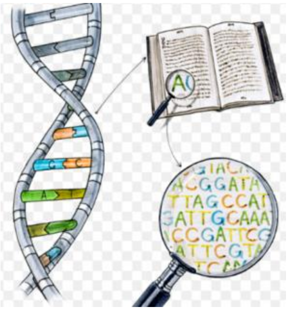
Collaborative Accelerators for In-Memory MapReduce on Scale-up Machines

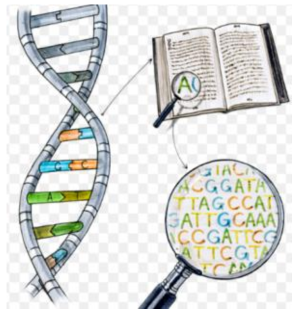
Abraham Addisie, Valeria Bertacco
University of Michigan

ASP-DAC 2019
January 24, 2019 Tokyo, Japan

Data-intensive applications challenges



Data-intensive applications challenges



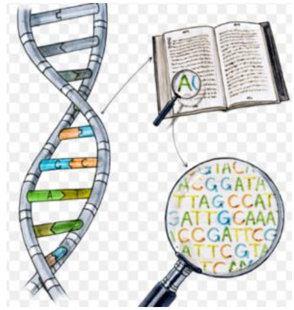
Challenge #1 :

- Complex parallel programming models (e.g. Pthreads, OpenCL, OpenMP)

Solution:

- Simplified parallel programming models (e.g. MapReduce)

Data-intensive applications challenges



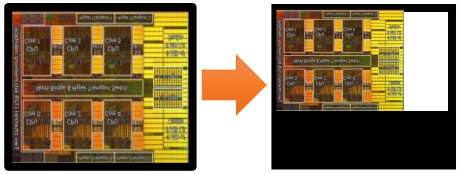
Challenge #1 :

- Complex parallel programming models (e.g. Pthreads, OpenCL, OpenMP)

Solution:

- Simplified parallel programming models (e.g. MapReduce)

Challenge #2 :



New features

Dark silicon

- Inefficiency in new hardware systems due to poor device scaling

Solution:

