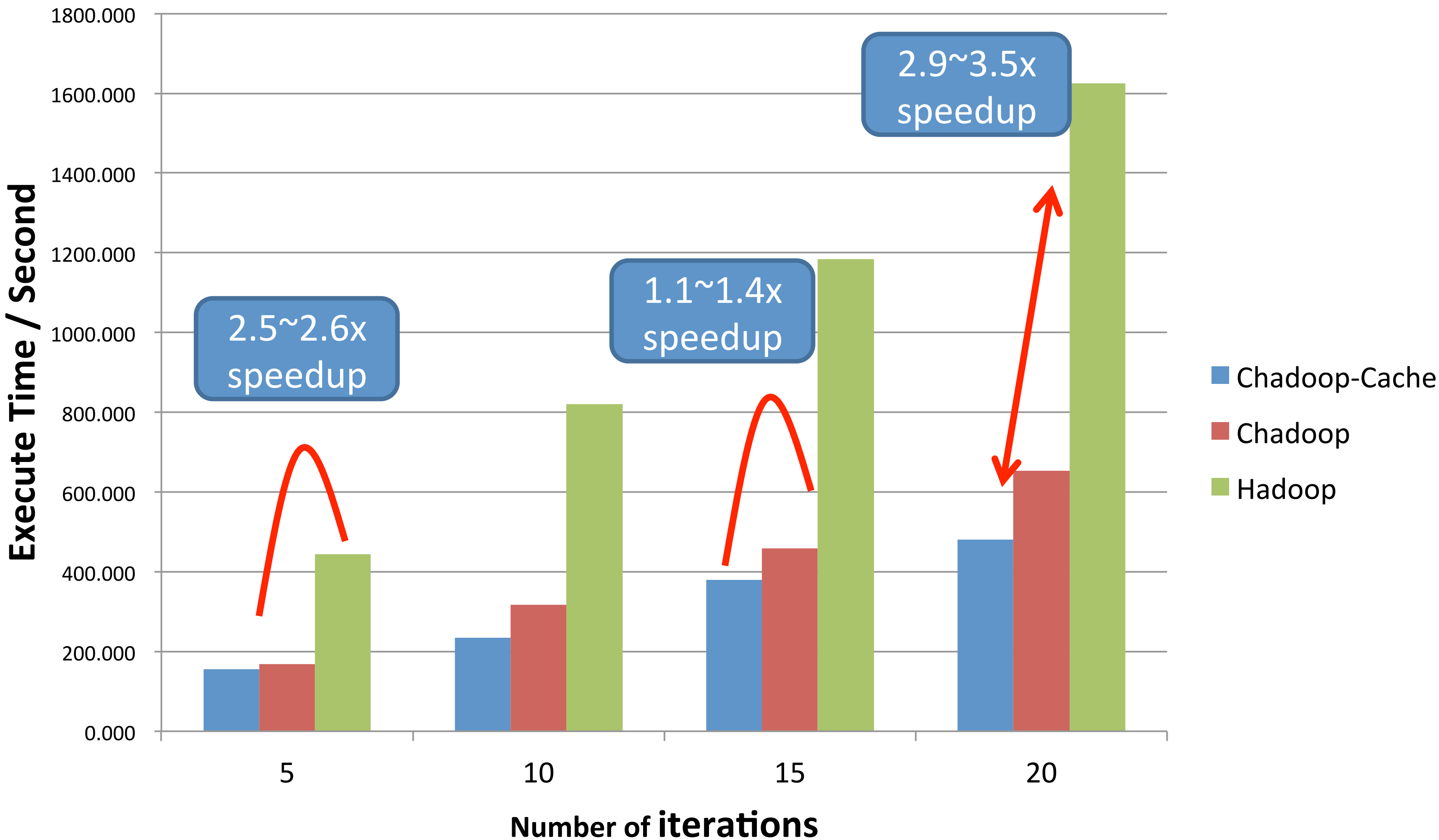
- Adapt ReduceTask



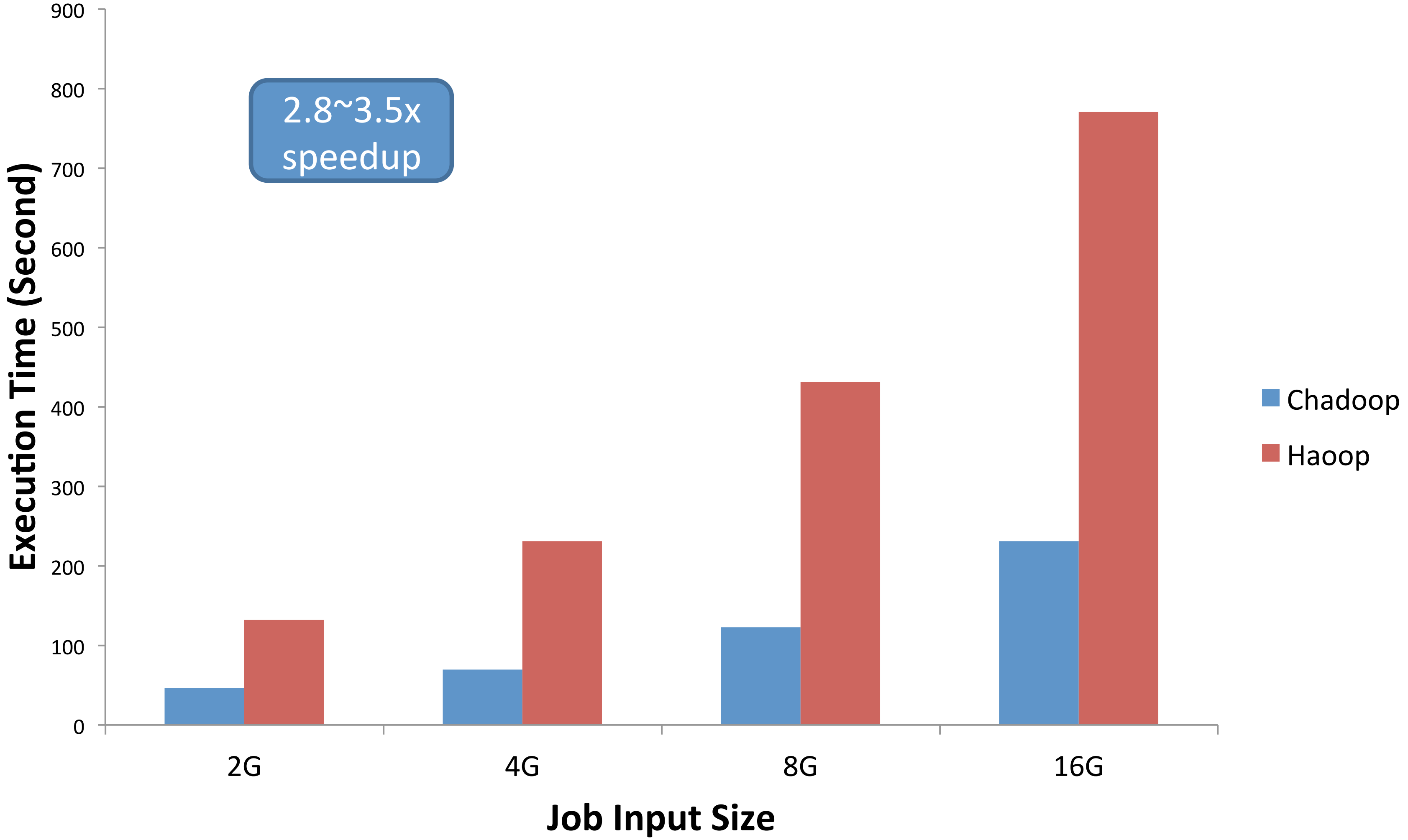
Hadoop support

- Start cache server when initializing TaskTracker
- Assign an id to each TaskRunner
  - Enforce each TaskRunner mapped to a specific control block
- Improved Scheduler Affinity
  - Tasks report the current <split, cache location> pair to NameNode before done
  - NameNode maintains these info
  - JobClient queries split locations from NameNode before submitting jobs
  - Scheduler gives higher priority to assign task with cached data
- No more than 50 LOC hacked in Hadoop

K-Means overall performance



WordCount



900
800
700
600
500
400
300
200
100
0
Execution Time (Second)
2.8~3.5x
speedup
2G
4G
8G
16G
Job Input Size
Chadoop
Haoop

Conclusion

Performance and Scalability are two major concerns for

- *MapReduce* on multicore based single machine and clusters

Ostrich

- Tiled MapReduce for multicore single machine

Chadoop

- Hierarchical parallelism and locality on multicore based MapReduce clusters

Further Information

- Tiled MapReduce: Optimizing Resource Usages of Data-parallel Applications on Multicore with Tiling.
  - The 19th International Conference on Parallel Architectures and Compilation Techniques (**PACT 2010**). pp.523–534. Vienna, Austria, September, 2010.

- A Hierarchical Apporach to Maximizing MapReduce Efficiency
  - The 20th International Conference on Parallel Architectures and Compilation Techniques (**PACT 2011, poster**). October, 2011

