Tiled-MapReduce

Optimizing Resource Usages of Data-parallel Applications on Multicore

Rong Chen Haibo Chen Binyu Zang

Parallel Processing Institute

Fudan University

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 Labs

Data-parallel Programming Model

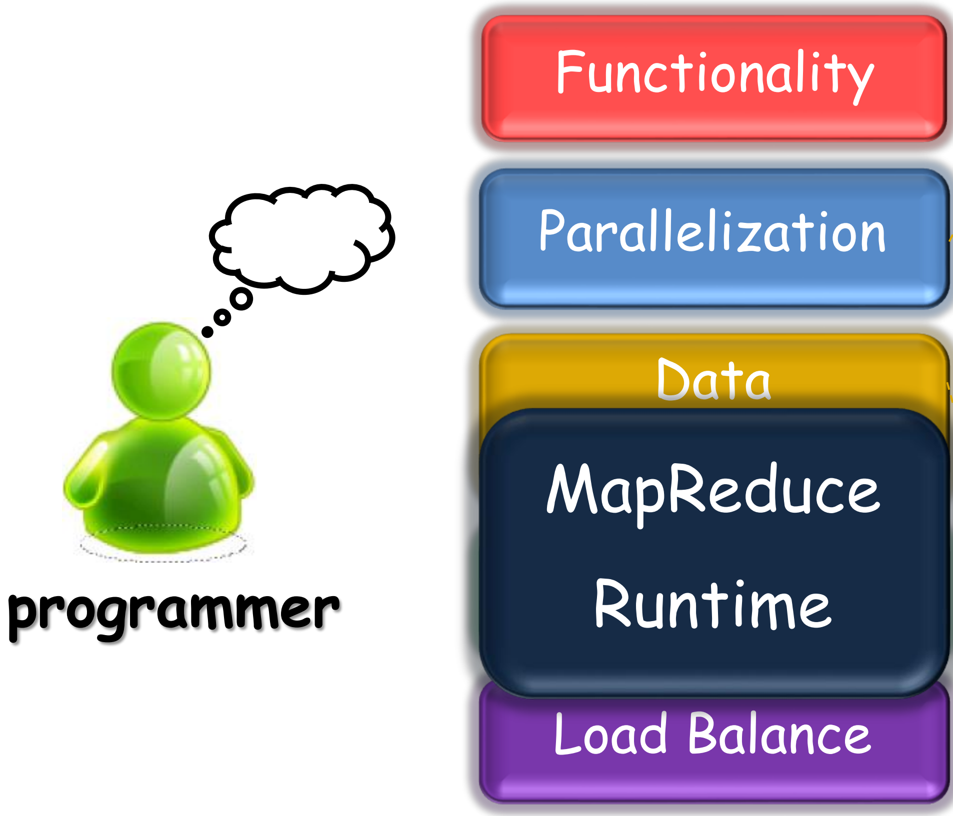
MapReduce: a simple programming model for data-parallel applications from Google

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



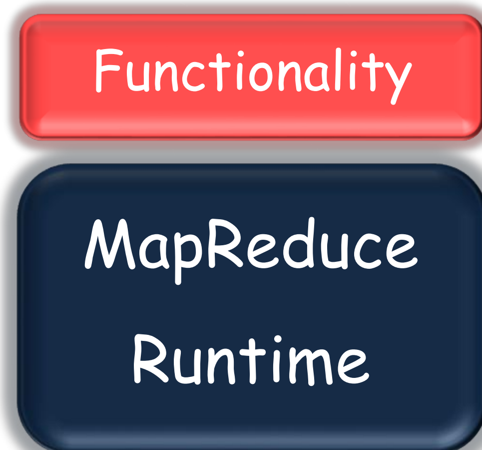
Two Primitive:

Map (*input*)

Reduce (*key, values*)

Data-parallel Programming Model

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



Two Primitive:

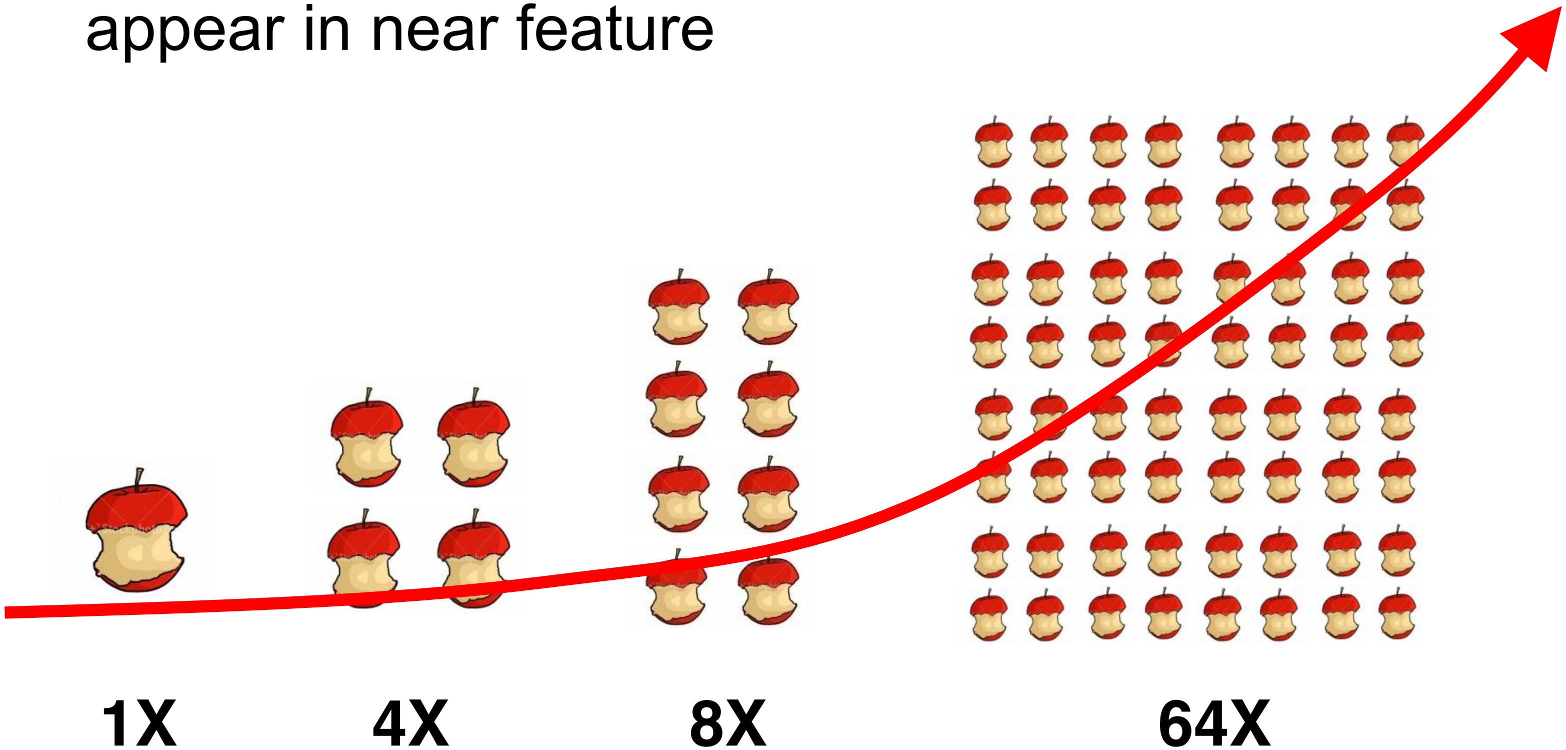
Map (***input***)
  for each ***word*** in ***input***
    emit (***word***, ***1***)

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

Multicore

Multicore is commercially prevalent recently

- Quad-cores and eight cores on a chip are common,
- Tens and hundreds of cores on a single chip will appear in near feature