Accelerator-rich architectures

MapReduce – WordCount example

Input

car bus
car car
bus

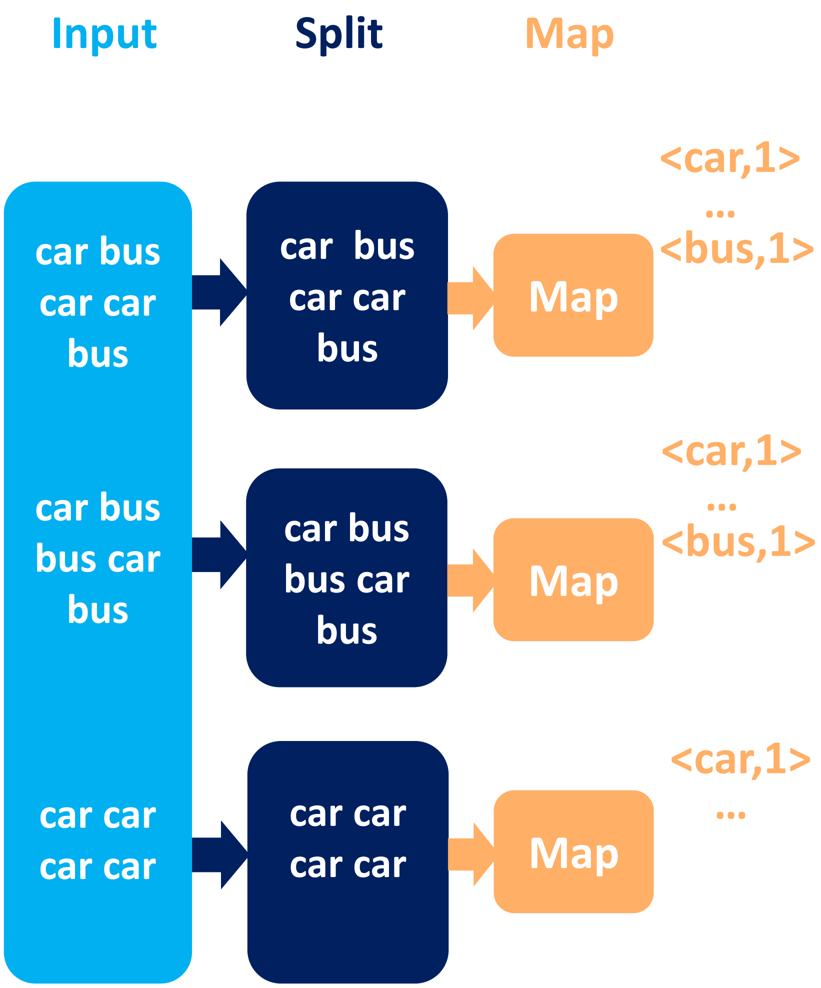
car bus
bus car
bus

car car
car car

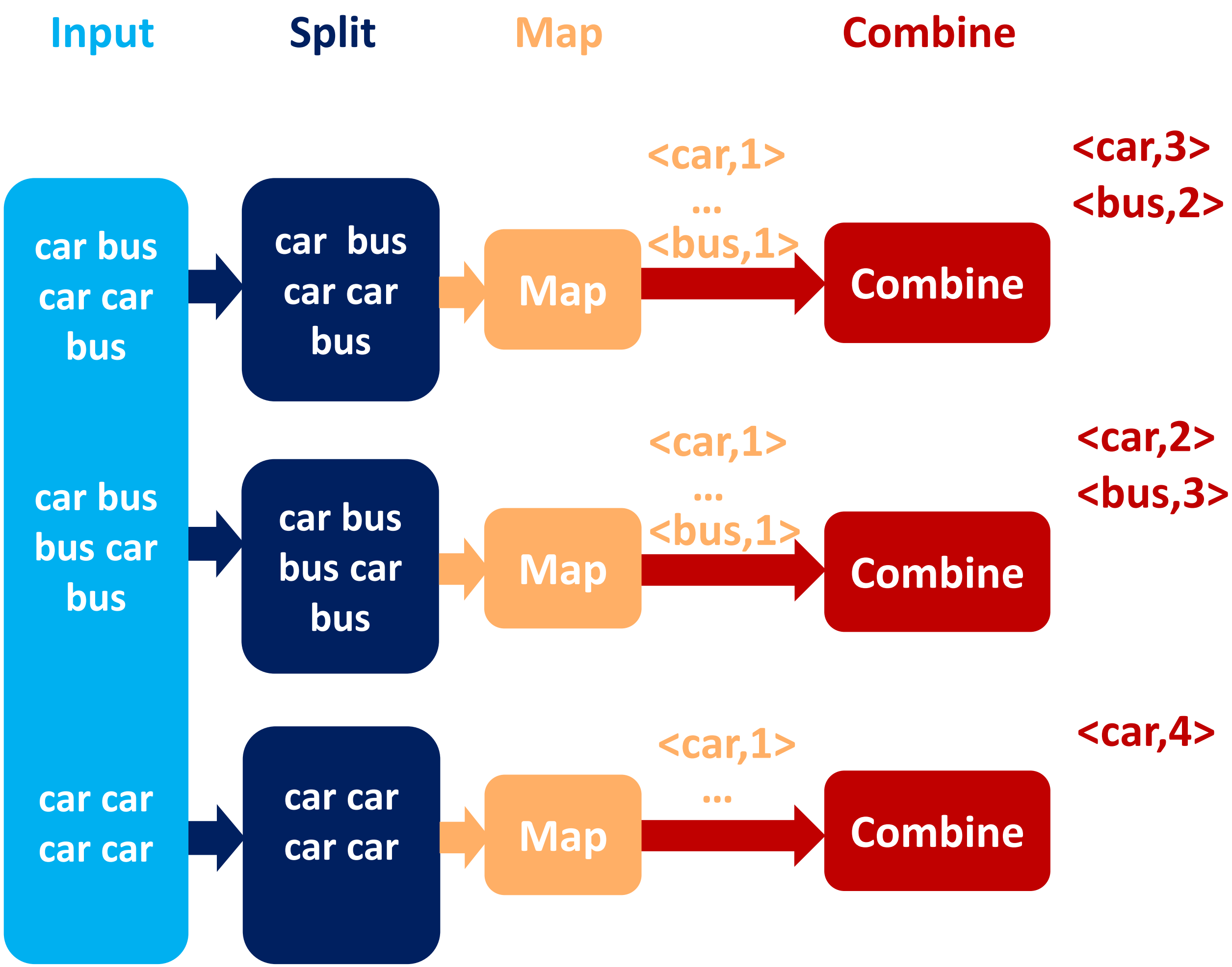
MapReduce – WordCount example

Input		Split
car bus car car bus	→	car bus car car bus
car bus bus car bus	→	car bus bus car bus
car car car car	→	car car car car