Thanks

Ostrich

The top land speed
and the largest of bird

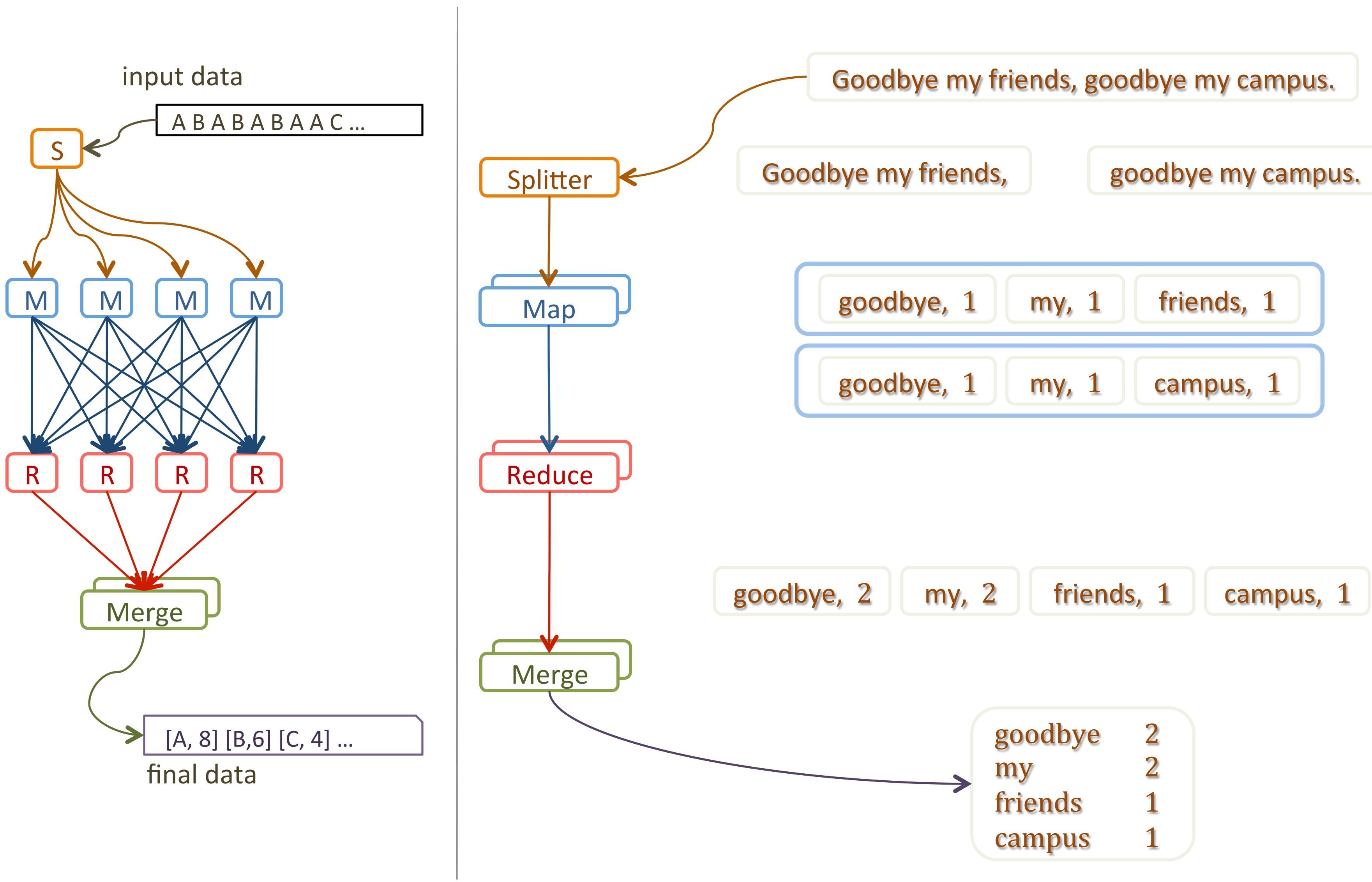
Parallel Processing Institute

http://ppi.fudan.edu.cn

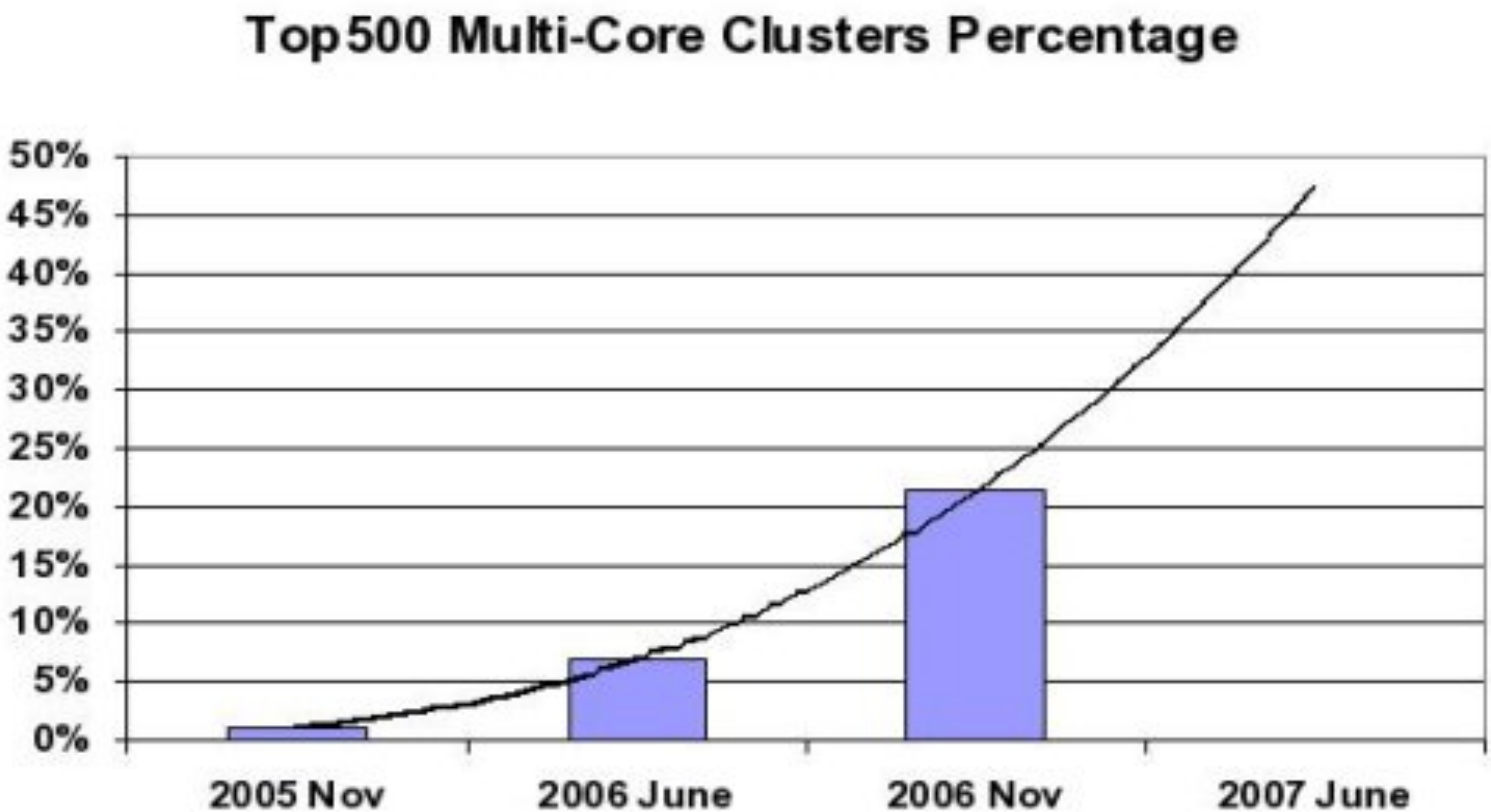
Questions ?