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

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

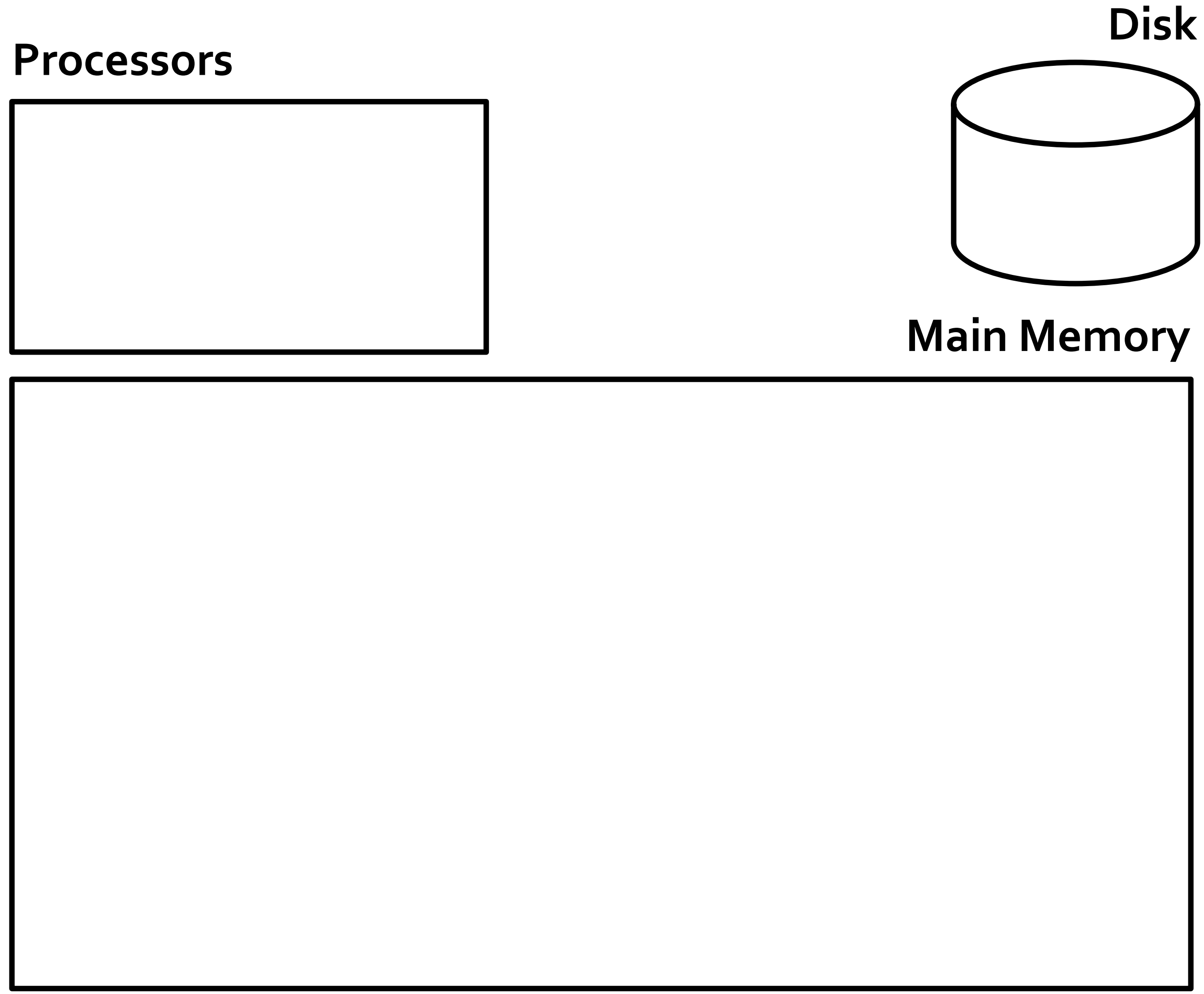
> Scalable data structure

e.g. hash table

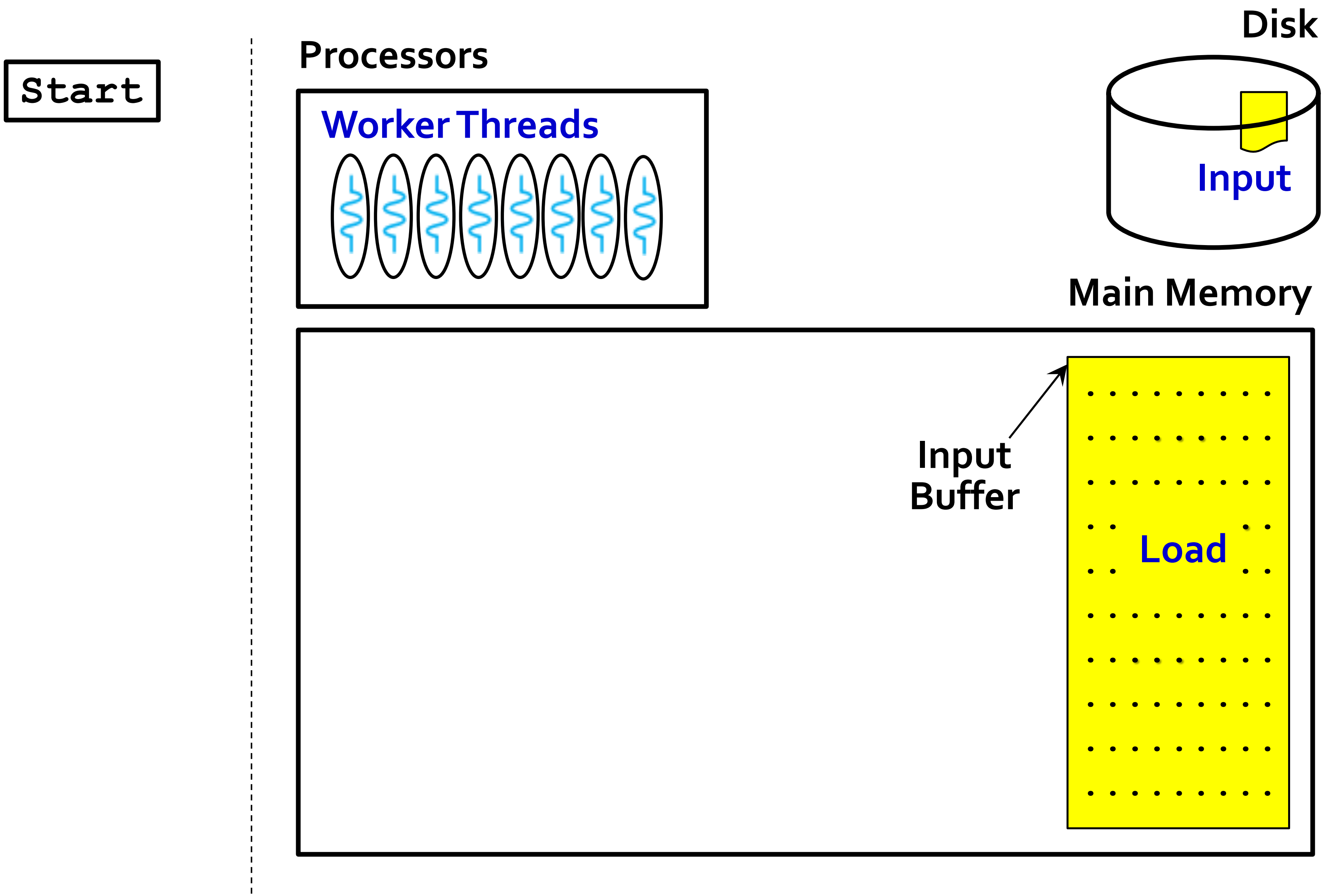
> OS Interaction

e.g. memory allocator, thread pool

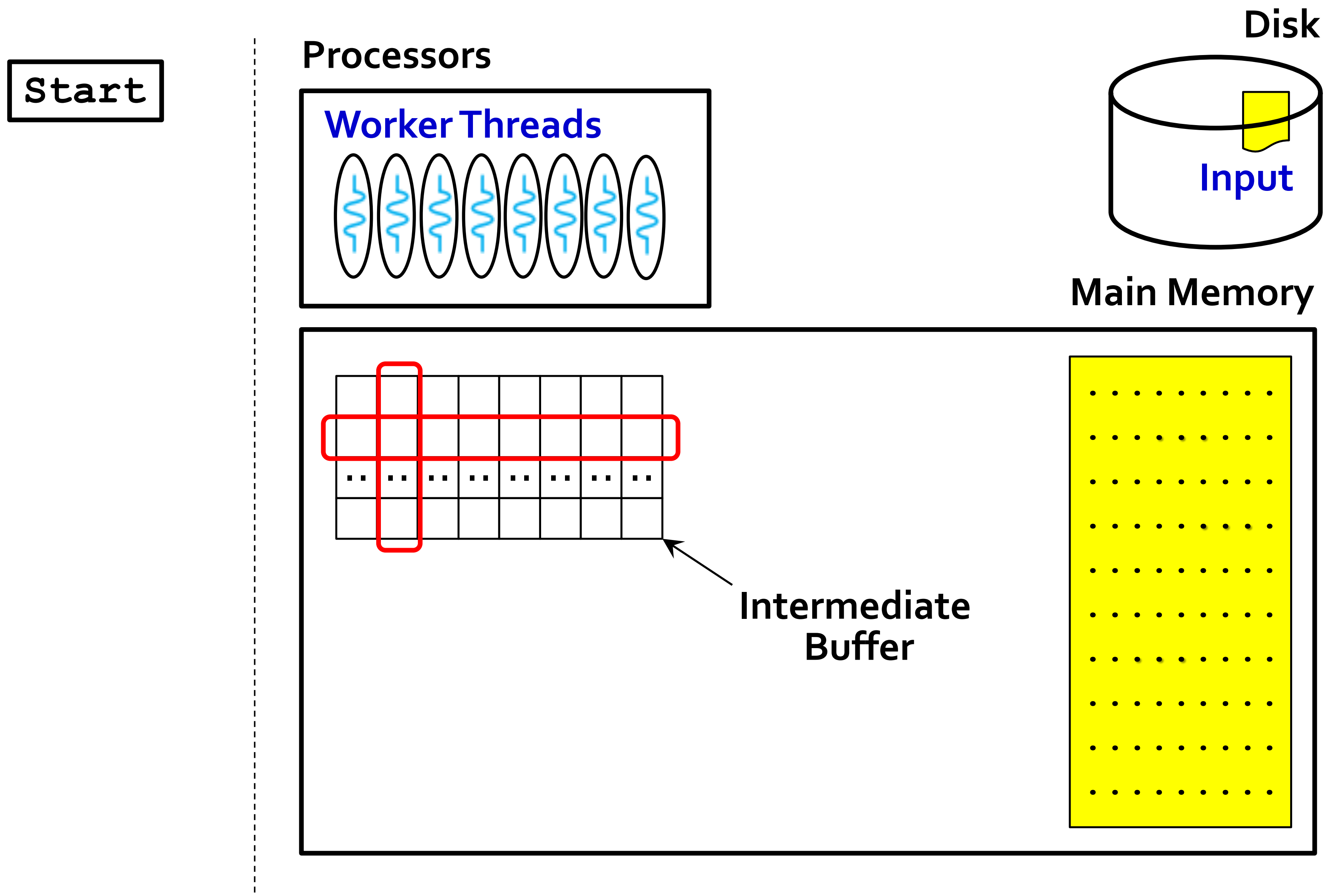
Implementation on Multicore



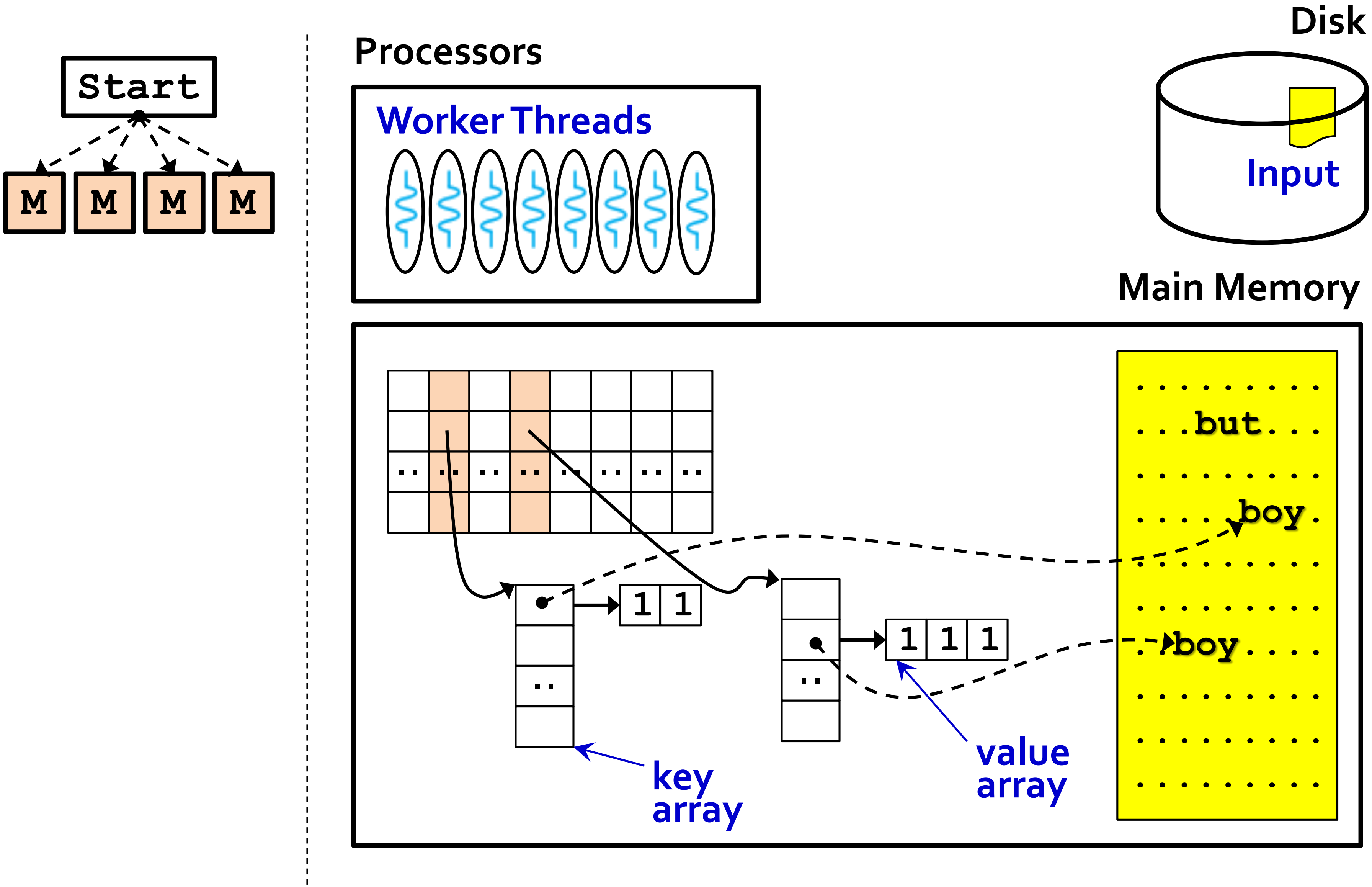
Implementation on Multicore



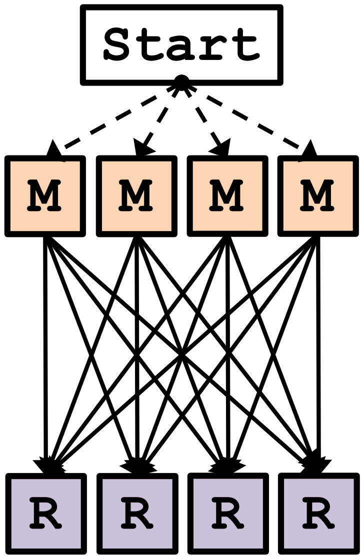
Implementation on Multicore

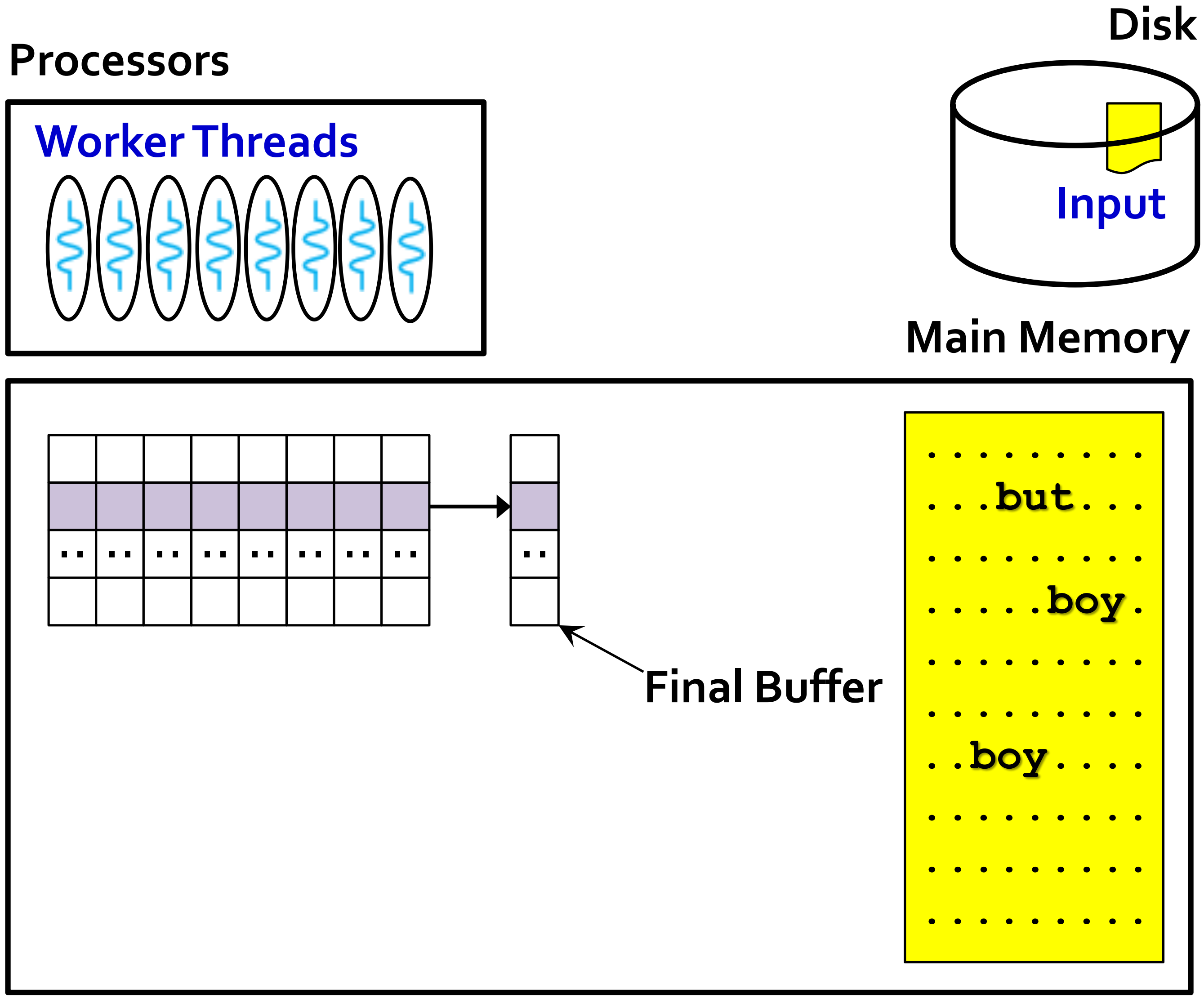


Implementation on Multicore

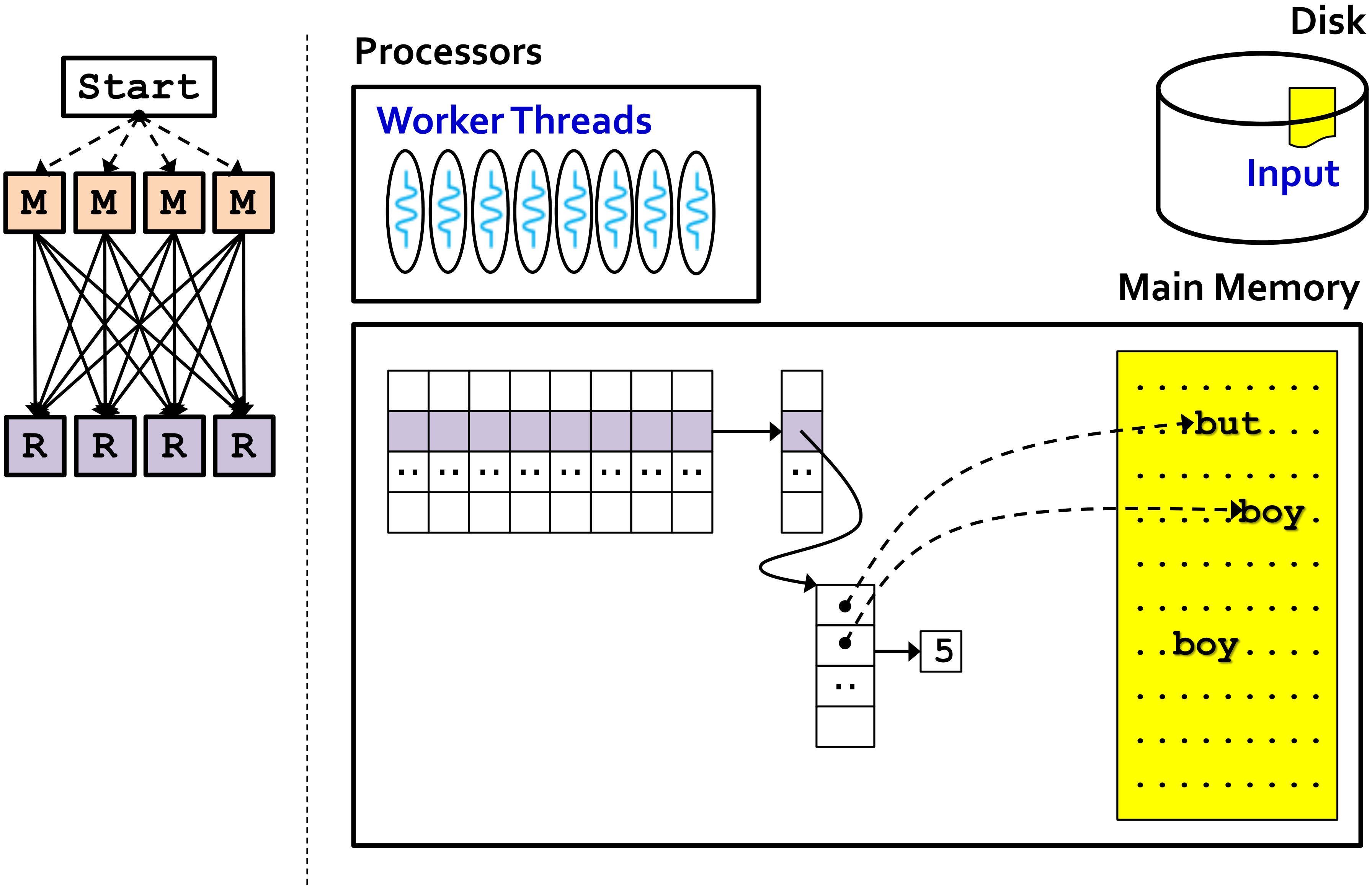


Implementation on Multicore

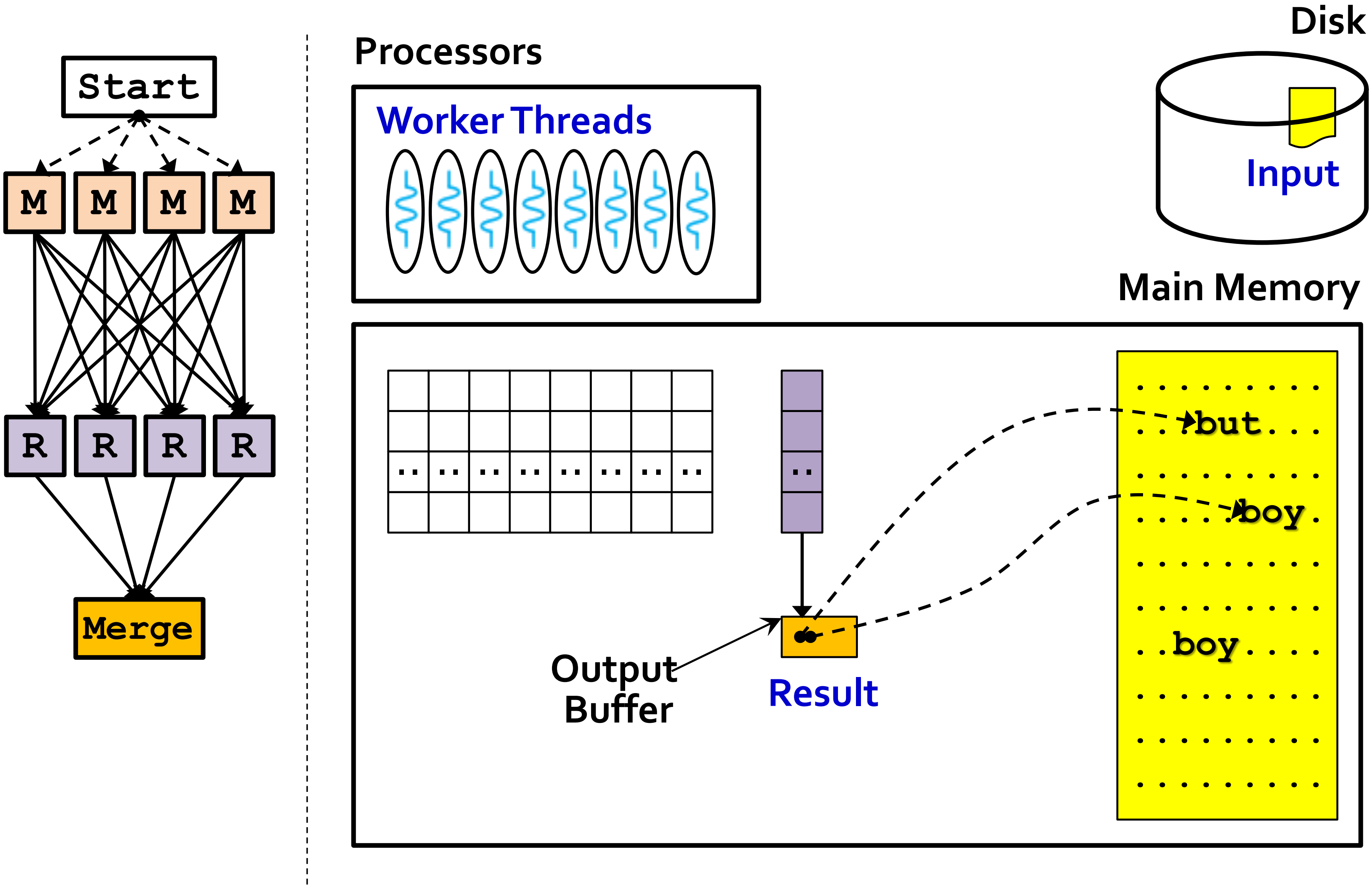




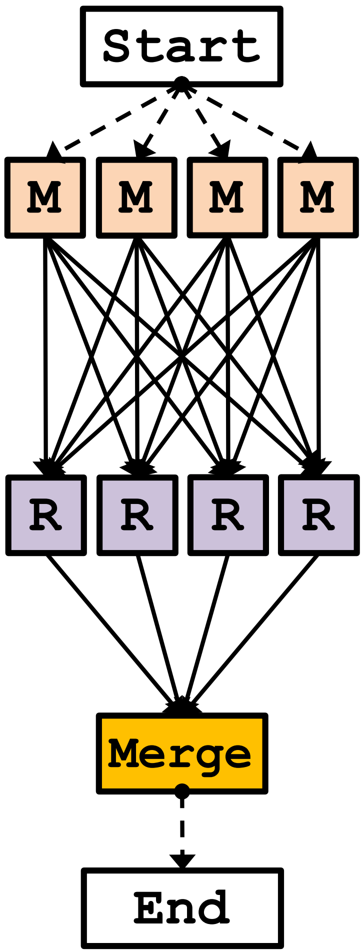
Implementation on Multicore

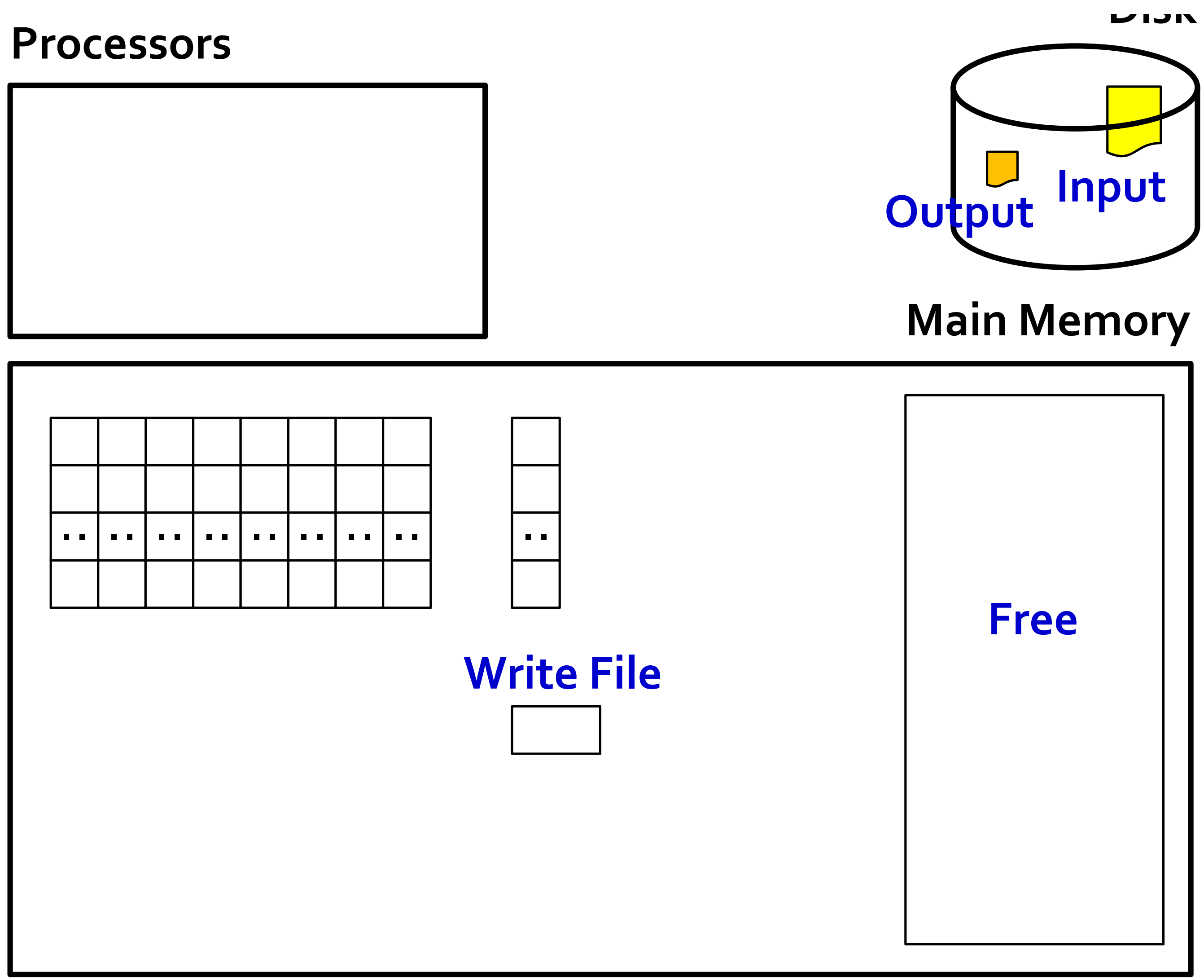


Implementation on Multicore



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

Strict dependency barriers

- 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"

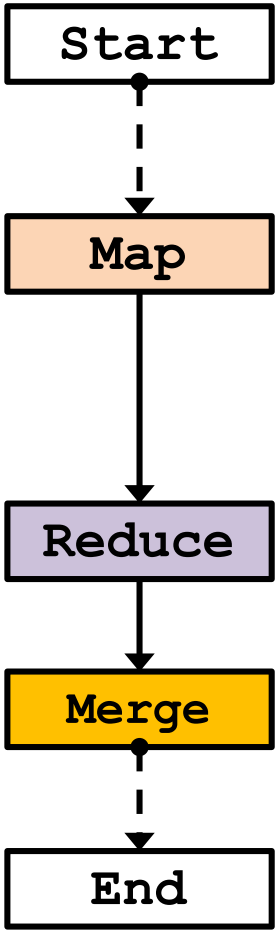
- Divide a large MapReduce job into a number of independent small sub-jobs