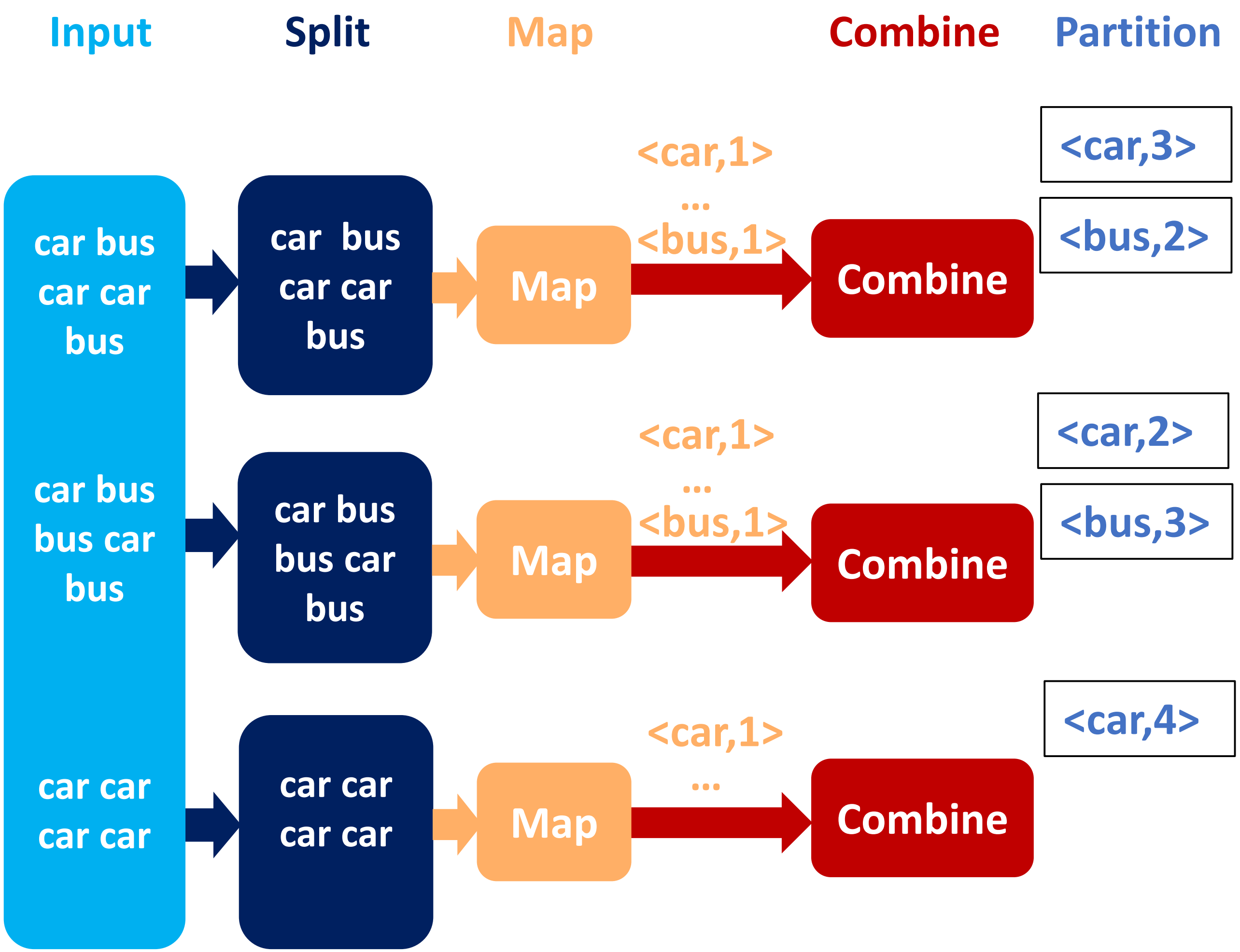
MapReduce – WordCount example



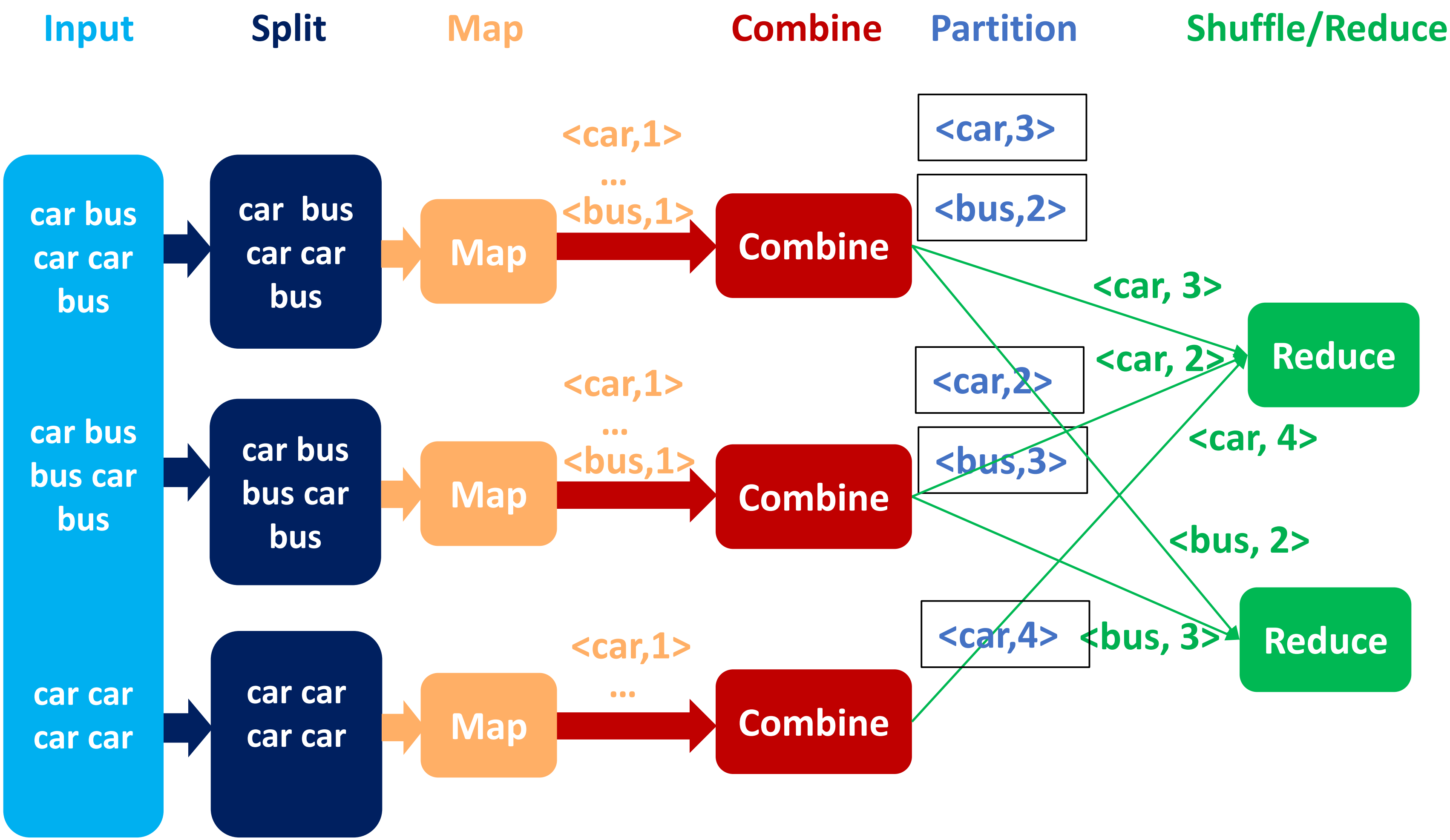
MapReduce – WordCount example



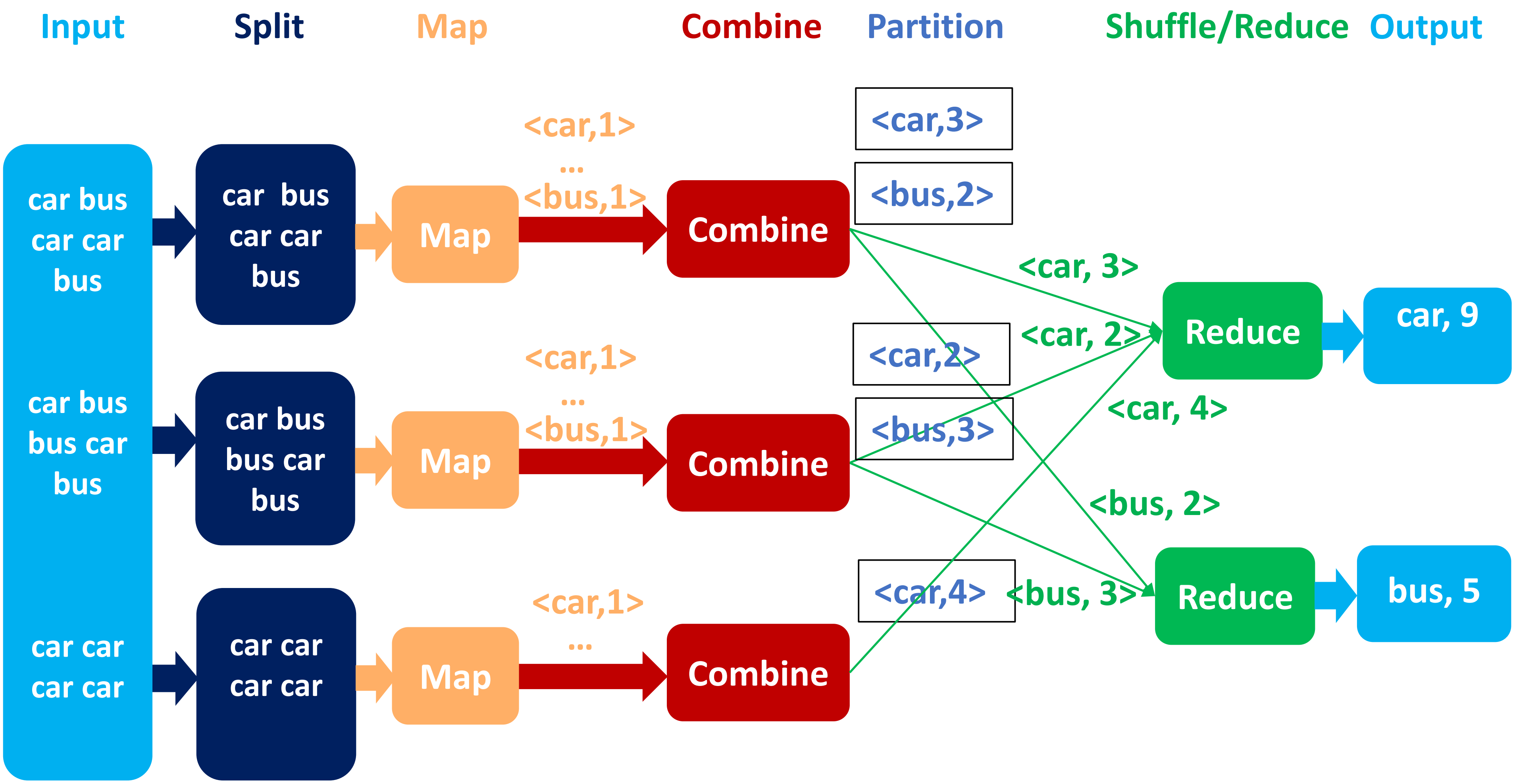
MapReduce – WordCount example



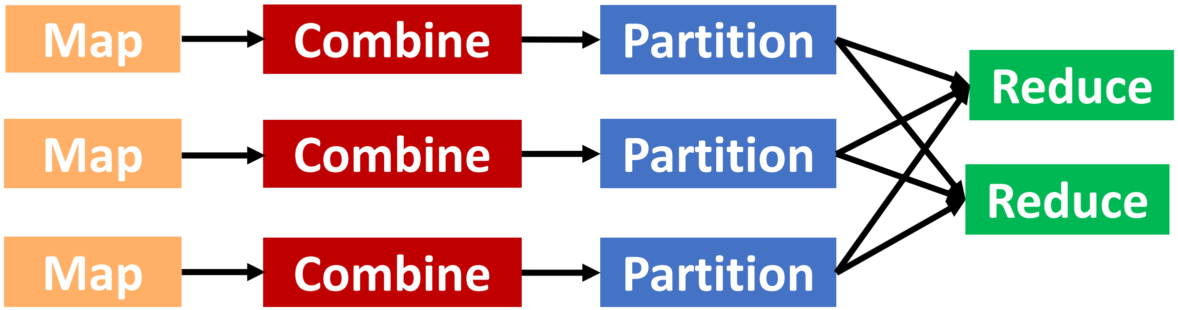
MapReduce – WordCount example

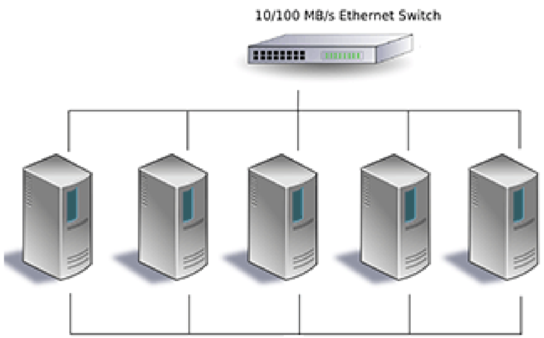


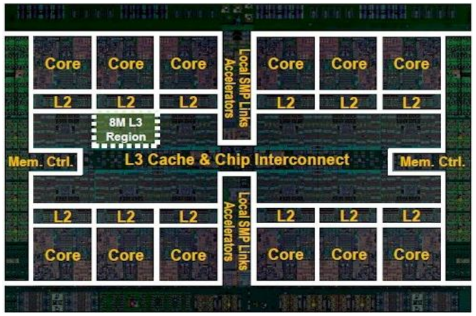
MapReduce – WordCount example



MapReduce implementations





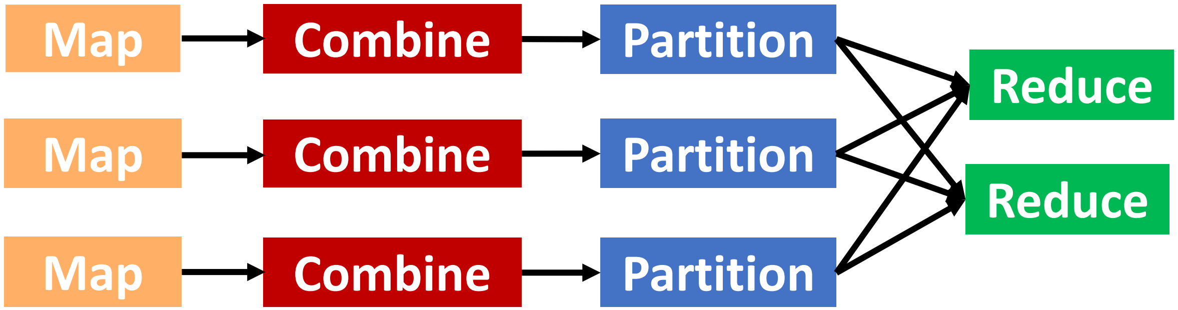