Backup Slides

MapReduce: WordCount example

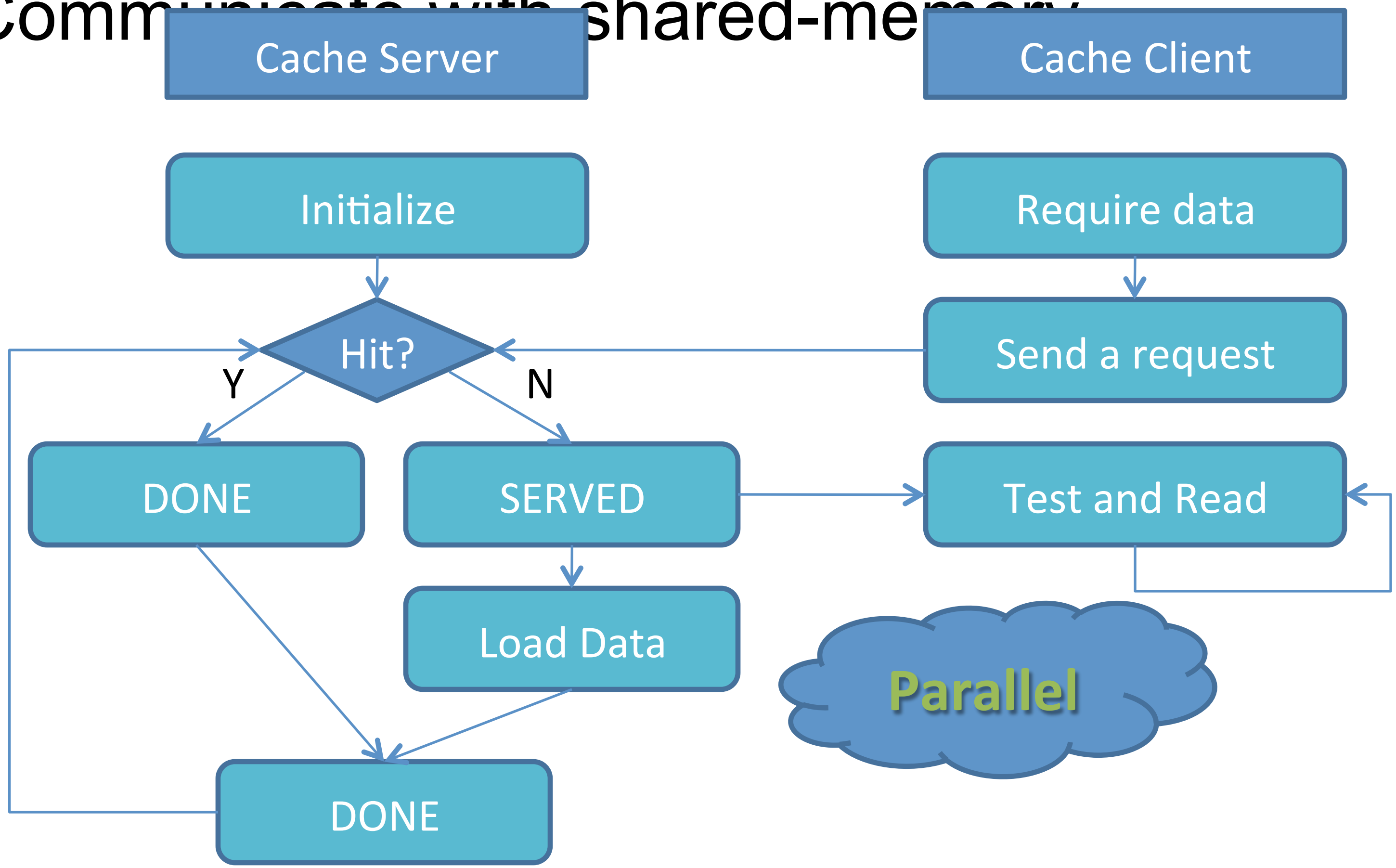


Prevalence of multi-core based clusters



Cache System Workflow

- Communicate with shared-memory



Cache miss penalty on K-Means

- Cache miss jobs penalty over cache hit jobs

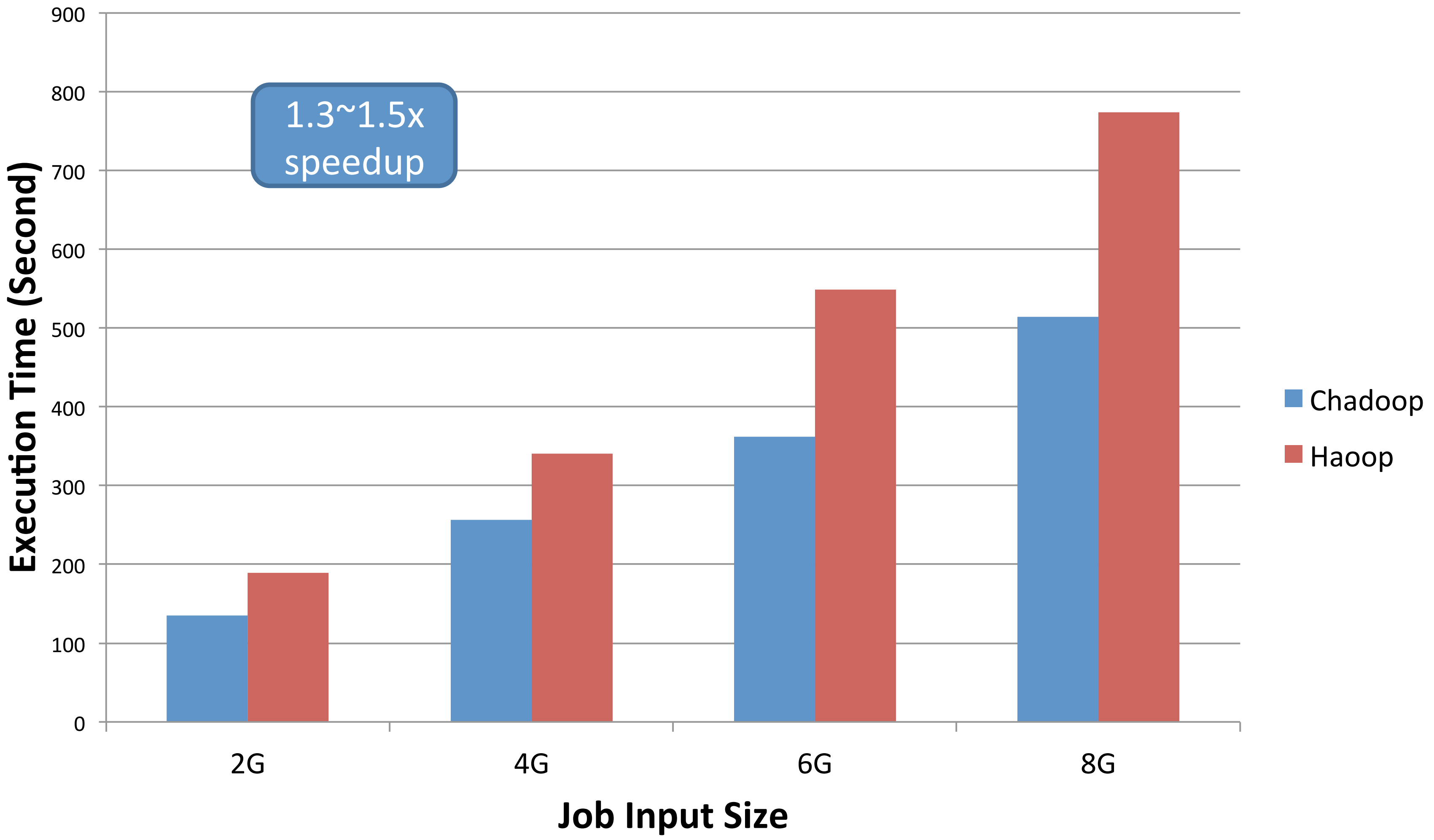
	Total time		Map time			
	hit	miss	hit	miss		
(8 node)				Average	Local (disk)	Remote (network)
400M 50M/map	100.00%	135.77%	100.00%	130.09%	130.52%	127.12%
800M 100M/map	100.00%	218.96%	100.00%	179.12%	142.75%	288.25%
1600M 100M/map	100.00%	218.36%	100.00%	234.51%	229.19%	314.23%

Cache miss rate on K-Means

- Remote cache miss rate
  - 6%~25%
  - Tuned HDFS and MapReduce configuration decrease remote data fetching
- Other data read from local disk