- Iteratively process one sub-job at a time

Requirement

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

Tiled-MapReduce

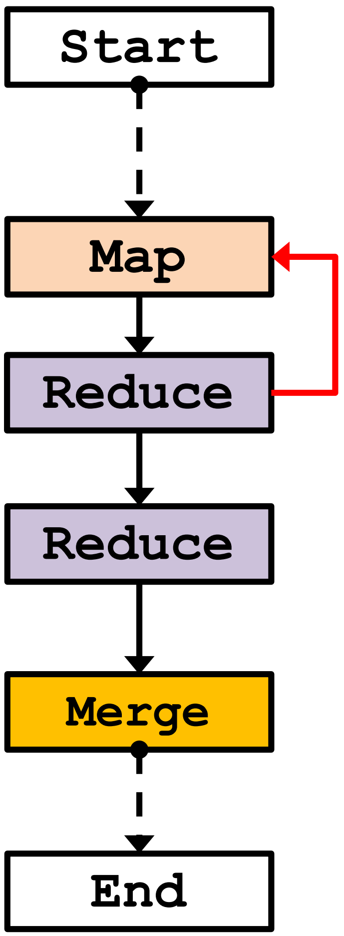
Extensions to MapReduce Model



Tiled-MapReduce

Extensions to MapReduce Model

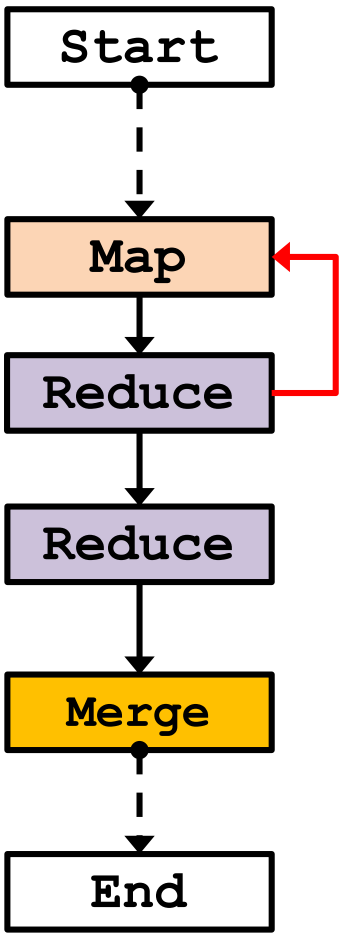
1. Replace the **Map** phase with a loop of **Map** and **Reduce** phases



Tiled-MapReduce

Extensions to MapReduce Model

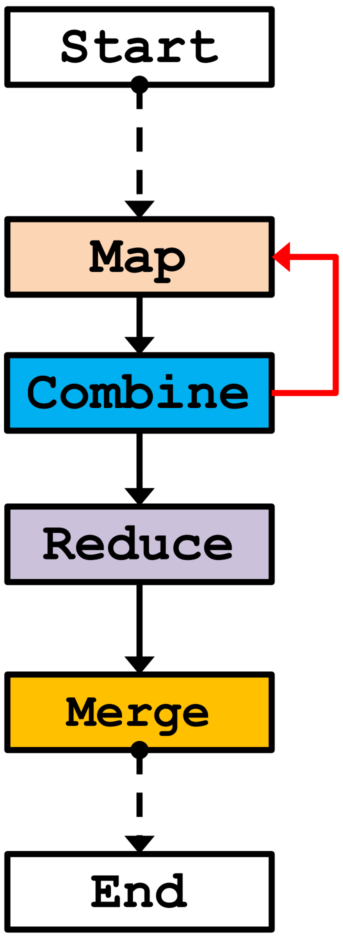
1. Replace the **Map** phase with a loop of **Map** and **Reduce** phases
2. Process one sub-job in each iteration



Tiled-MapReduce

Extensions to MapReduce Model

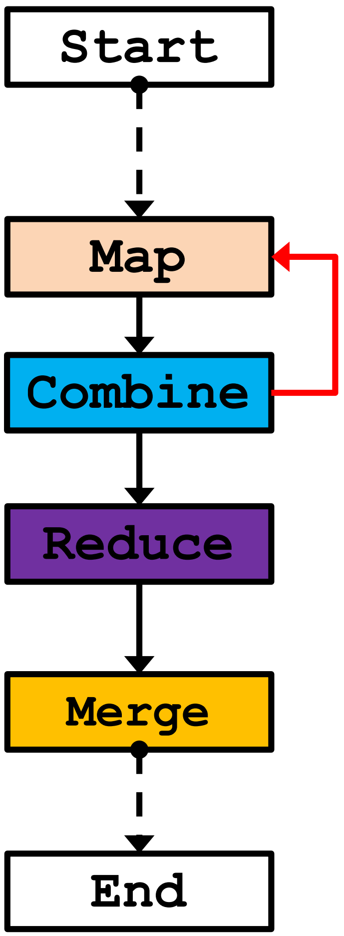
1. Replace the Map phase with a loop of Map and Reduce phases
2. Process one sub-job in each iteration
3. Rename the Reduce phase within loop to the Combine phase



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

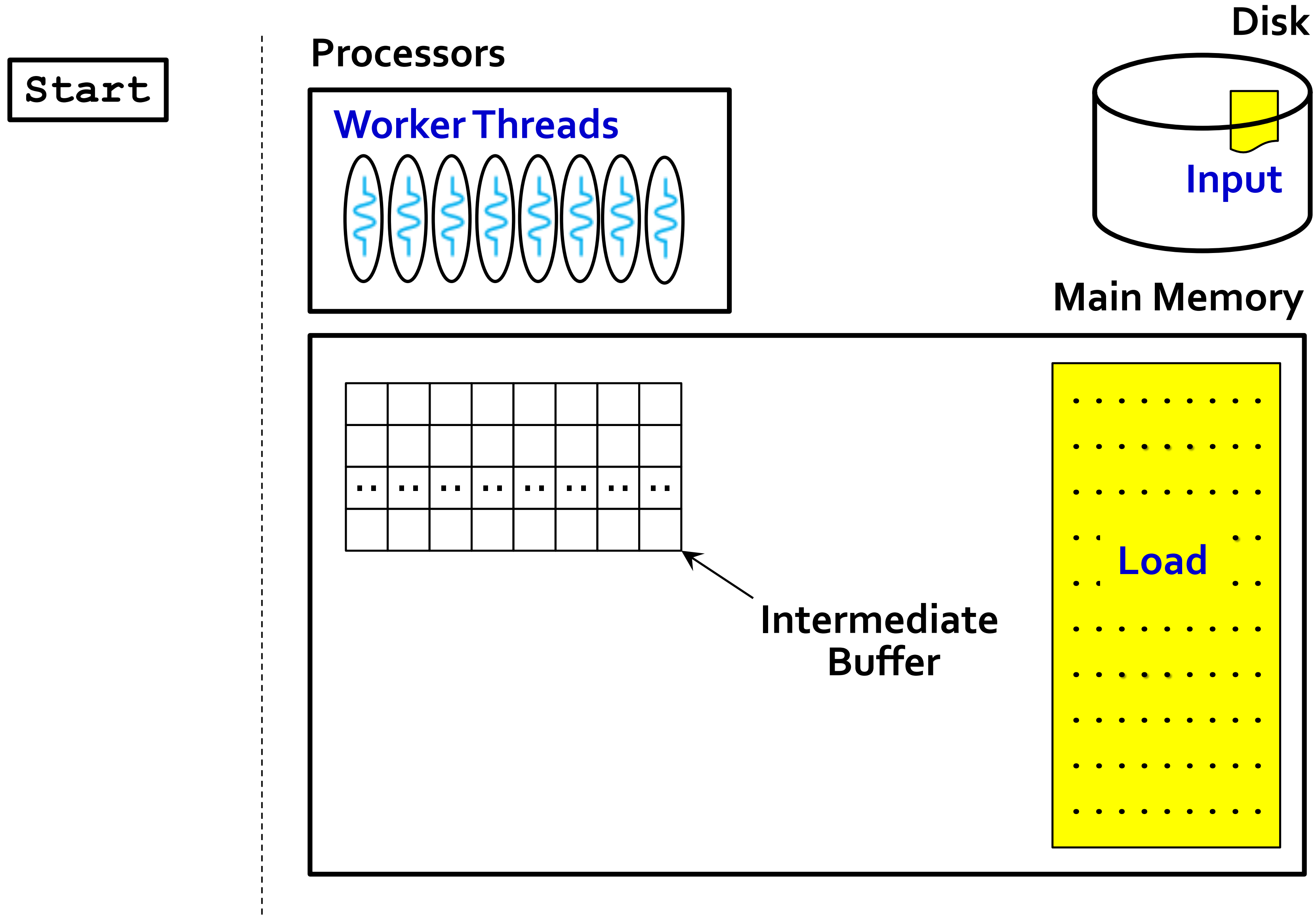


Prototype of Tiled-MapReduce

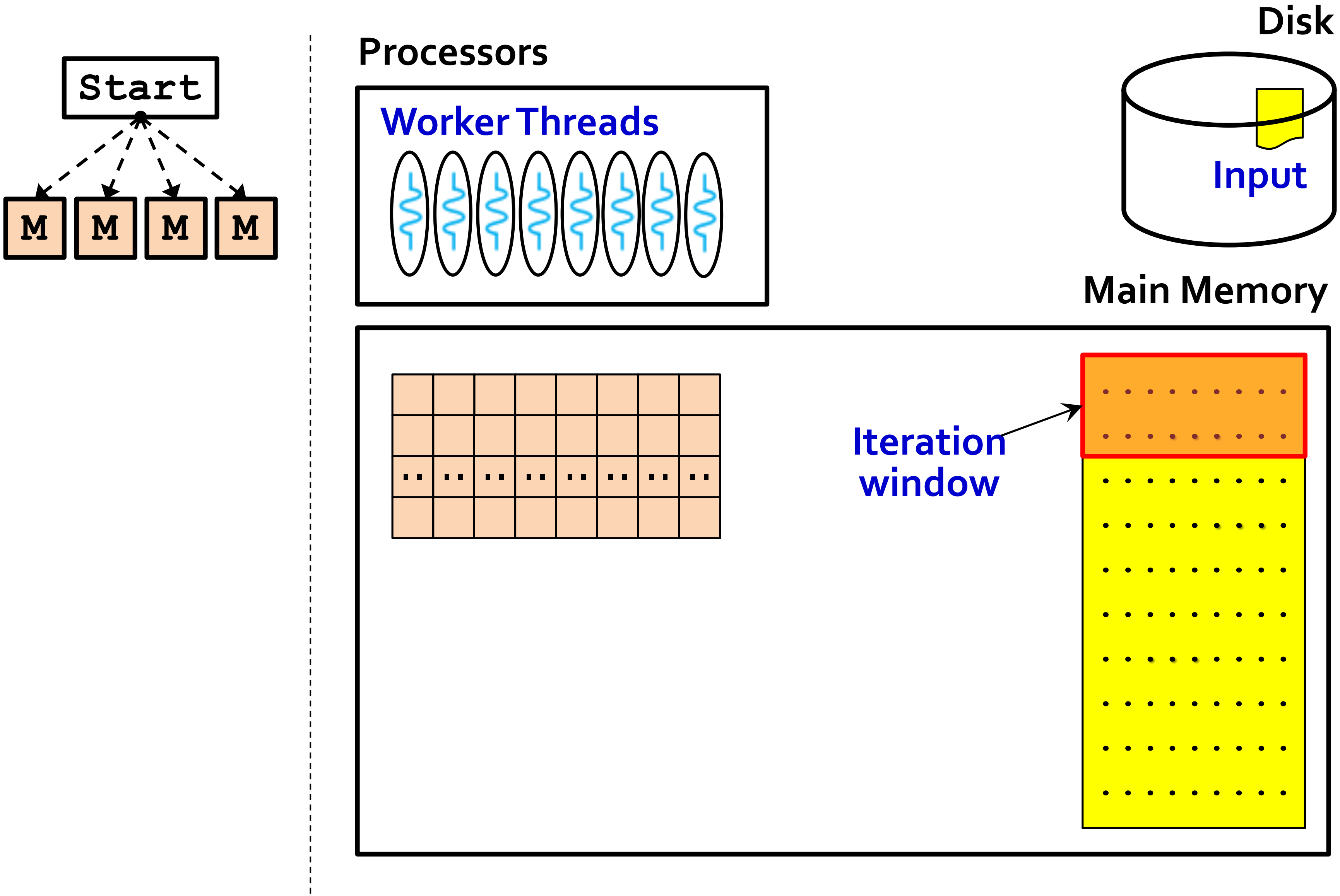
Ostrich: a prototype of Tiled-MapReduce programming model