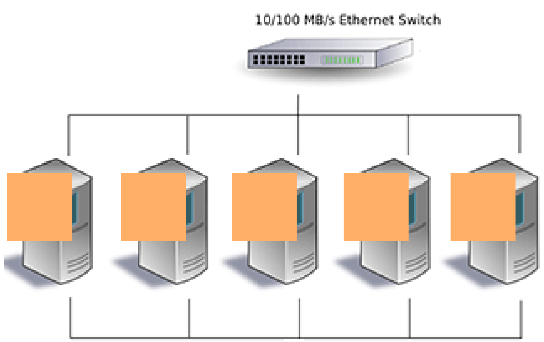
Scale-out:
Hadoop @ Yahoo

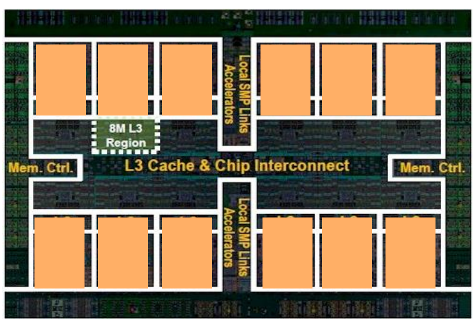
GPU:
Mars @ HKUST

CMP/Scale-up:
Phoenix++ @ Stanford

MapReduce implementations





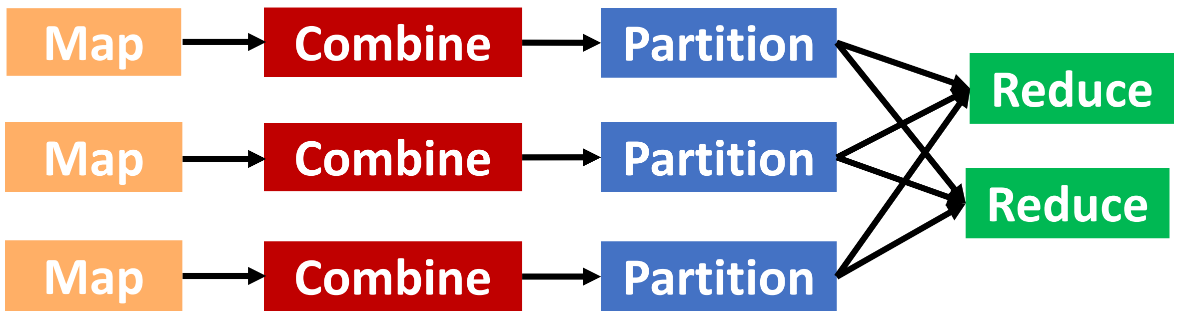


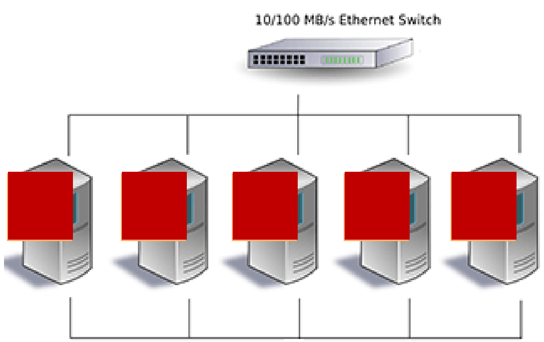
Scale-out:
Hadoop @ Yahoo

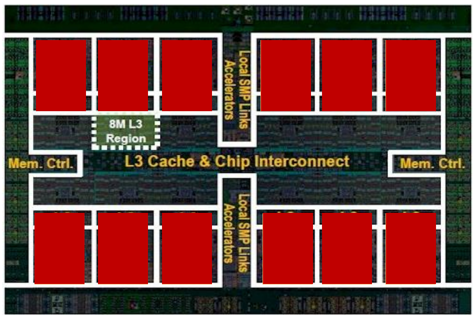
GPU:
Mars @ HKUST

CMP/Scale-up:
Phoenix++ @ Stanford

MapReduce implementations





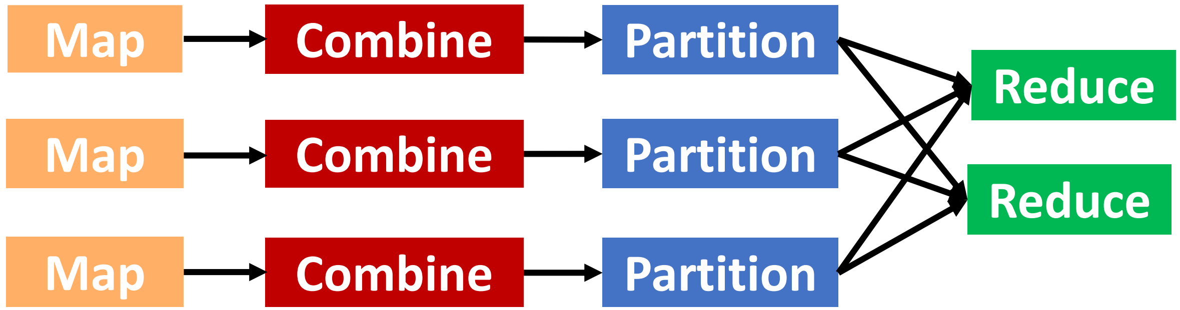