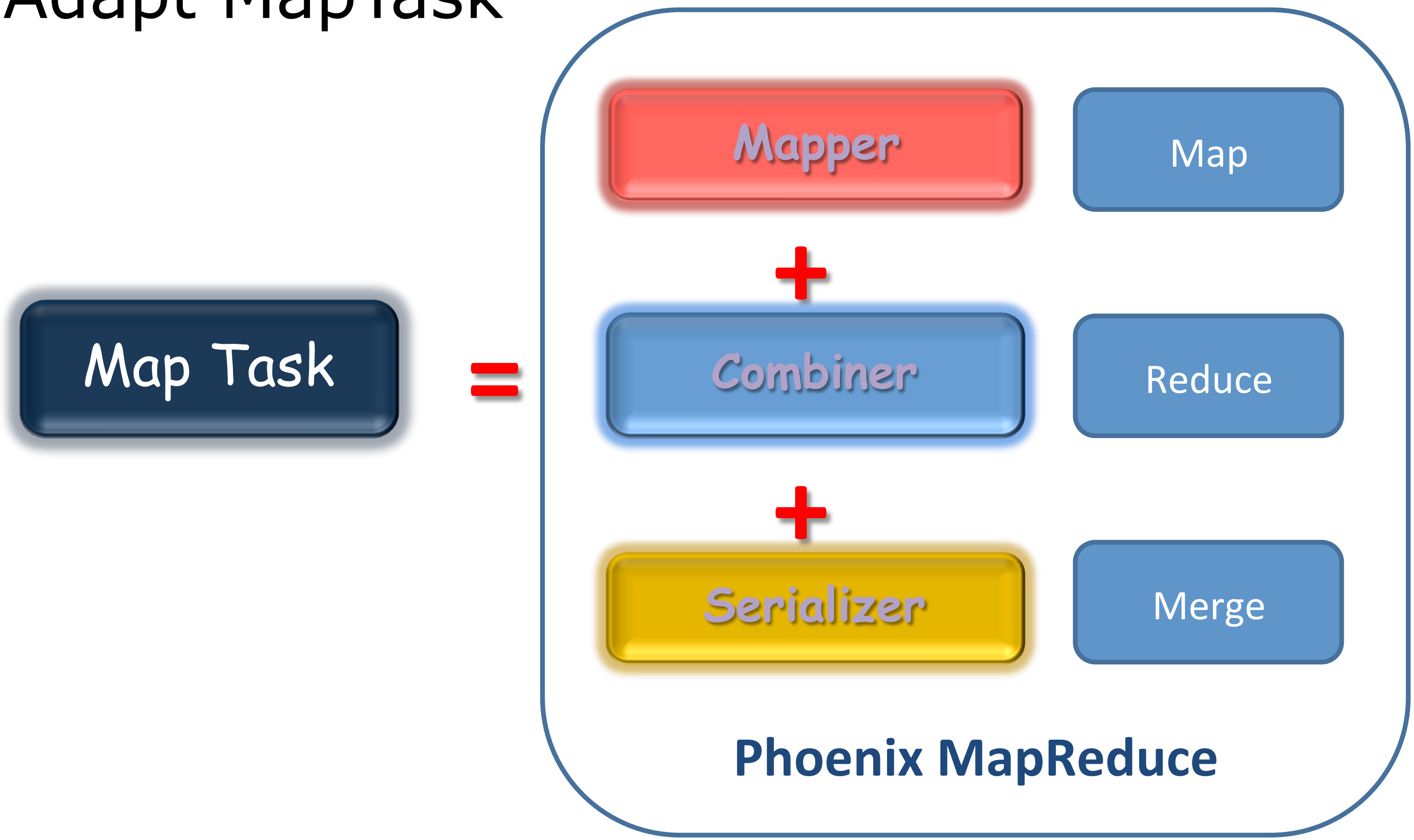
- Our cache system needs only an iteration to warm up
  - Succeeding iterations would process the in-memory data

GigaSort


1.3~1.5x speedup
Execution Time (Second)
900
800
700
600
500
400
300
200
100
0
2G
4G
6G
8G
Job Input Size
Chadoop
Haoop