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

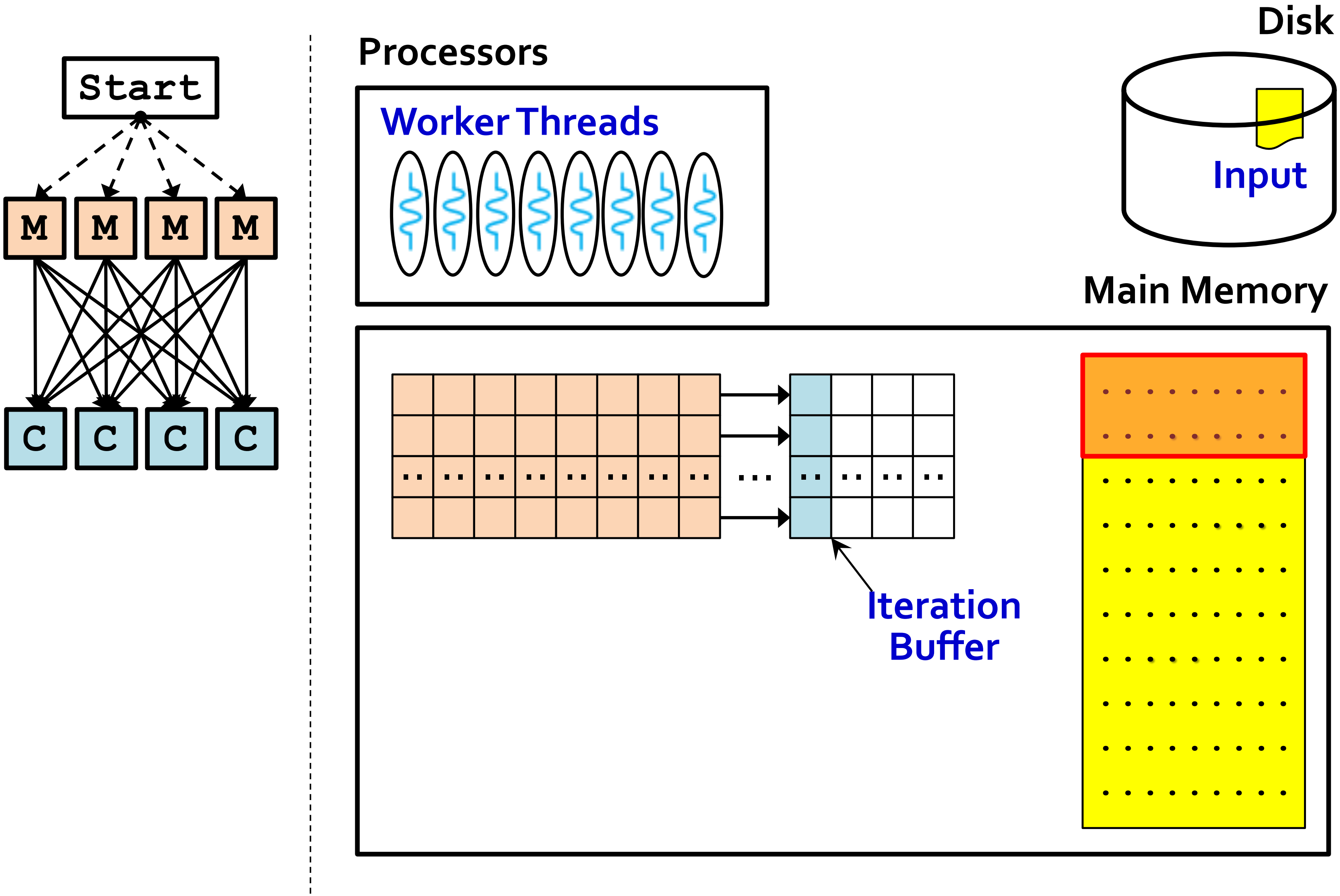
Ostrich Implementation



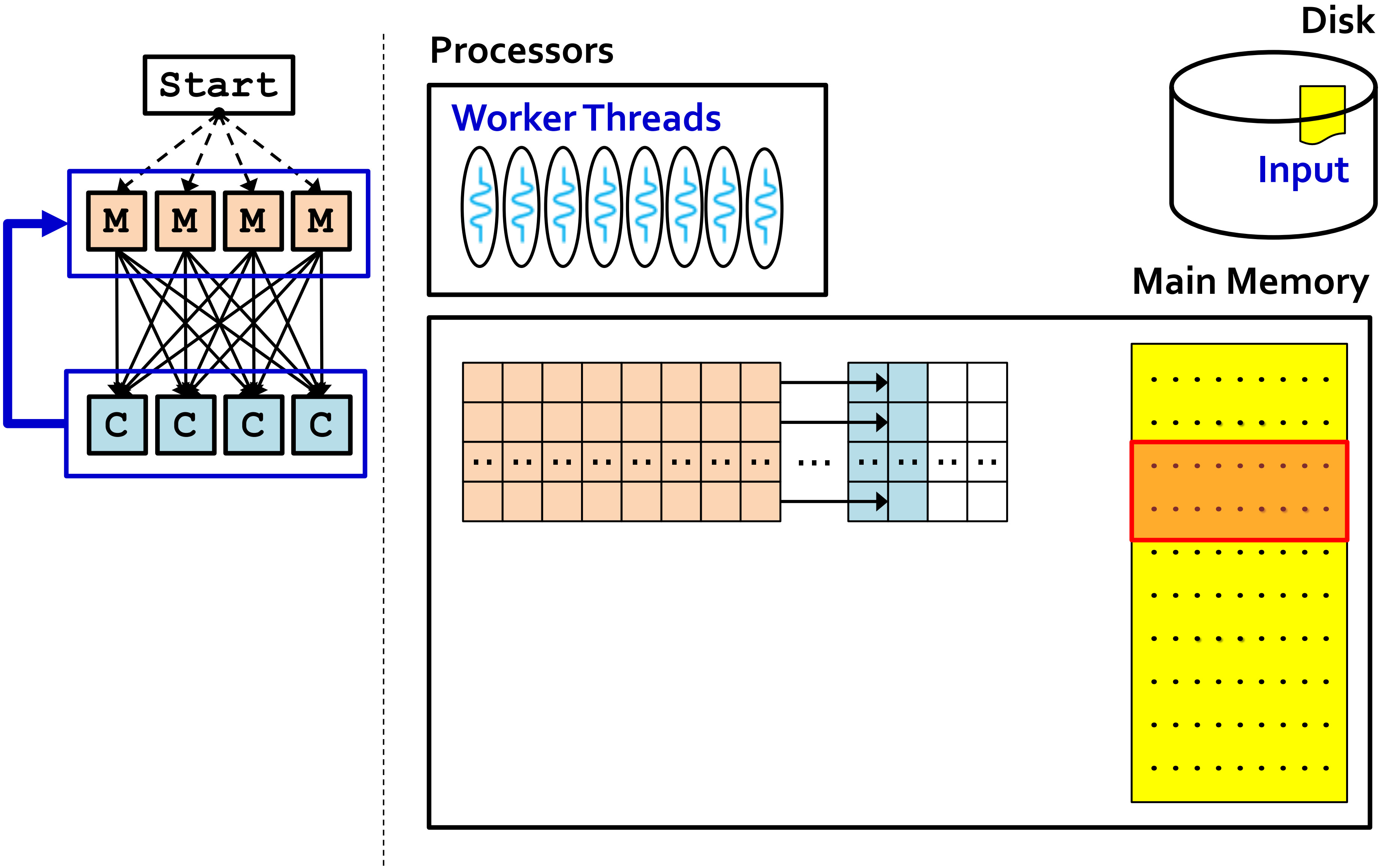
Ostrich Implementation



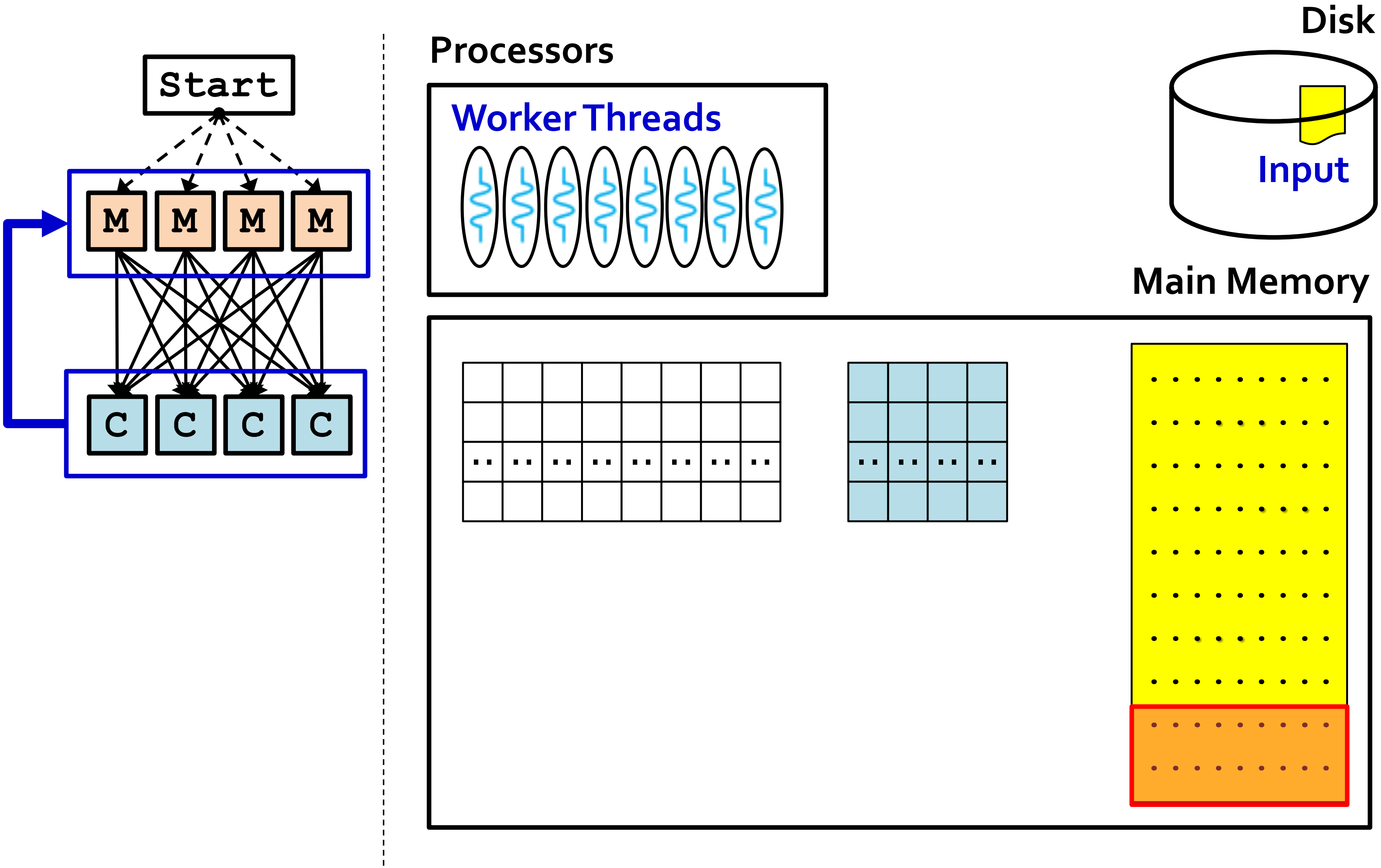
Ostrich Implementation



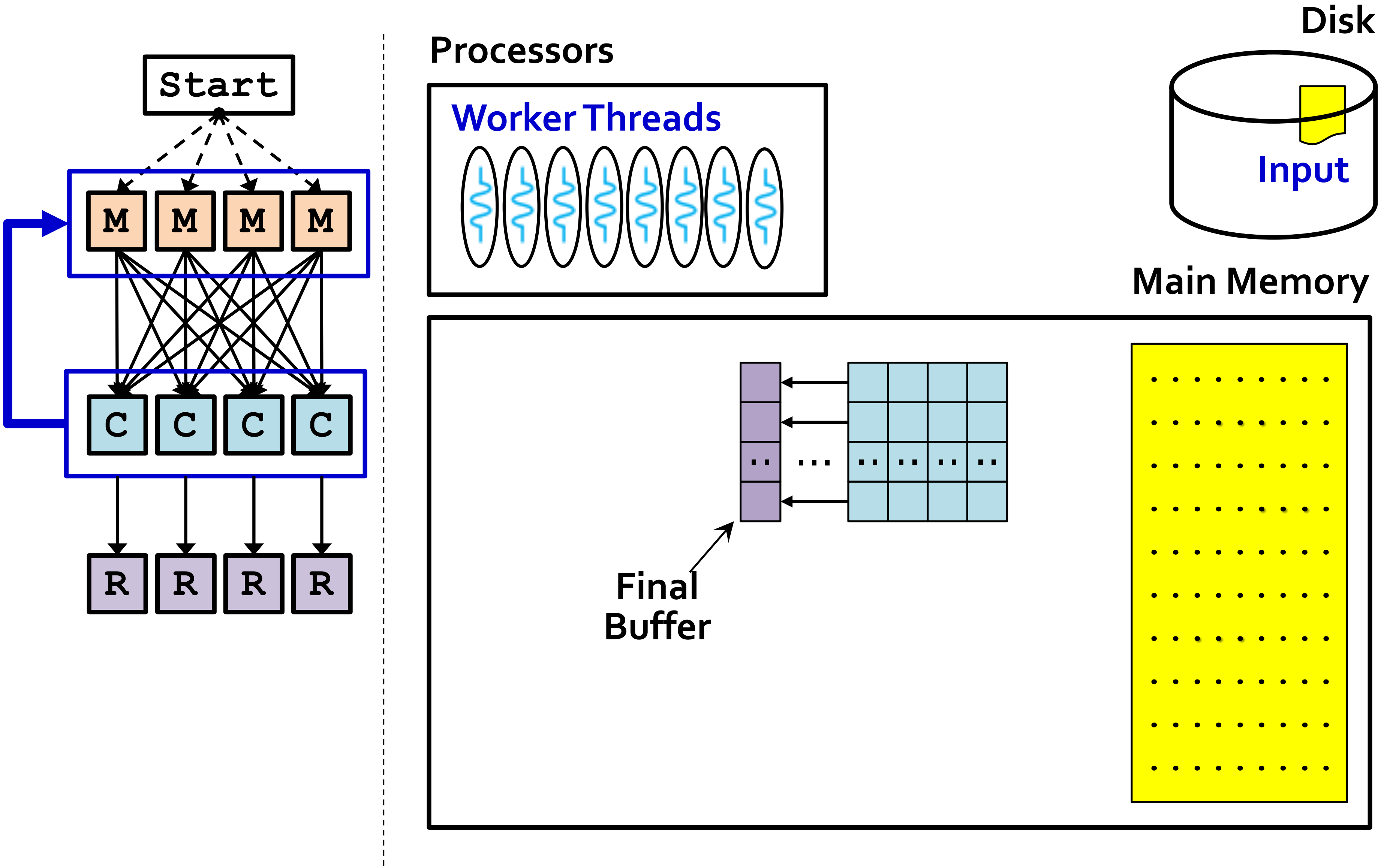
Ostrich Implementation



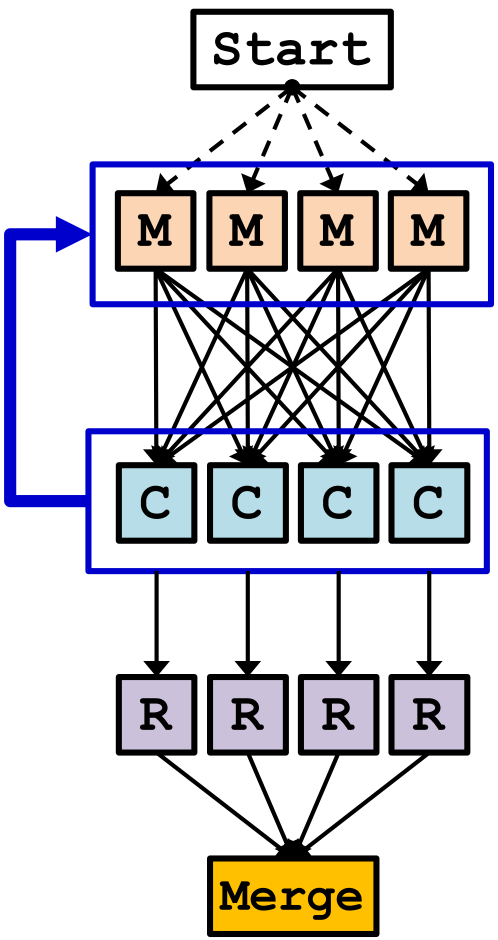
Ostrich Implementation

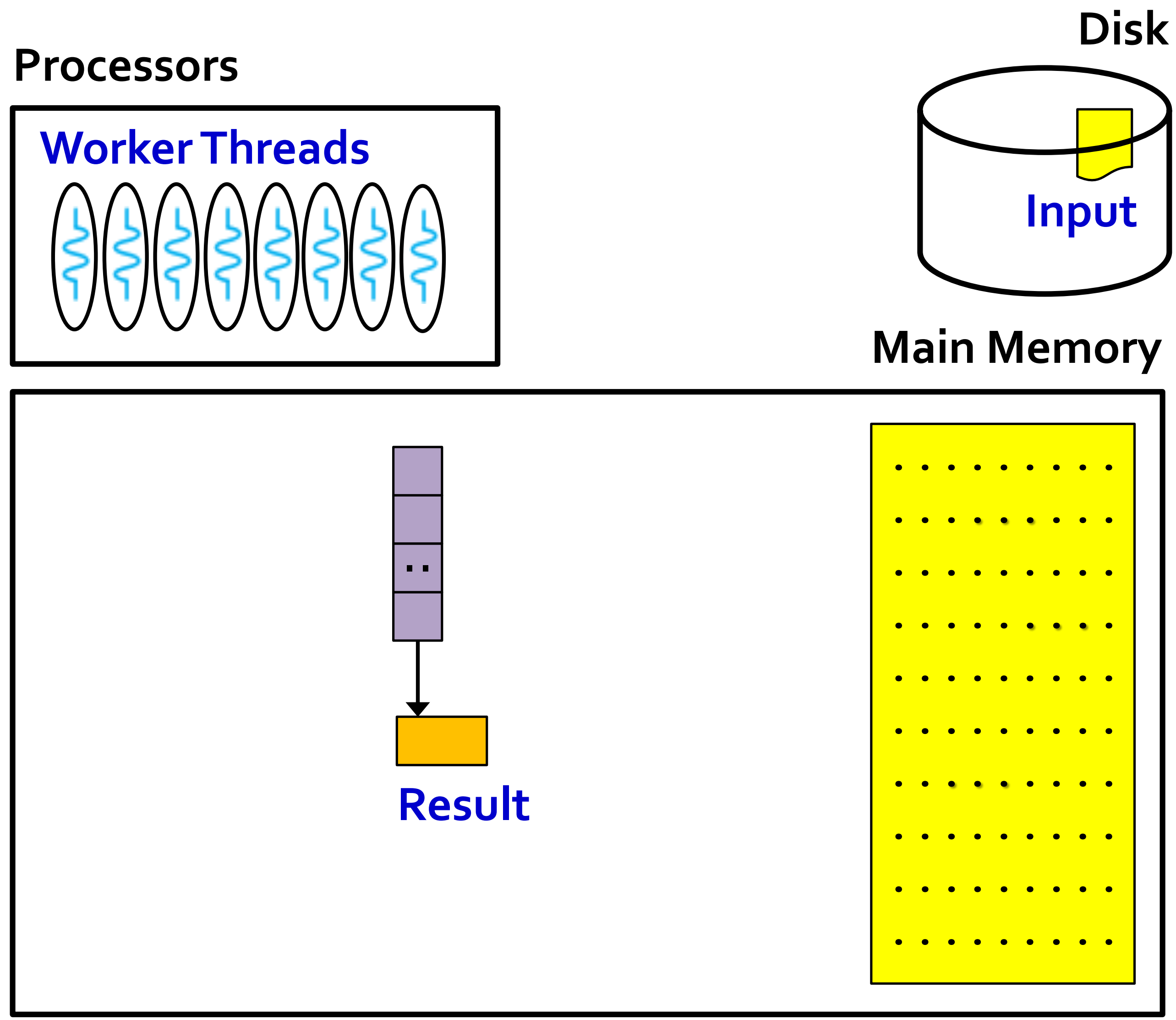


Ostrich Implementation