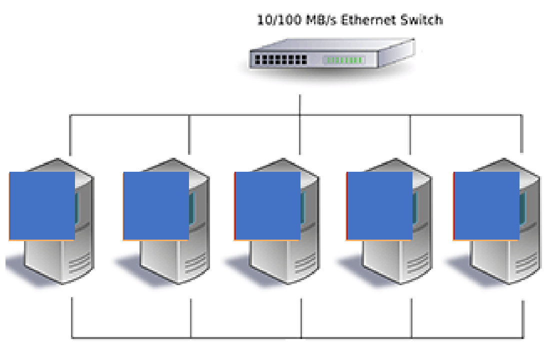
Scale-out:
Hadoop @ Yahoo

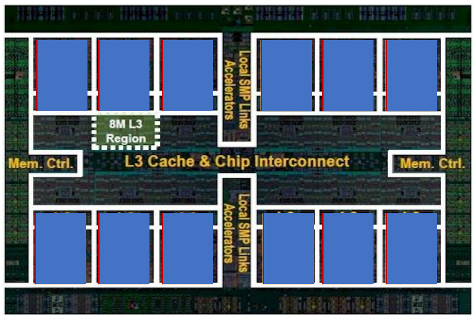
GPU:
Mars @ HKUST

CMP/Scale-up:
Phoenix++ @ Stanford

MapReduce implementations





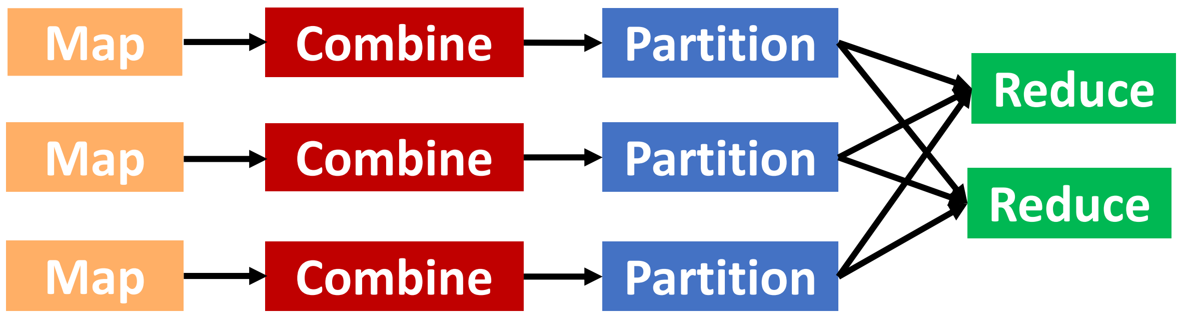


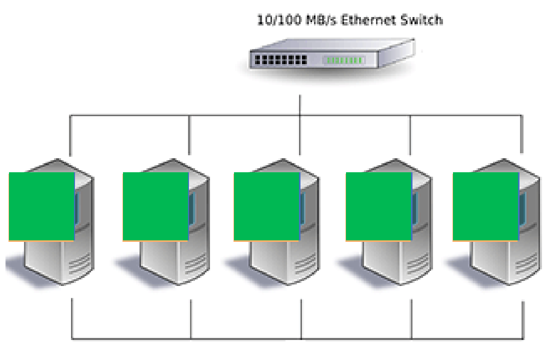
Scale-out:
Hadoop @ Yahoo

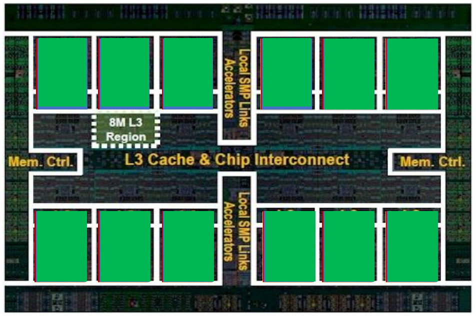
GPU:
Mars @ HKUST

CMP/Scale-up:
Phoenix++ @ Stanford

MapReduce implementations





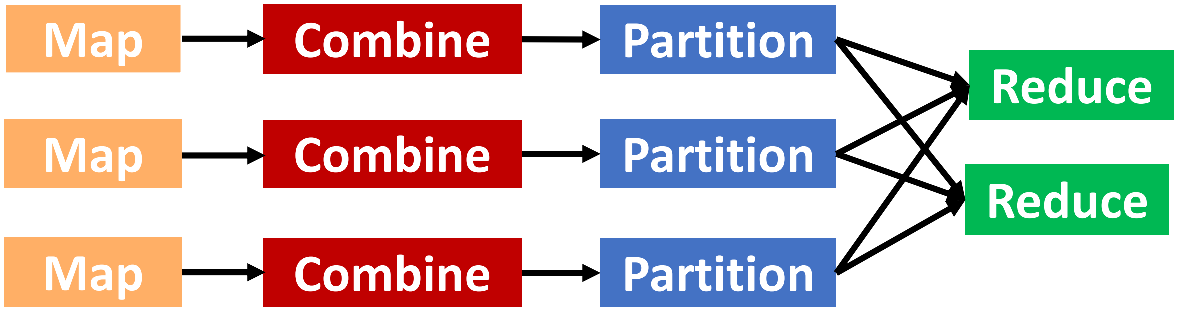


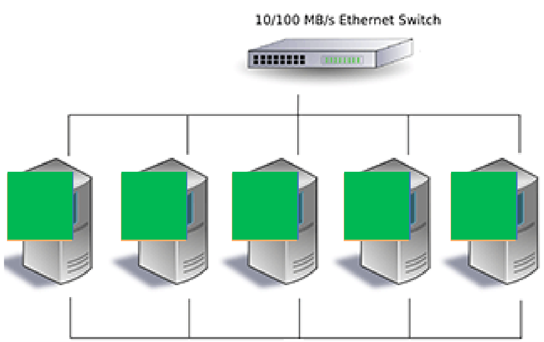