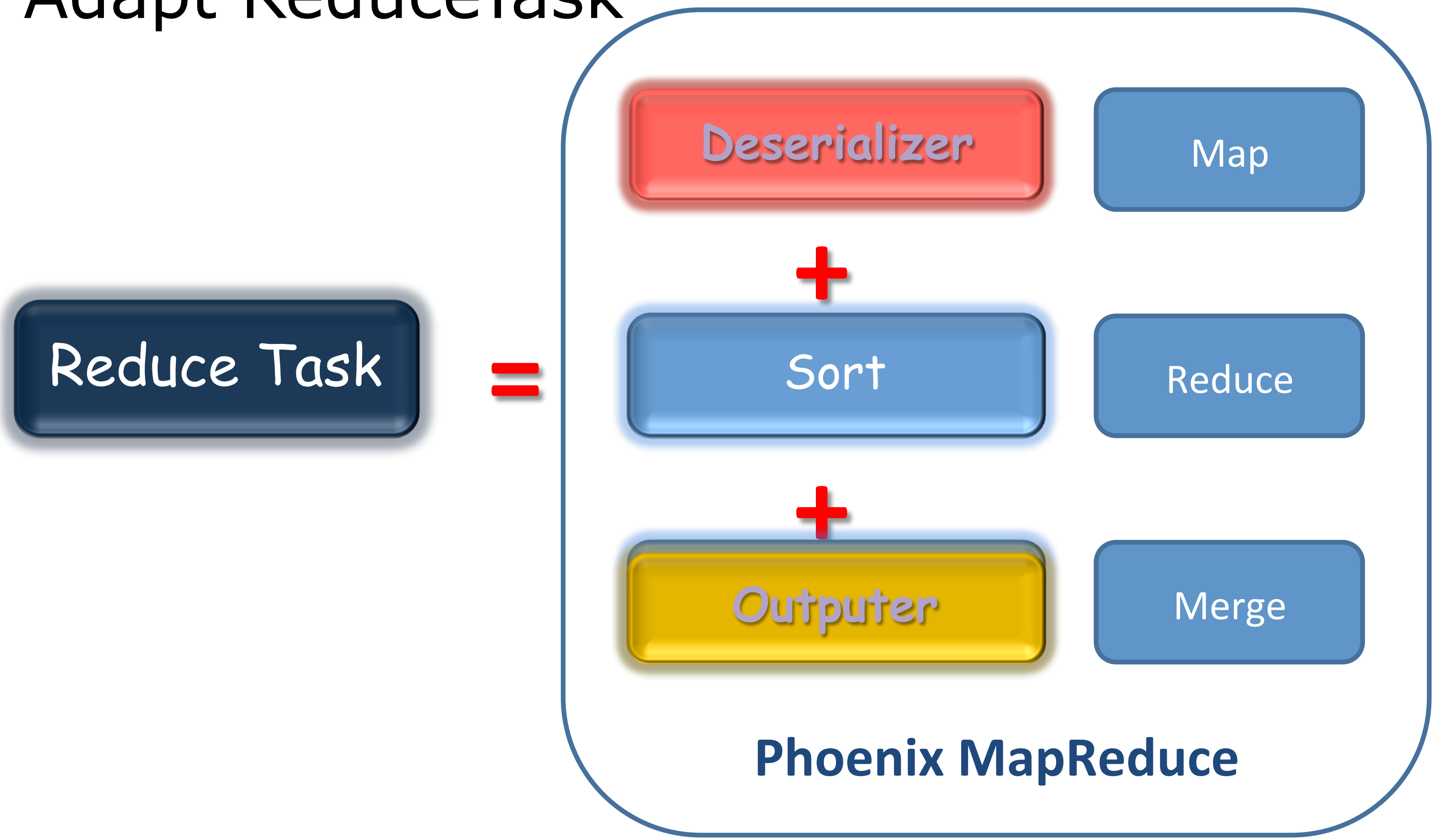
Adapt Ostrich to Hadoop (1)

- Adapt MapTask



Adapt Ostrich to Hadoop (2)

- Adapt ReduceTask



Fine-grained optimizations (1)

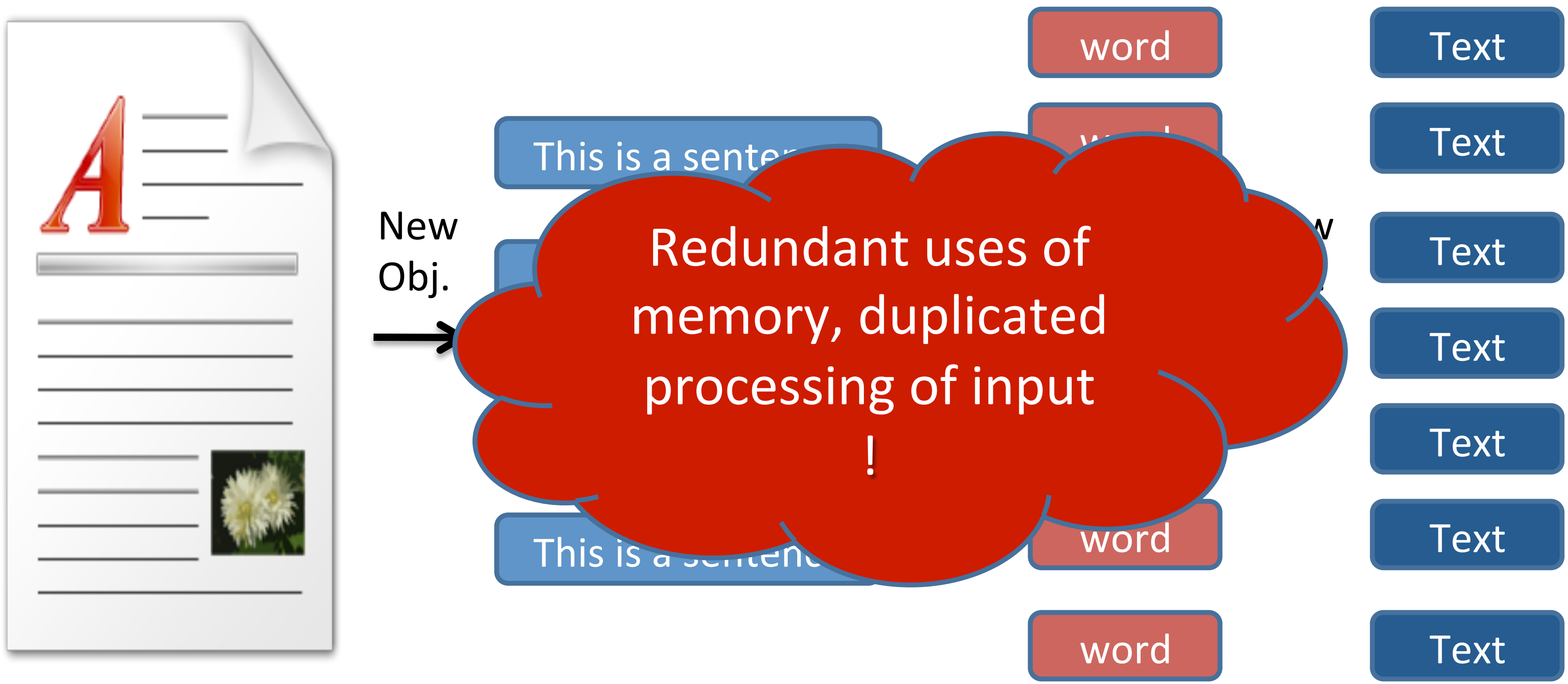
- Exploiting Parallelism
  - Overlap data loading and MapReduce processing time
    - Map input: byte-granularity
    - Reduce input: file-granularity