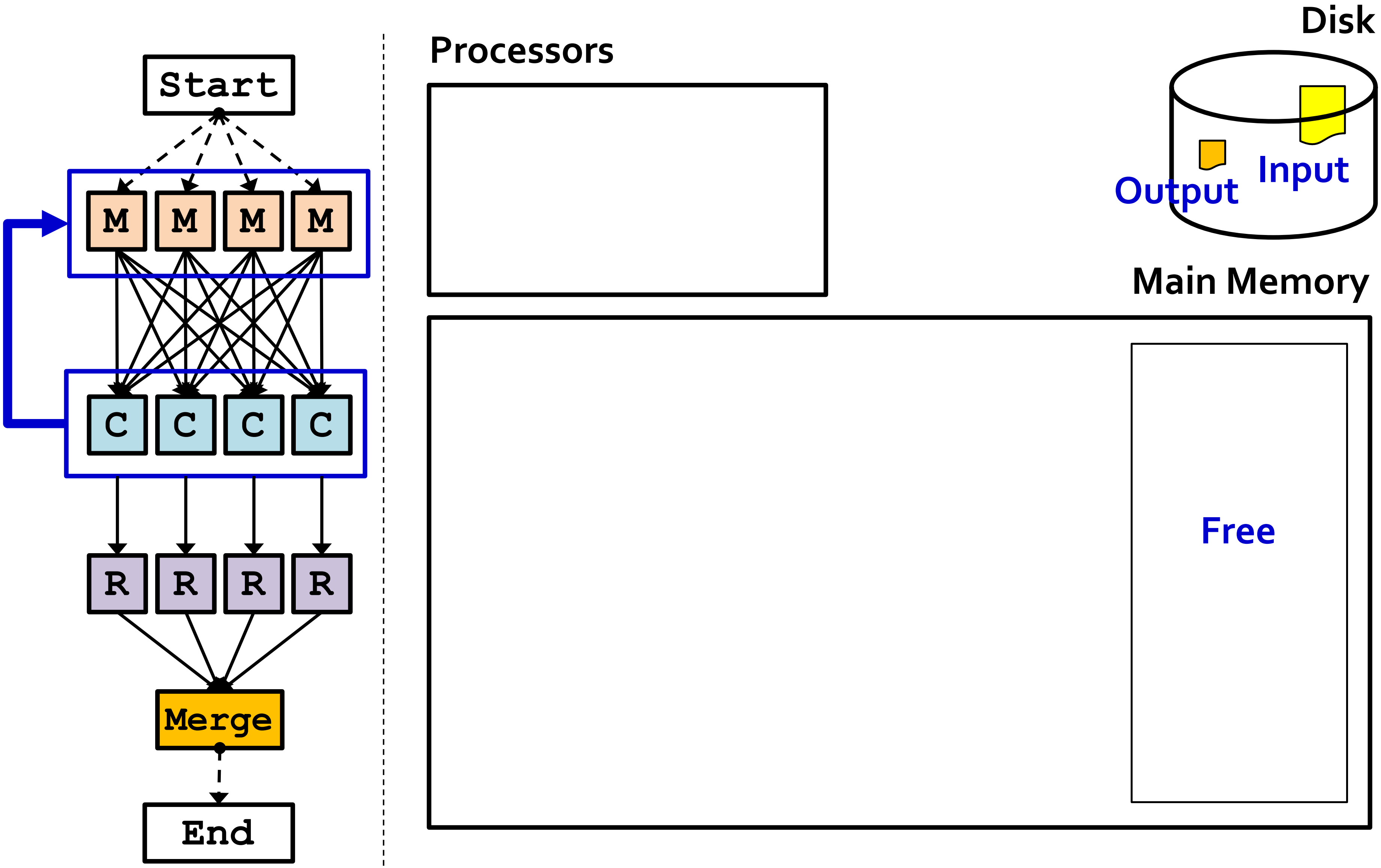


Ostrich Implementation





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

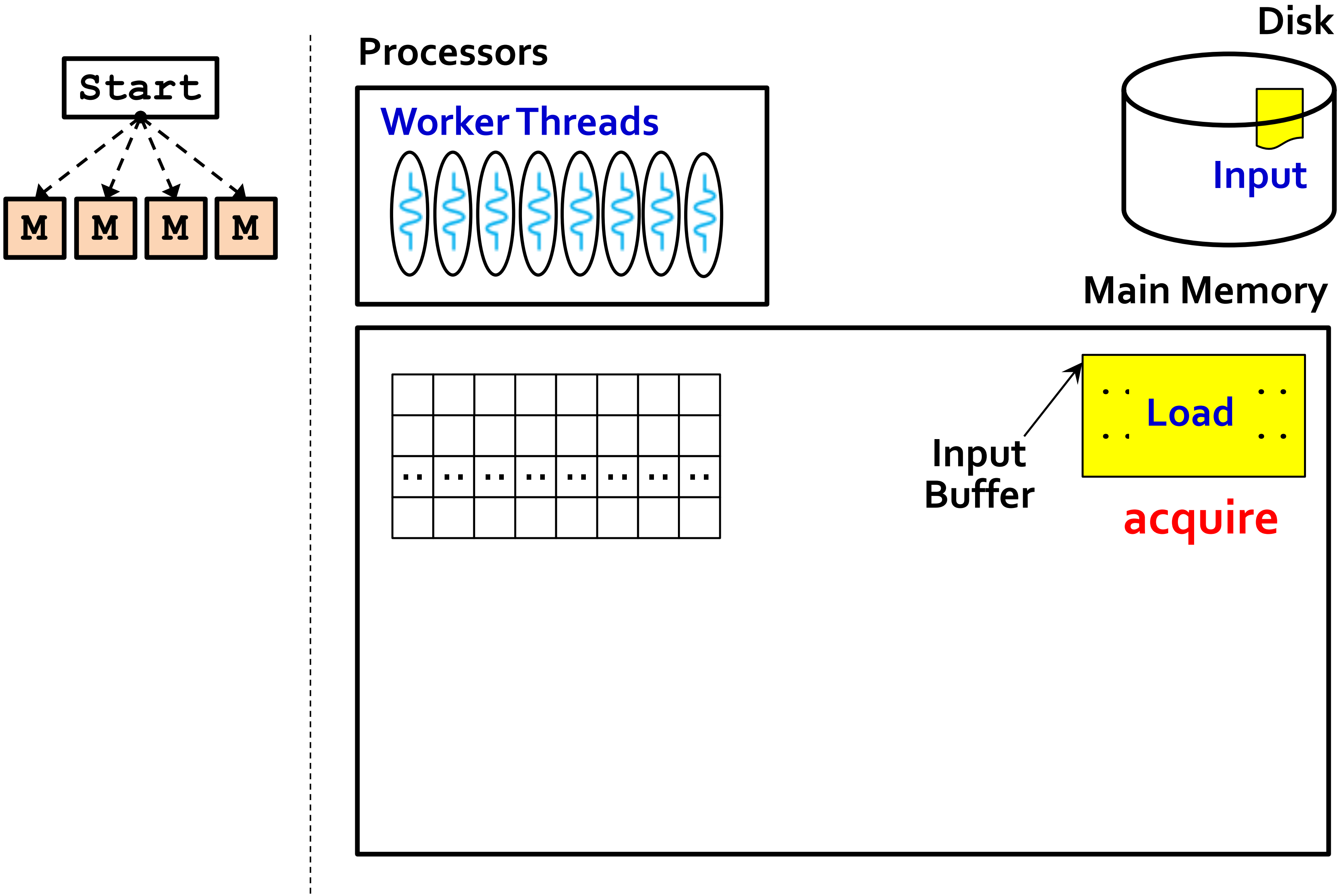
  Acquire: `load input data to memory`

  Release: `free input data from memory`

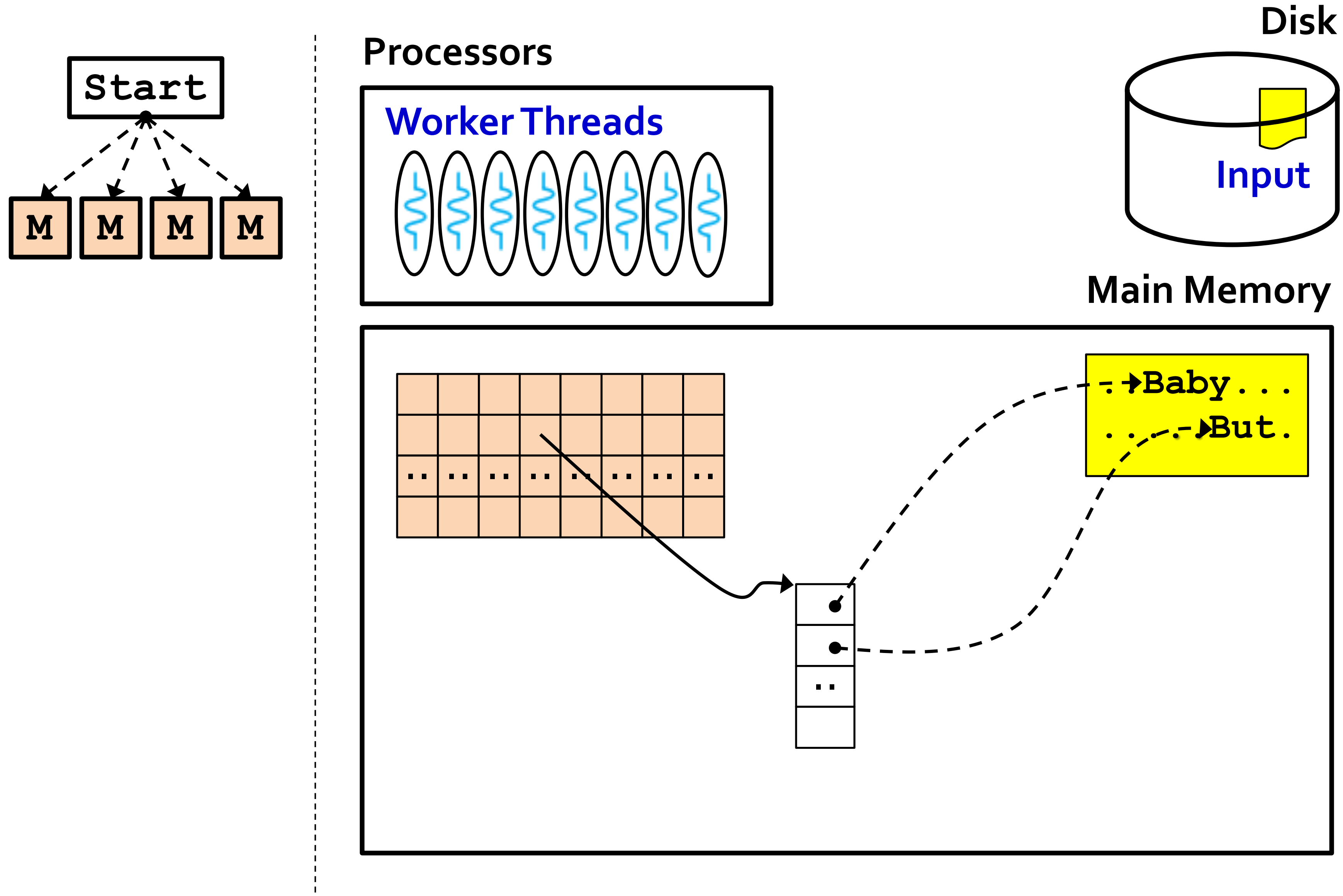
- The counterparts in other runtimes

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

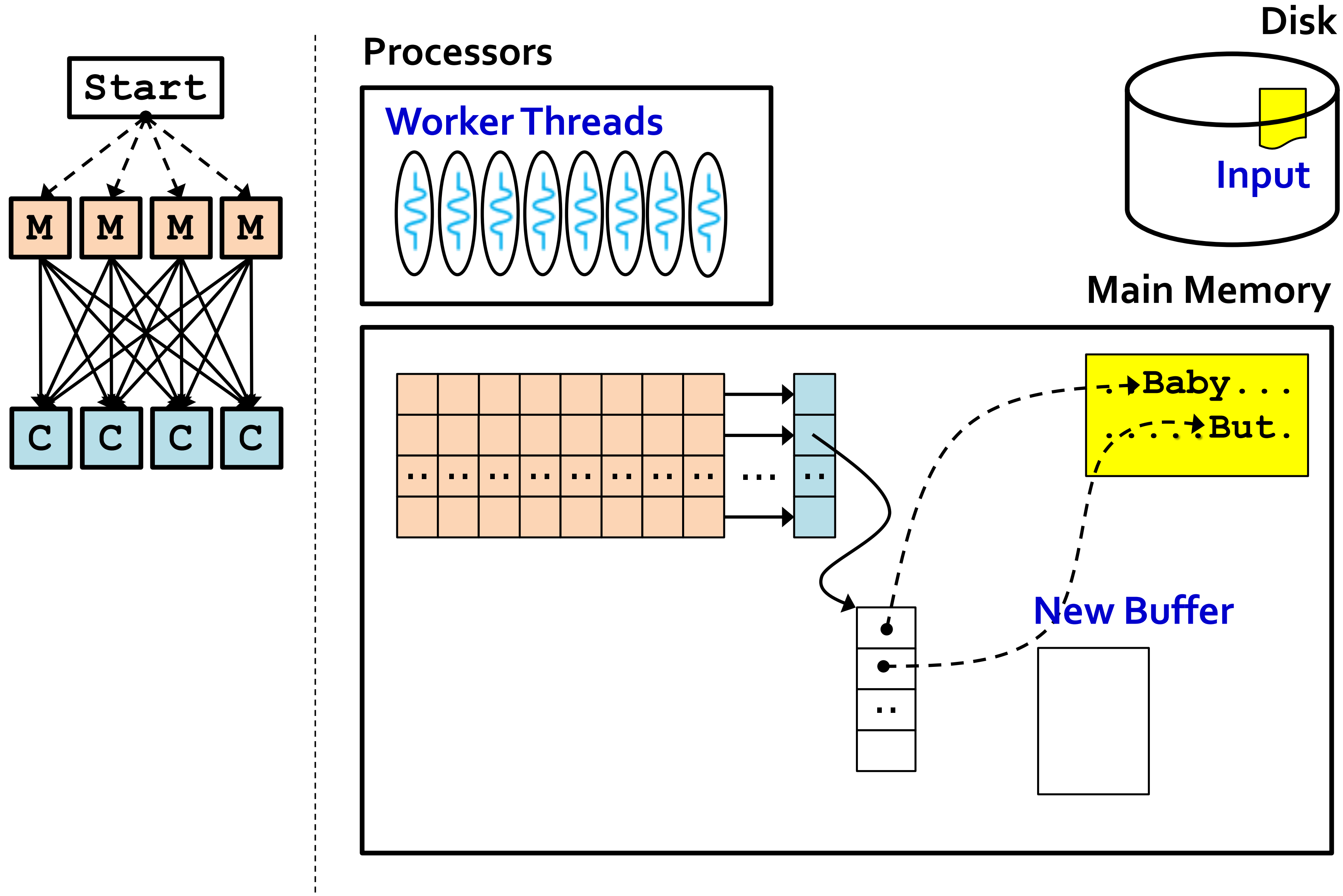
Input Data Memory Reuse