Scale-out:
Hadoop @ Yahoo

GPU:
Mars @ HKUST

CMP/Scale-up:
Phoenix++ @ Stanford

MapReduce implementations

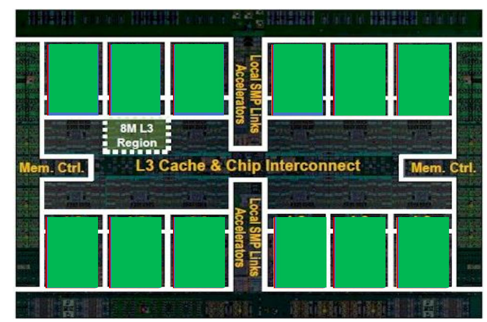




Scale-out:
Hadoop @ Yahoo

GPU:
Mars @ HKUST

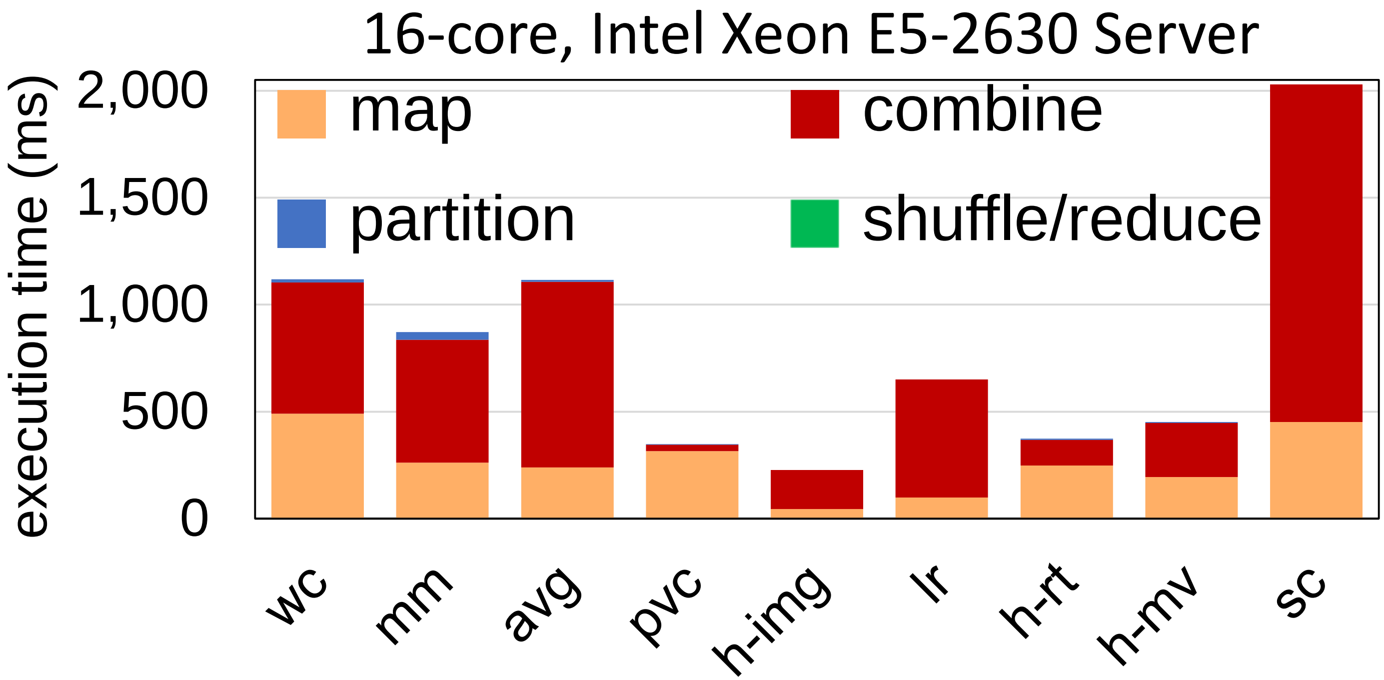
focus of this work



CMP/Scale-up:
Phoenix++ @ Stanford

Phoenix++ studies

- Execution breakdown



Workload description

wc: word count

mm: min-max

avg: average

pvc: page view count

h-img: histogram image