Fine-grained optimizations (2)

- Increase the granularity of the serialization and deserialization
  - Require users to provide their (de)serialization function
    - Hadoop requires users to implement the writable interface
  - Reduce application function-call overhead
- Configurable number of worker threads
  - E.g., less threads for data-intensive tasks
- Configurable inner-process unit size
  - Fit into L1 cache
  - Can use online-profiling to tune the unit size

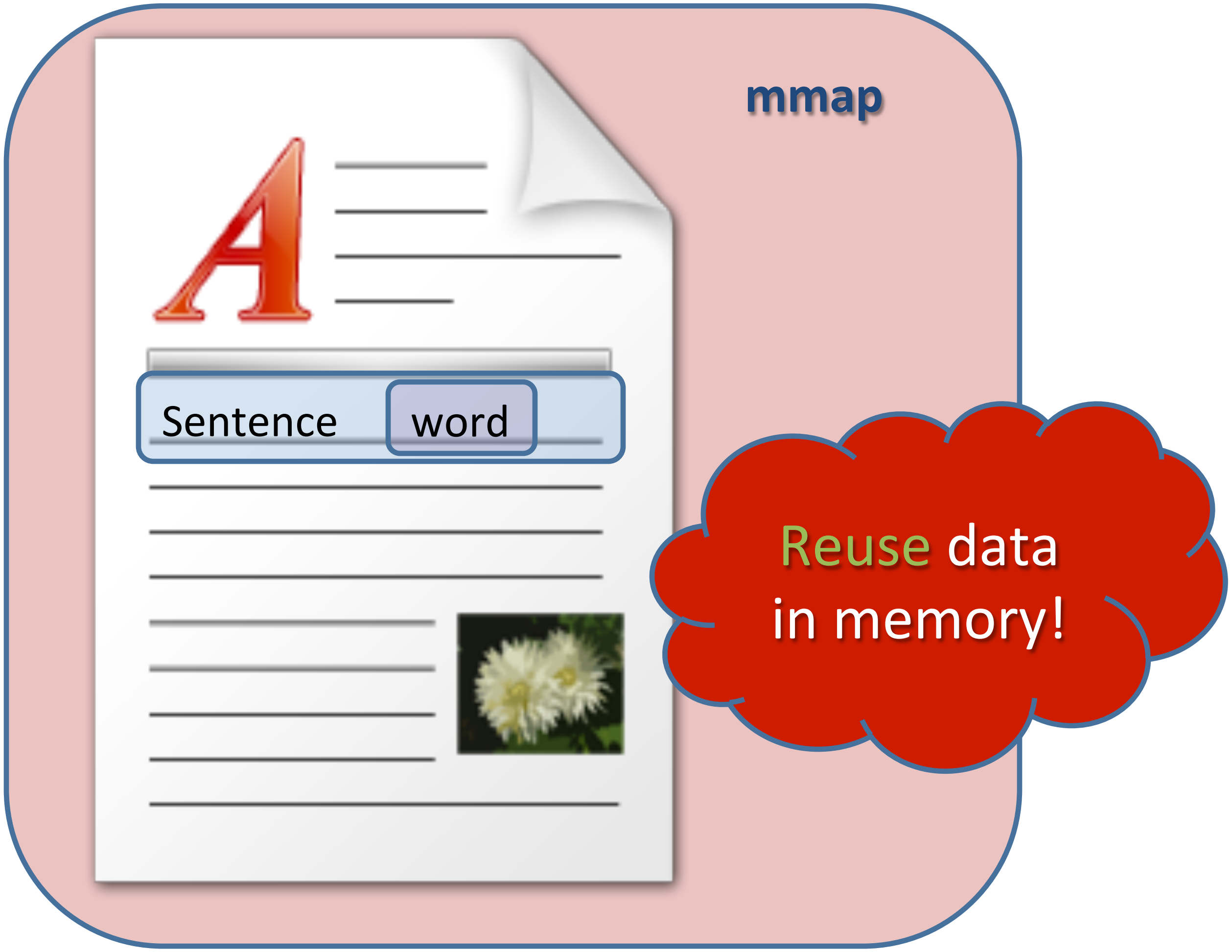
Exploit data locality - memory

- Hadoop (Java) style



Exploit data locality - memory

- Chadoop (C) style



Exploit data locality - storage

- Inner algorithm commo ta
  - Iterativ ML,
    like K-
- Cross
  - Natural
  - Incremental computi yadInc)
  - Joined tables (user info table etc.)

Cache allows sharing of data among jobs

Cache system design