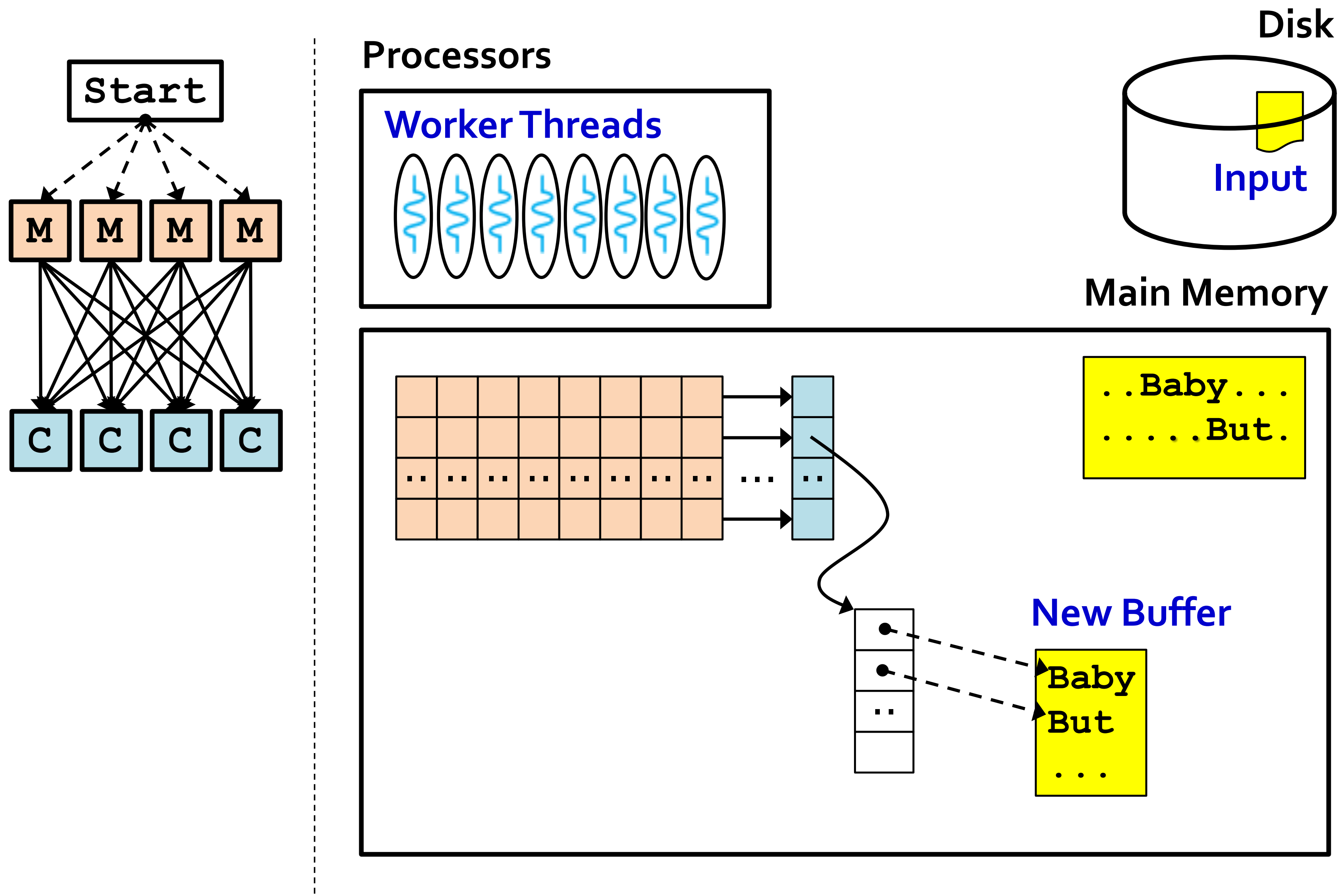
Input Data Reuse



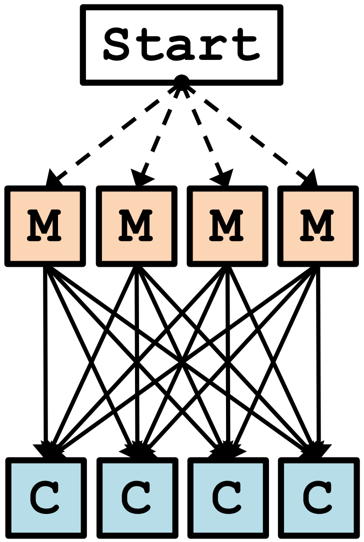
Input Data Reuse

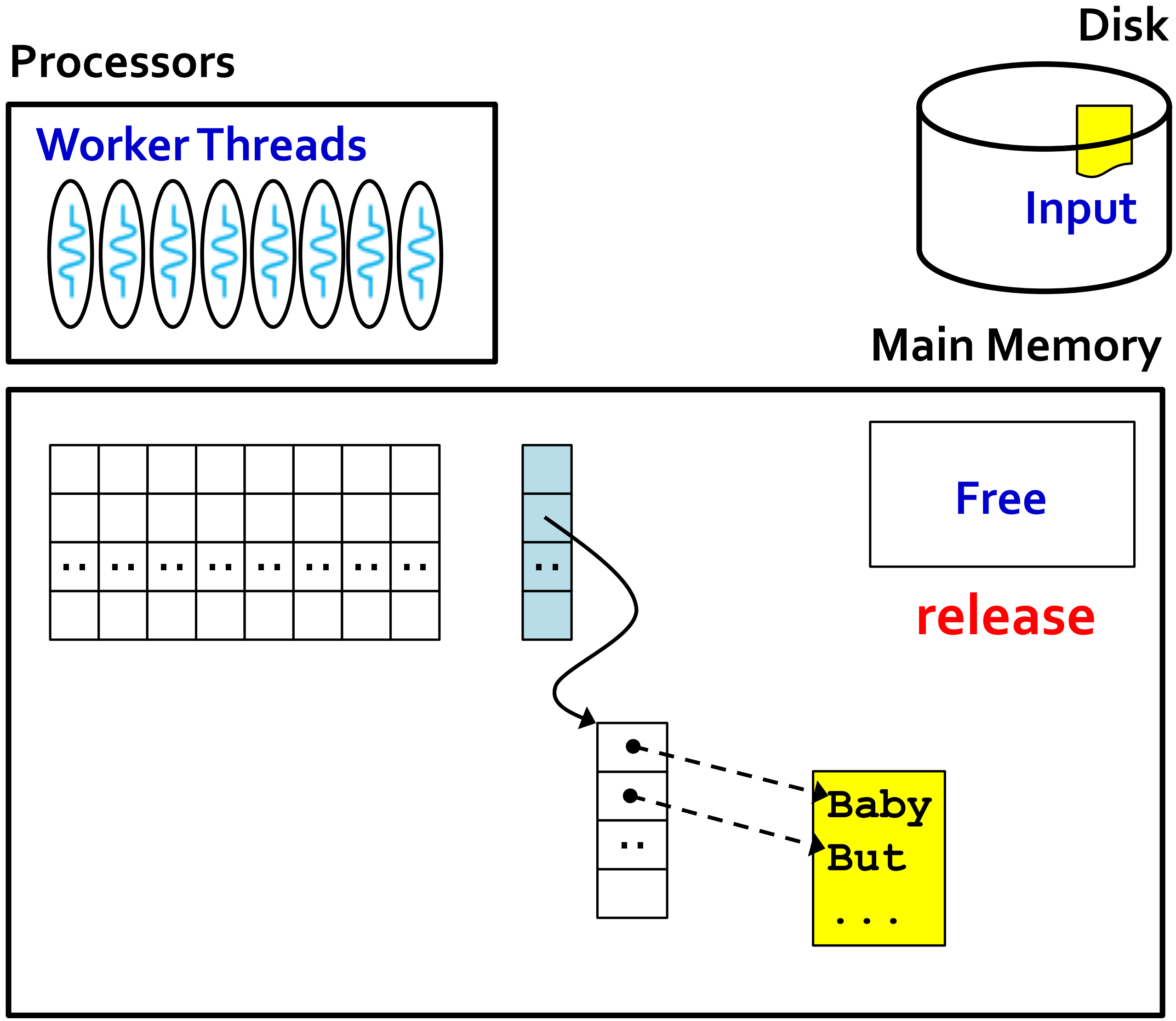


Input Data Reuse



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*

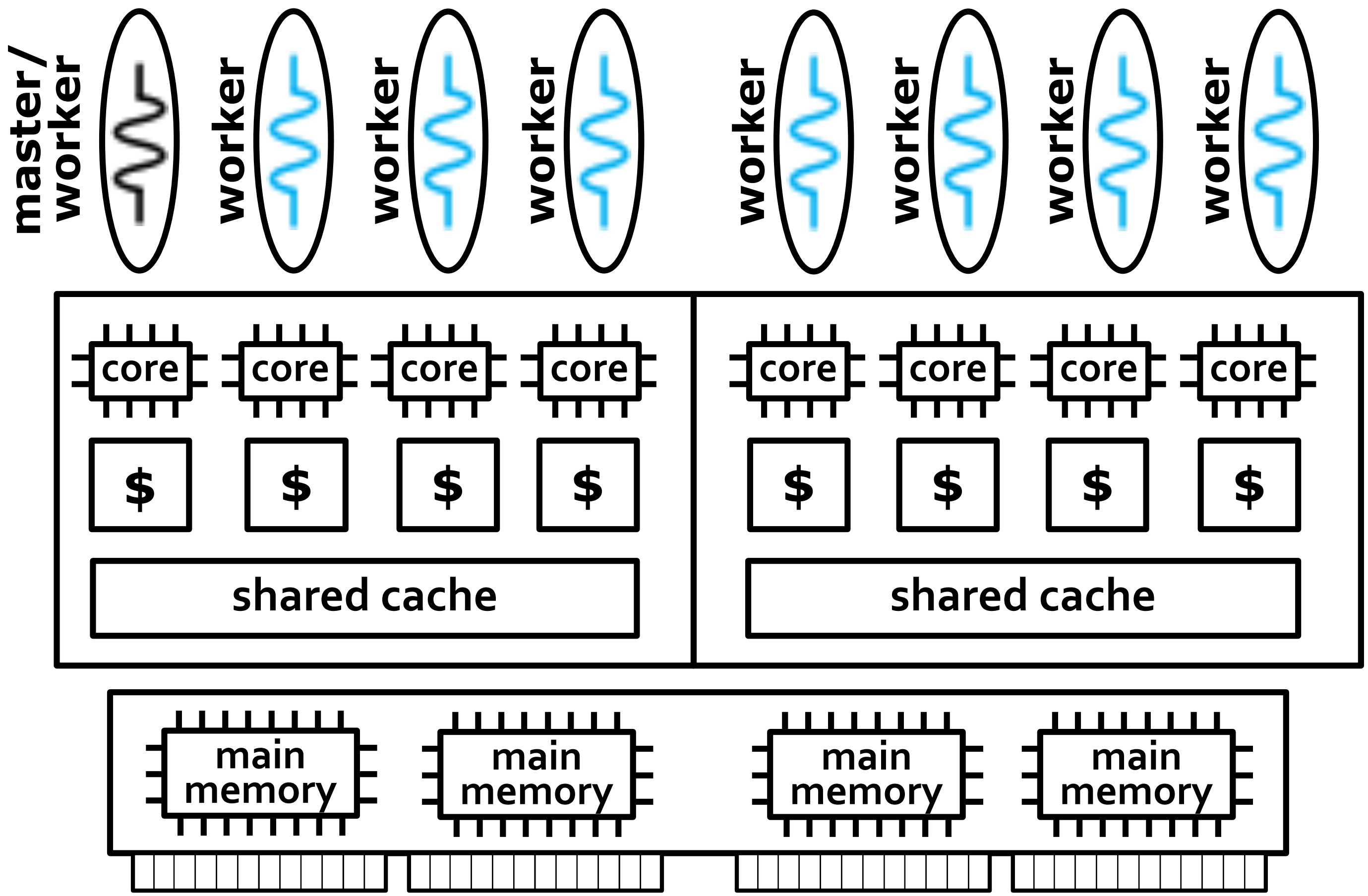
`e.g. Local/Remote L2 cache: 14/110 cycles*`

NUCA/NUMA-aware scheduler

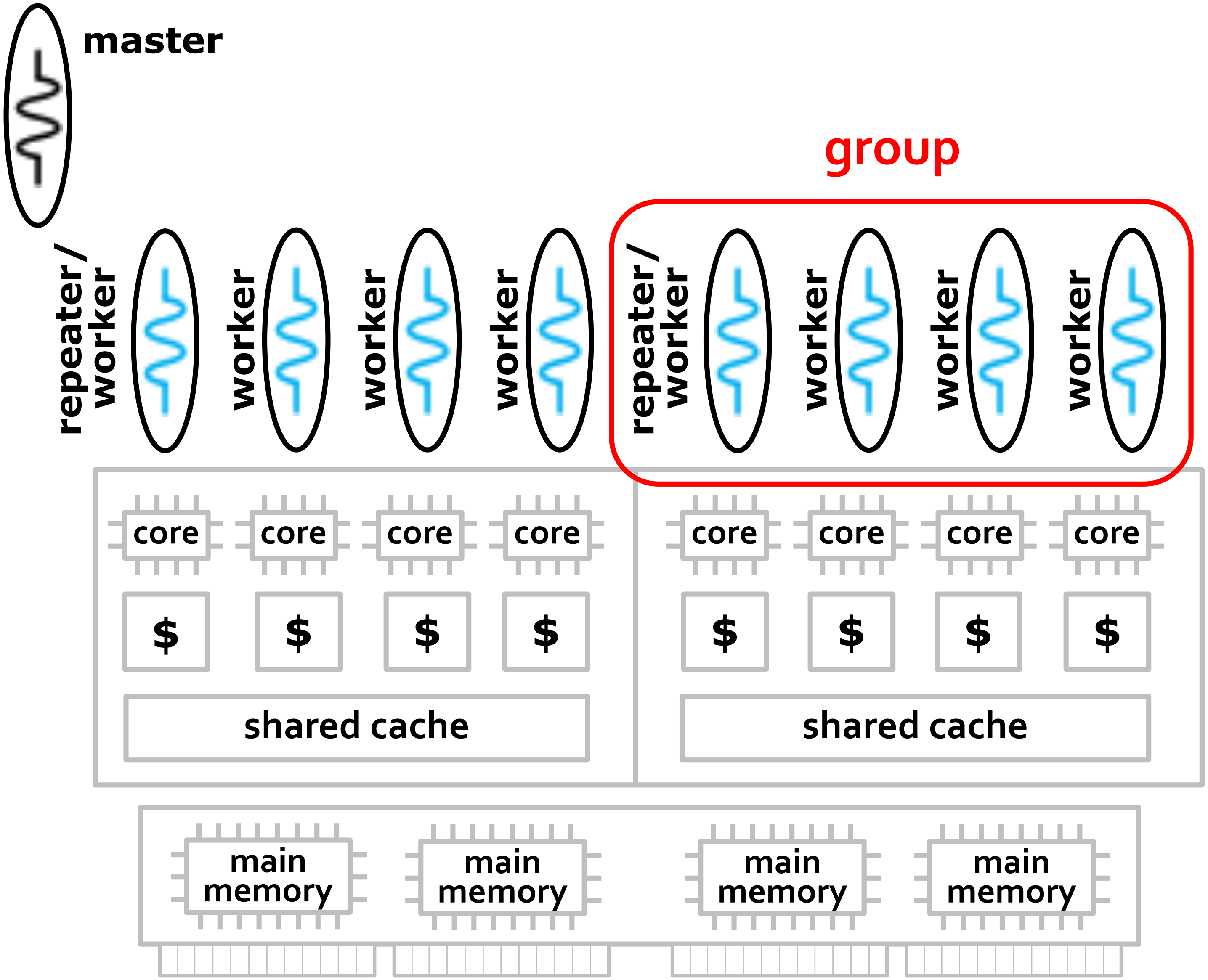
- Eliminate remote cache and memory access
- Run each sub-job on a single chip