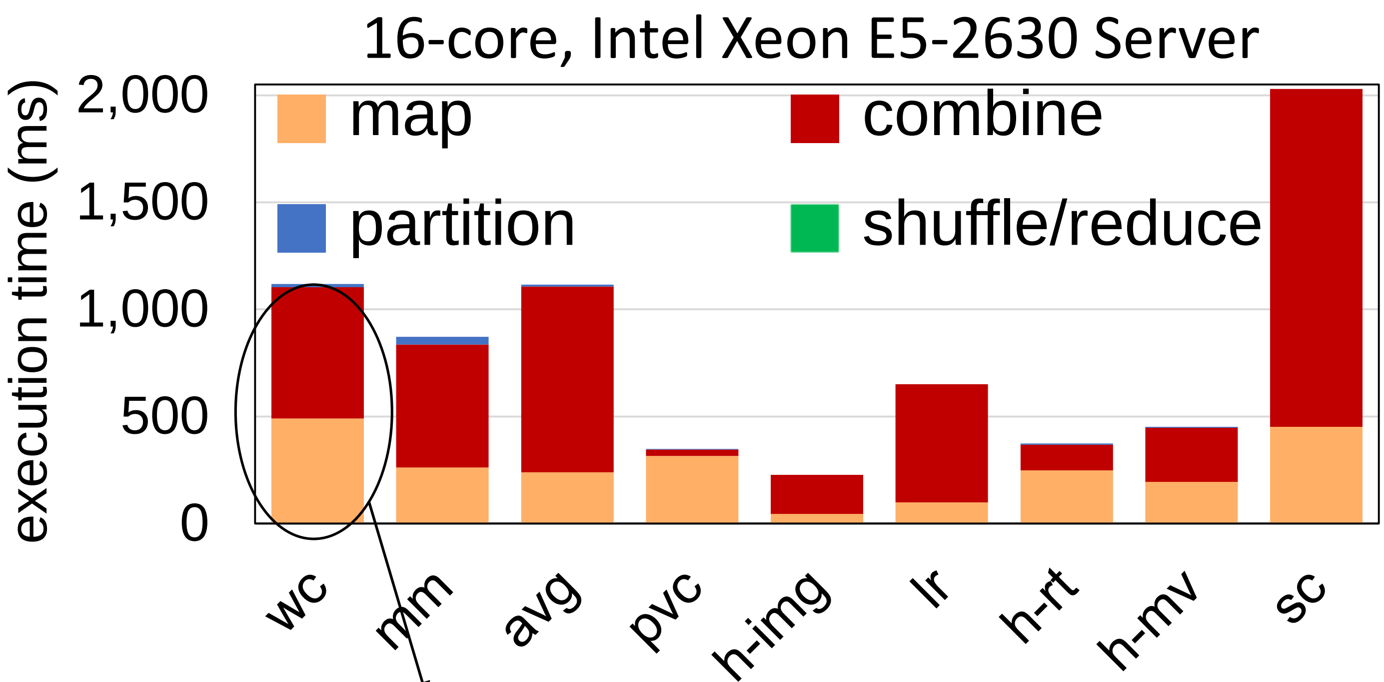
h-rt: histogram user

h-mv: histogram movie

sc: sequence count

Phoenix++ studies

- Execution breakdown



map & combine dominate

Workload description

wc: word count

mm: min-max

avg: average

pvc: page view count

h-img: histogram image

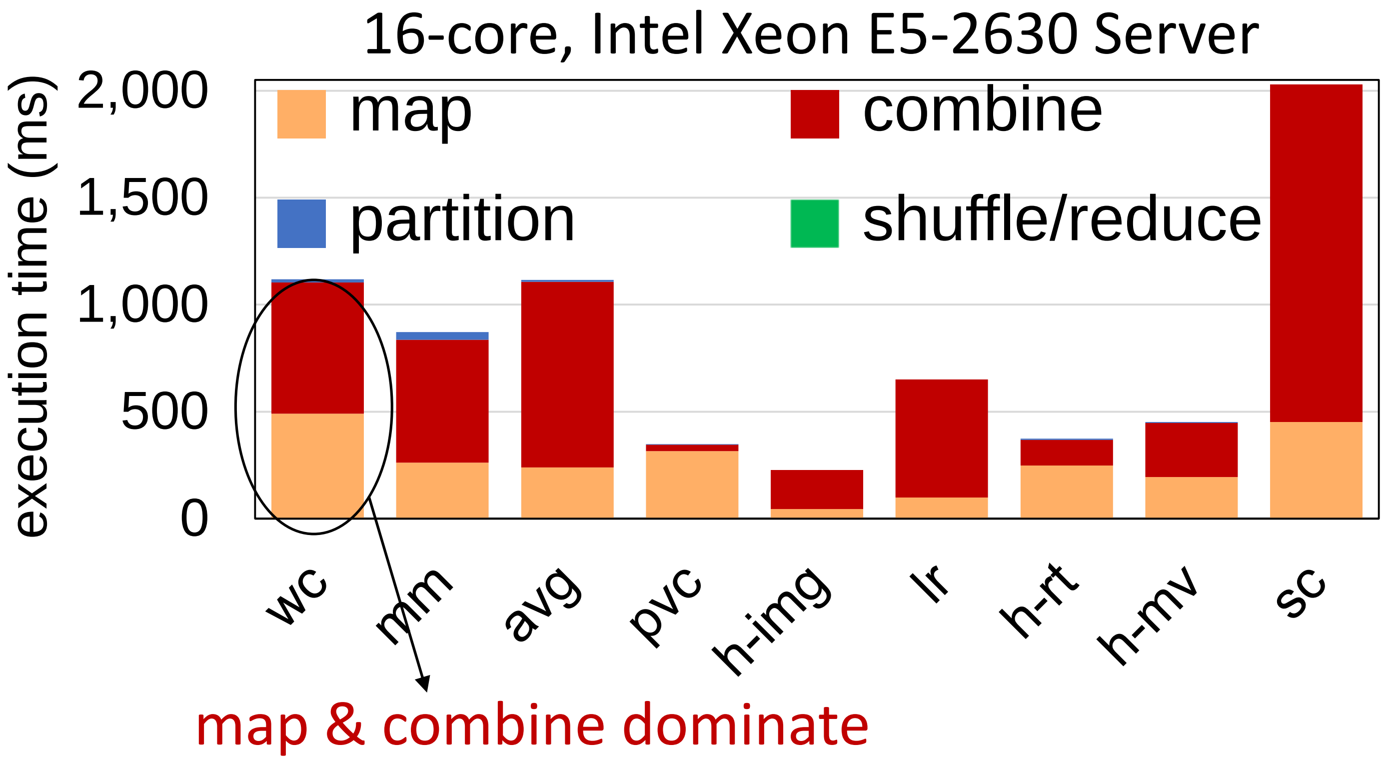
h-rt: histogram user

h-mv: histogram movie

sc: sequence count

Phoenix++ studies

- Execution breakdown



<u>Workload description</u>

wc: word count

mm: min-max

avg: average