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

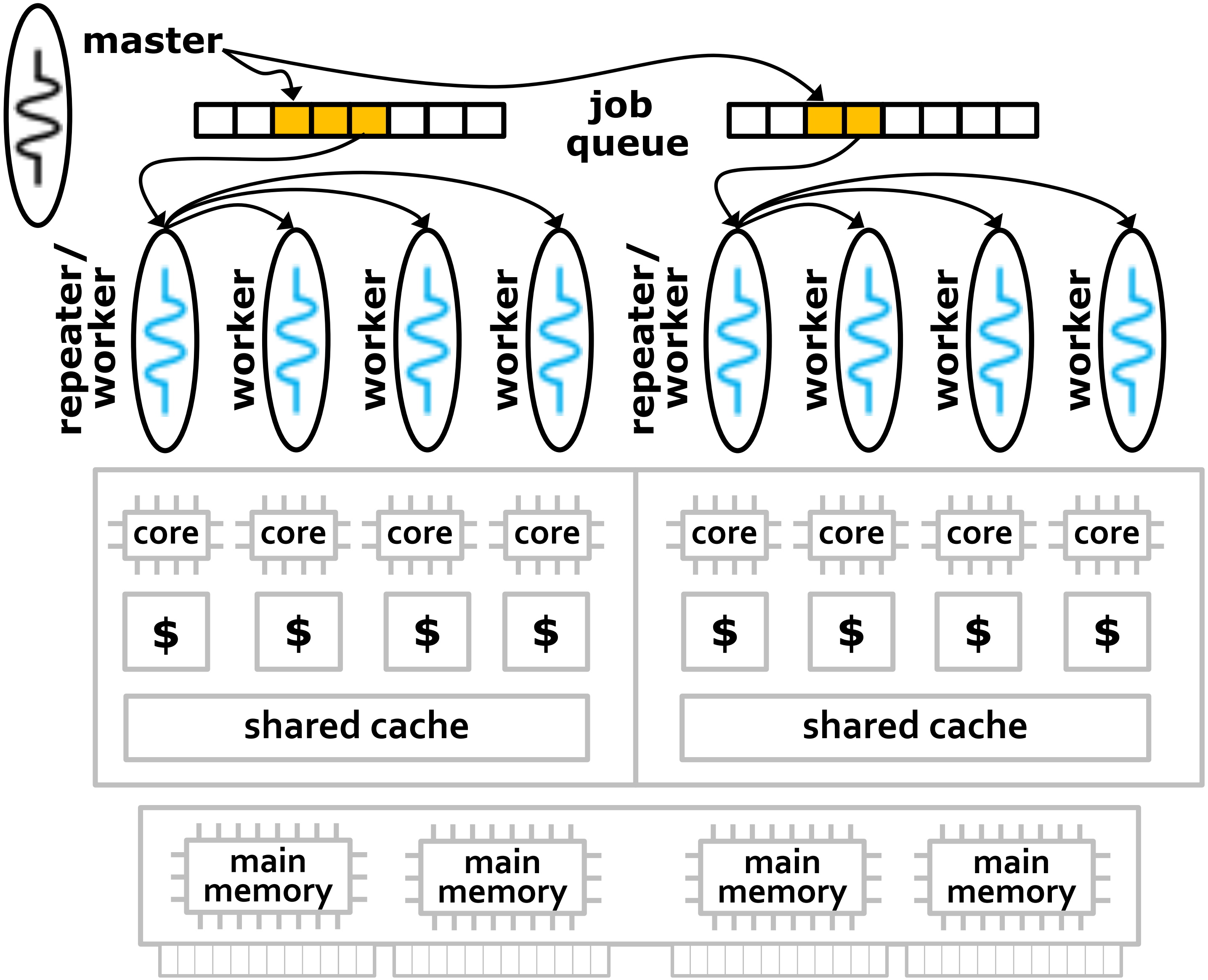
NUCA/NUMA-Aware Scheduler



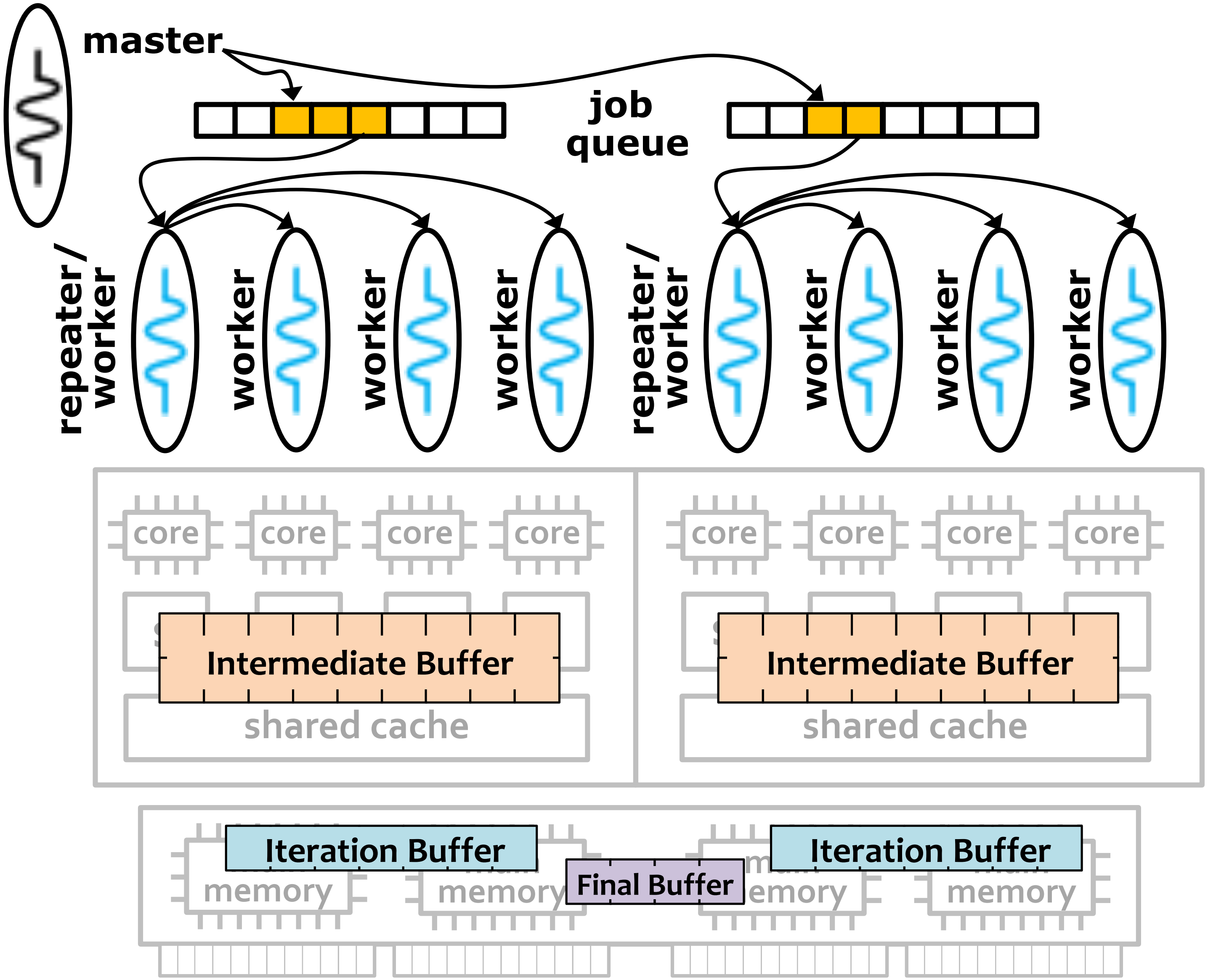
NUCA/NUMA-Aware Scheduler



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

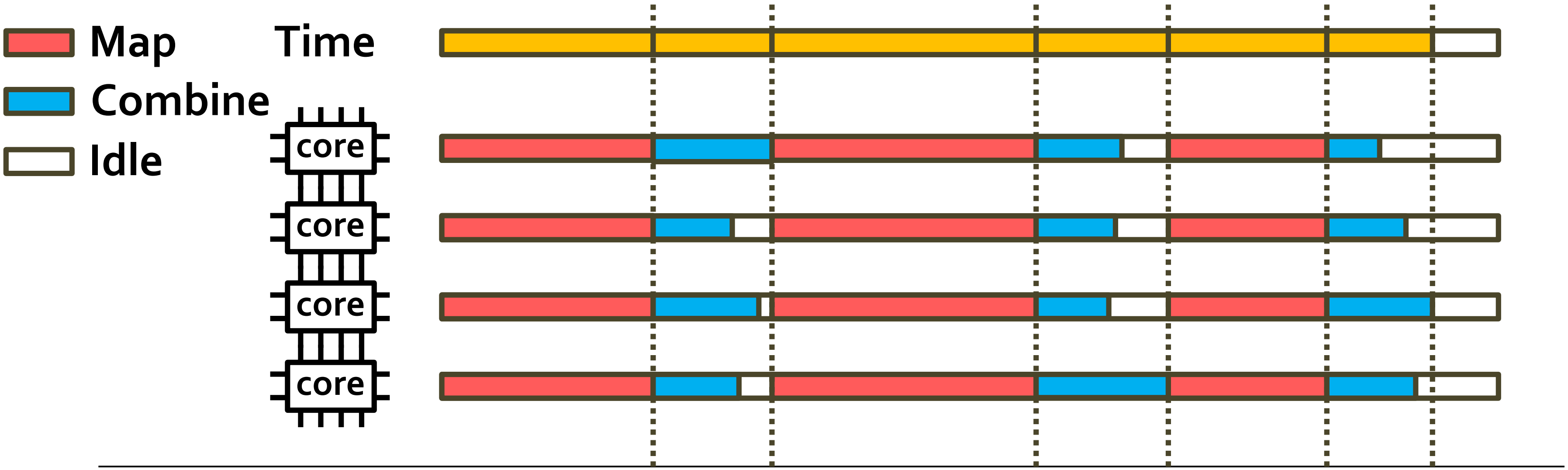
- No data dependency between one sub-job's *Combine* phase and its *successor's* *Map* phase

OPT3: CPU Optimization

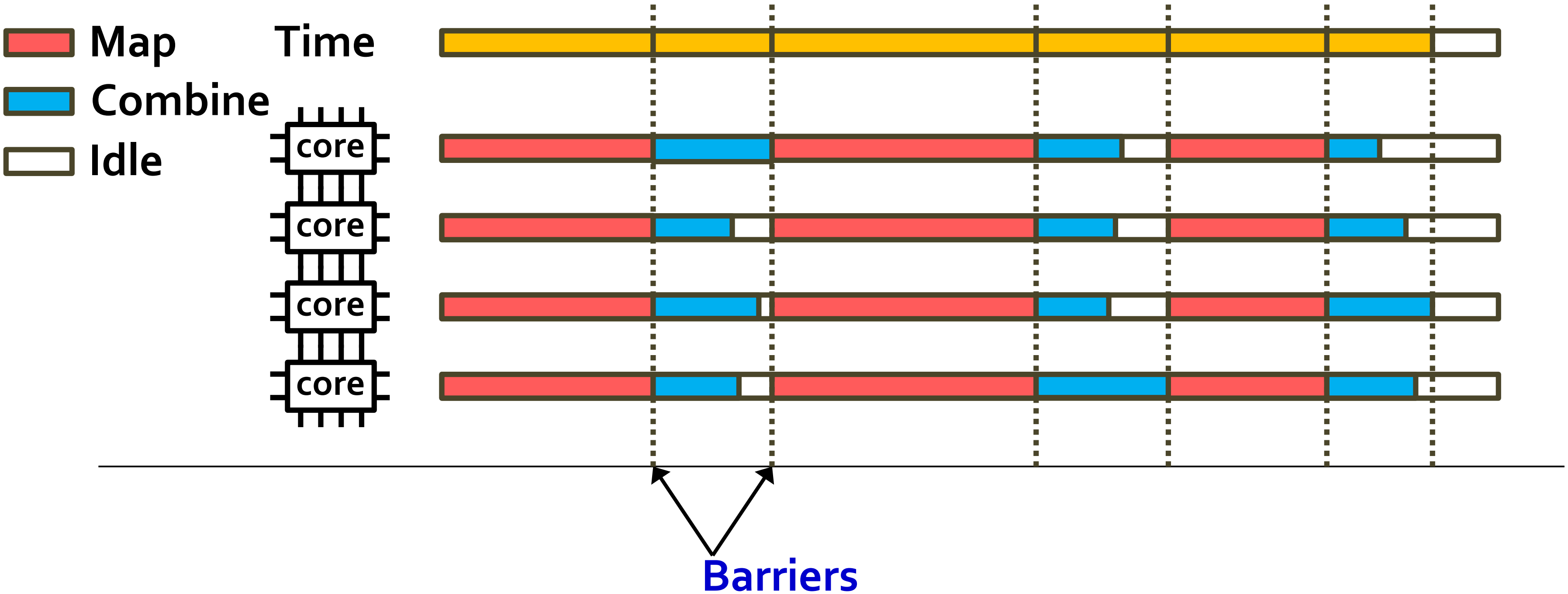
Software Pipeline

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

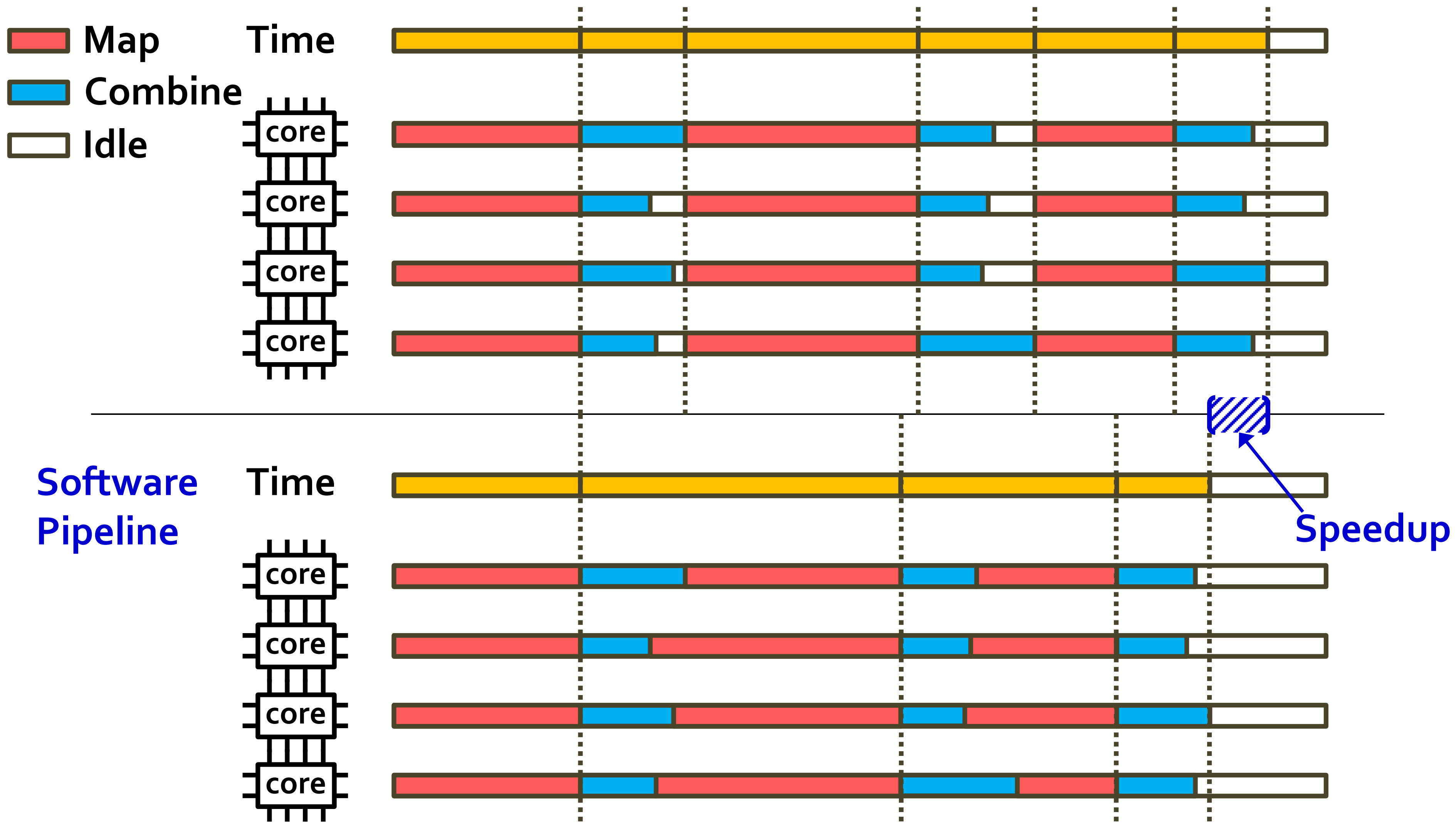
Software Pipeline



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

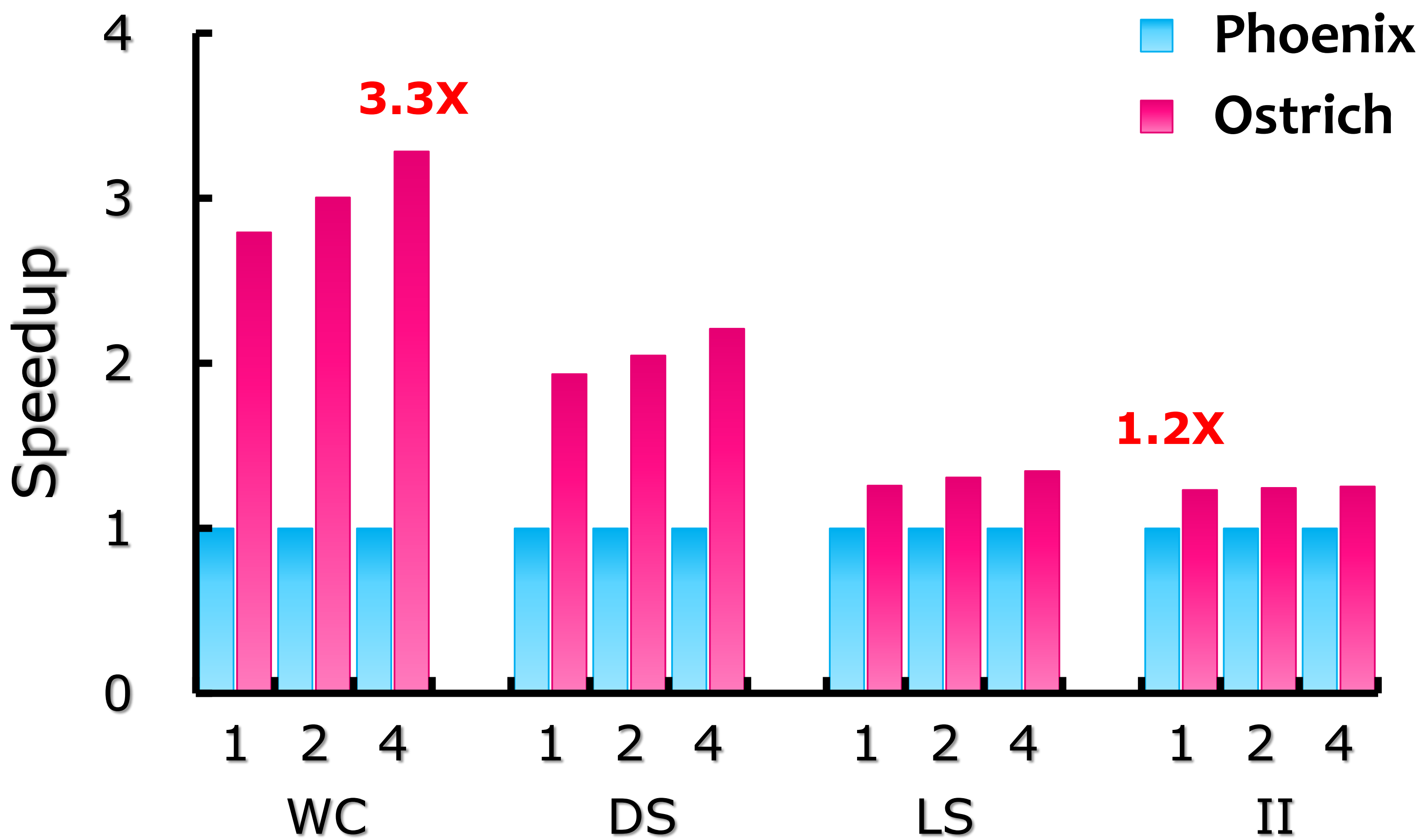
Burden of Programmer

Code Modification

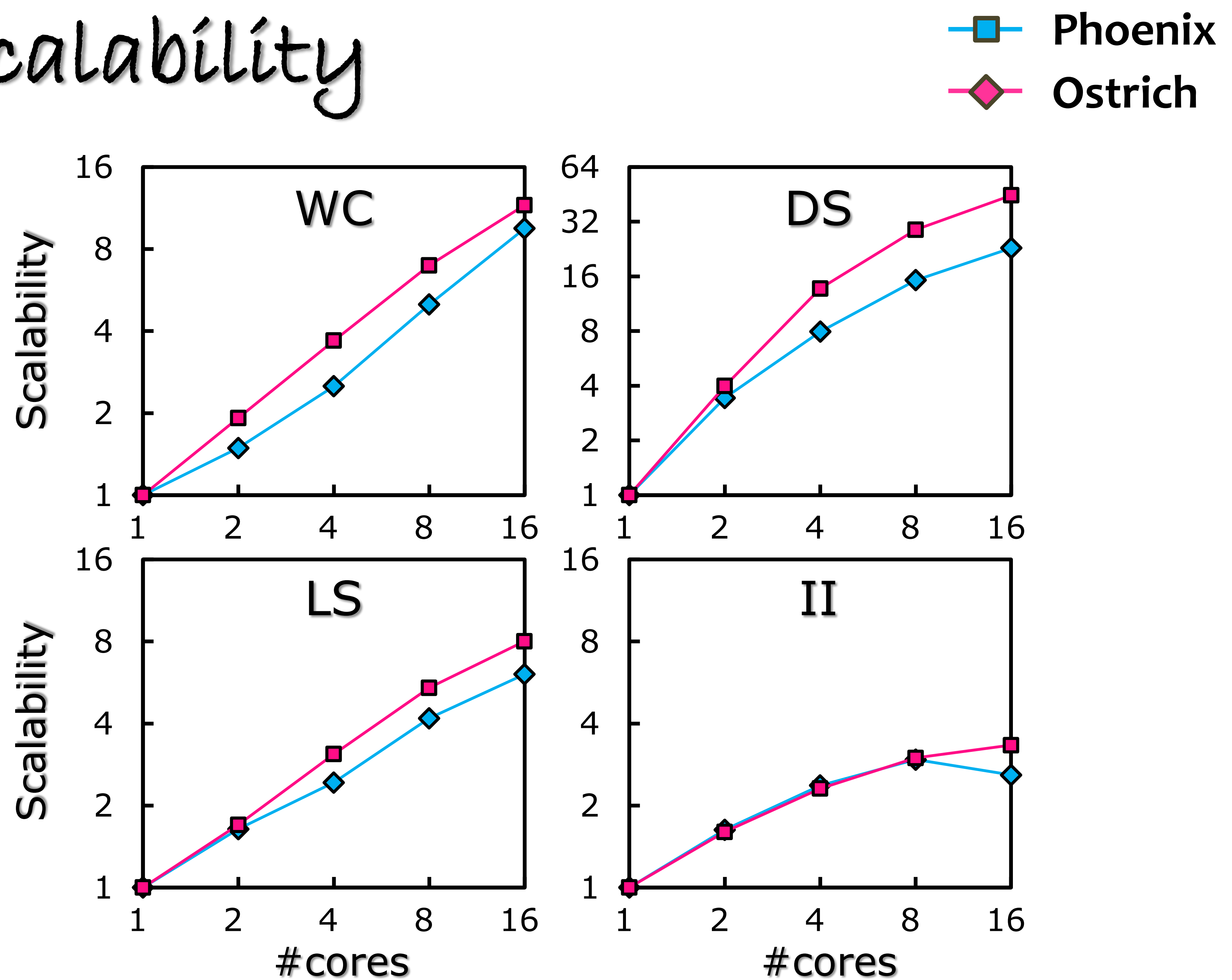
- Support input data memory reuse

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

Overall Performance



Scalability



Related Work

Other extension to MapReduce model

- Database: Map-reduce-merge [SIGMOD'07]
- Online Aggregation: *MapReduce Online* [NSDI'10]
- *…*

Other implementation of MapReduce runtime

- Cluster: *Hadoop* [Apache, OSDI'08]
- Shared Memory: *Phoenix* [HPCA'07, IISWC'09] and *Metis* [MIT-TR]
- GPGPU: *Mars* [PACT'07]
- Heterogeneous: *MapCG* [PACT'10]
- *…*

Conclusion

- Environments differences between *cluster* and *multicore* open new design spaces and optimization opportunities

- *Tiled-MapReduce* and the three optimizations

- *Ostrich* outperforms *Phoenix* by up to 3.3X

Thanks

Ostrich

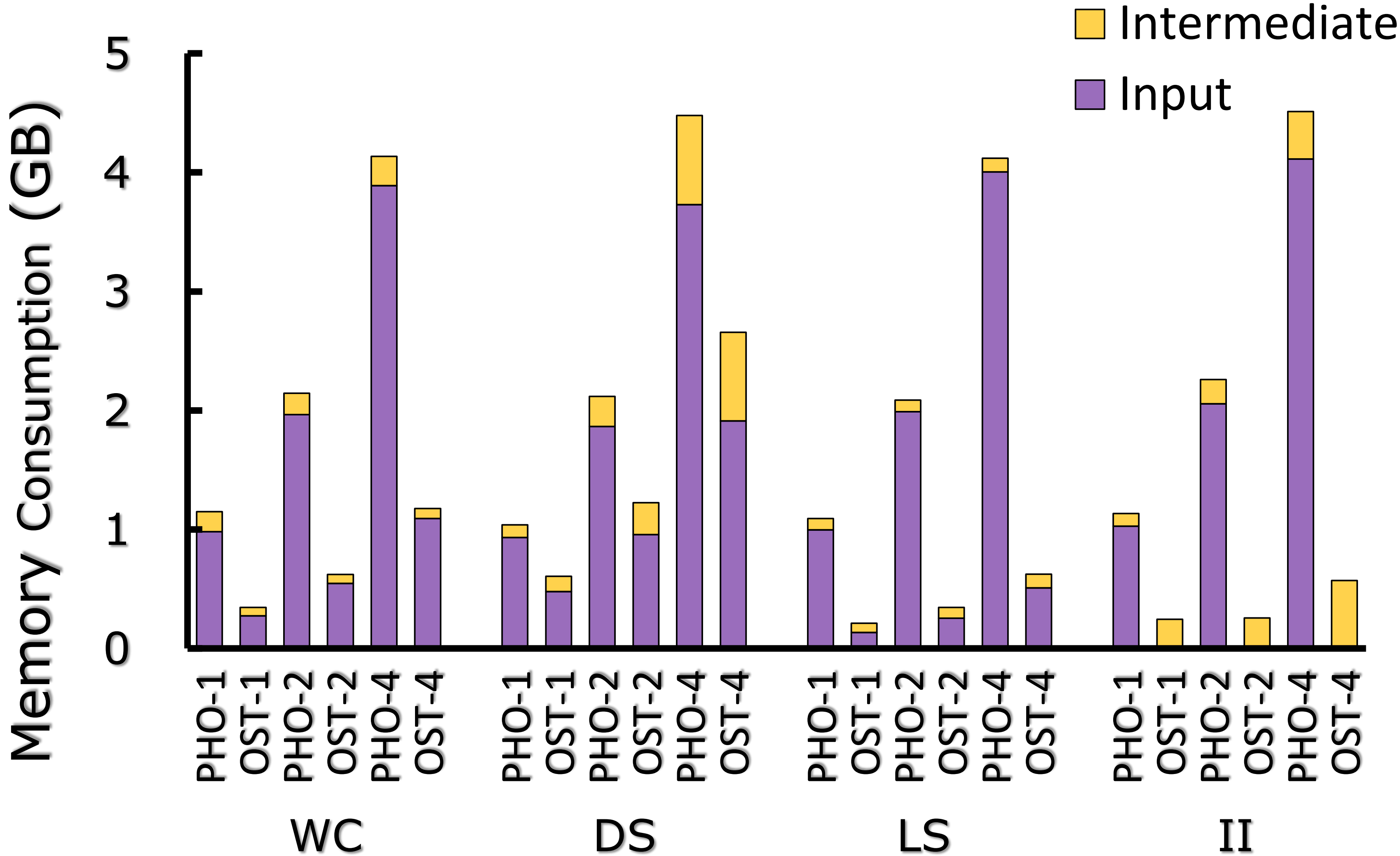
The top land speed
and the largest of bird

Parallel Processing Institute

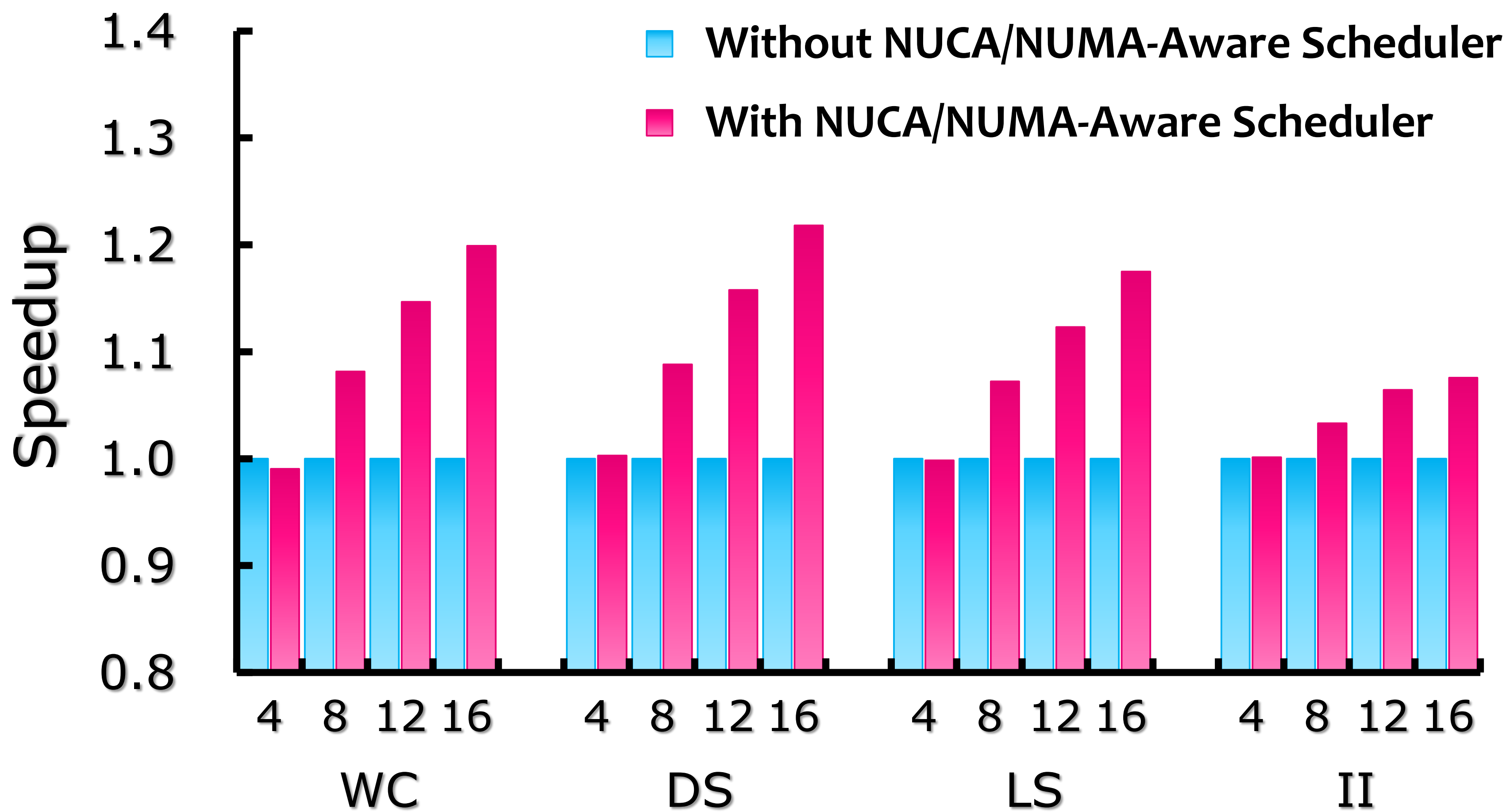
http://ppi.fudan.edu.cn

Questions ?

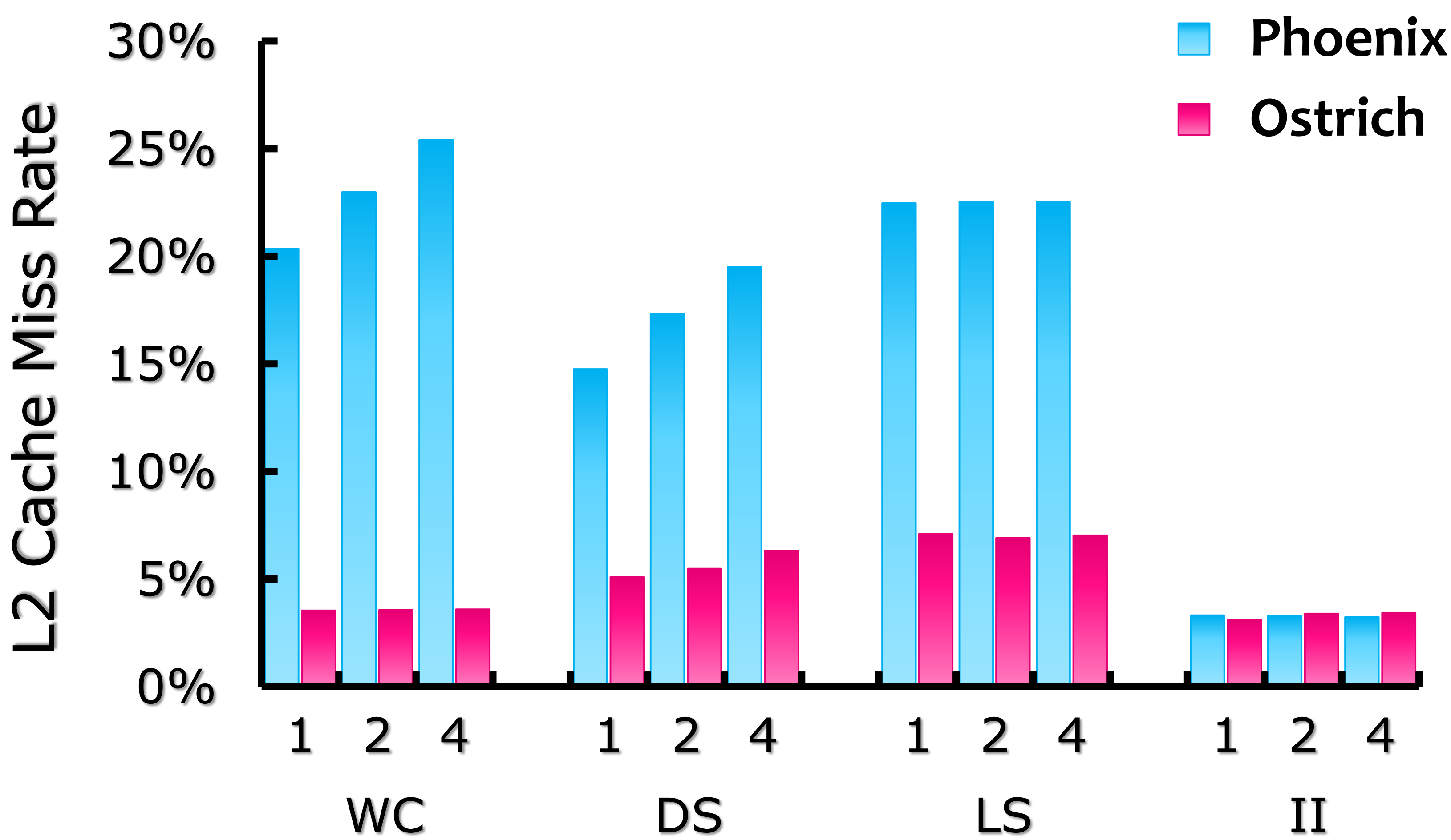
Memory Consumption



NUCA/NUMA-Aware Scheduler



Exploit Locality



Software Pipeline