pvc: page view count

h-img: histogram image

h-rt: histogram user

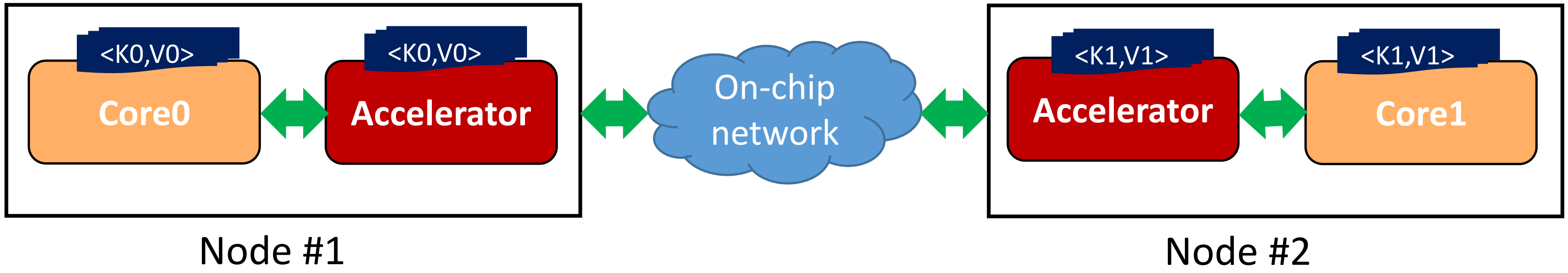
h-mv: histogram movie

sc: sequence count

- Inefficiencies:
  - Serial execution of map and combine phases
  - Inefficient key-value lookup during combine phase

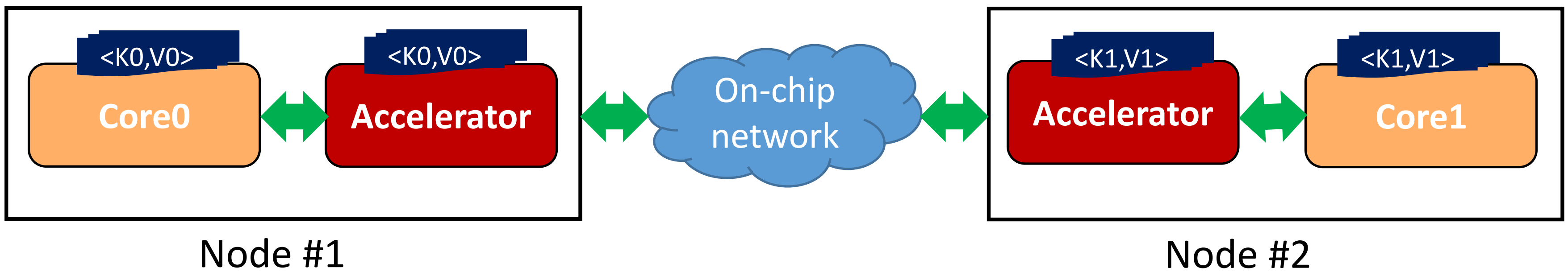
CASM overview

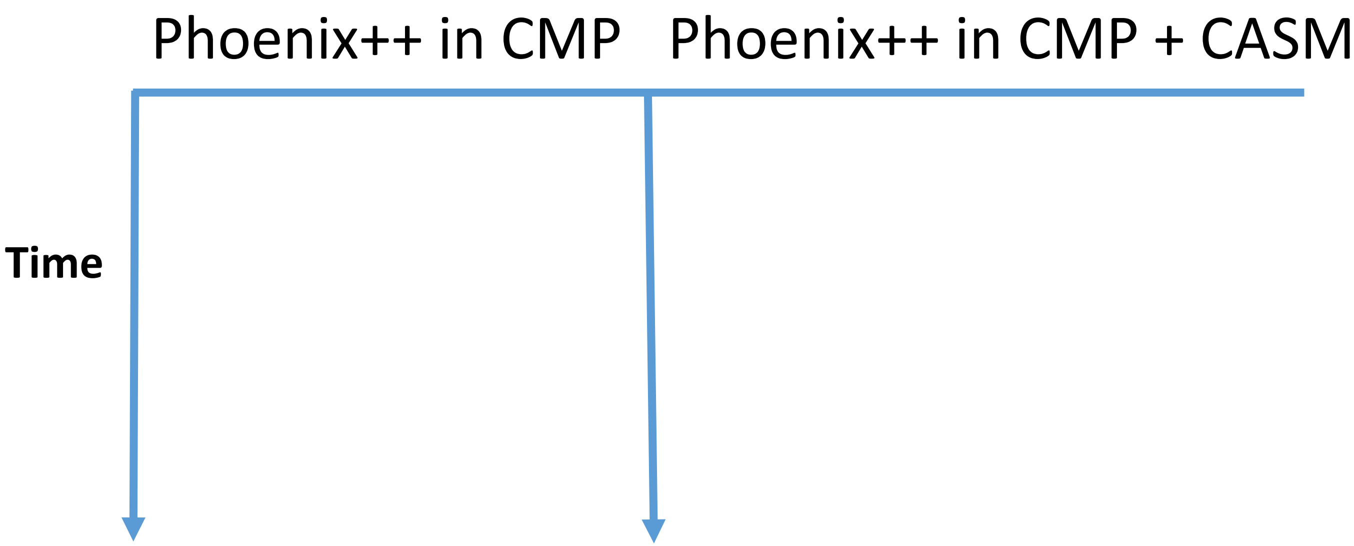
- Execution flow



CASM overview

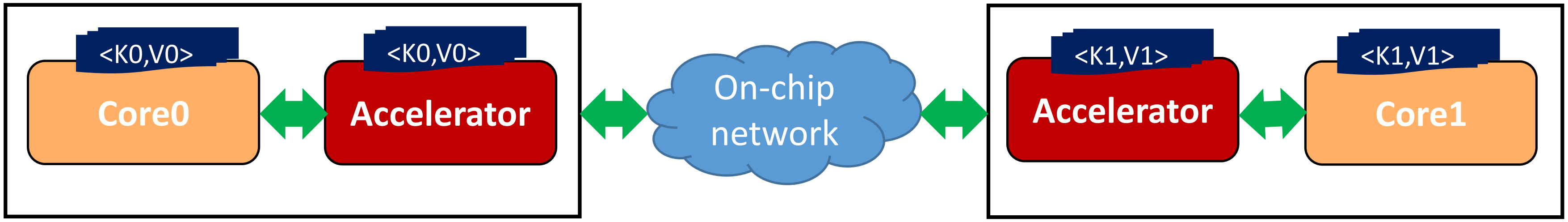
- Execution flow





CASM overview

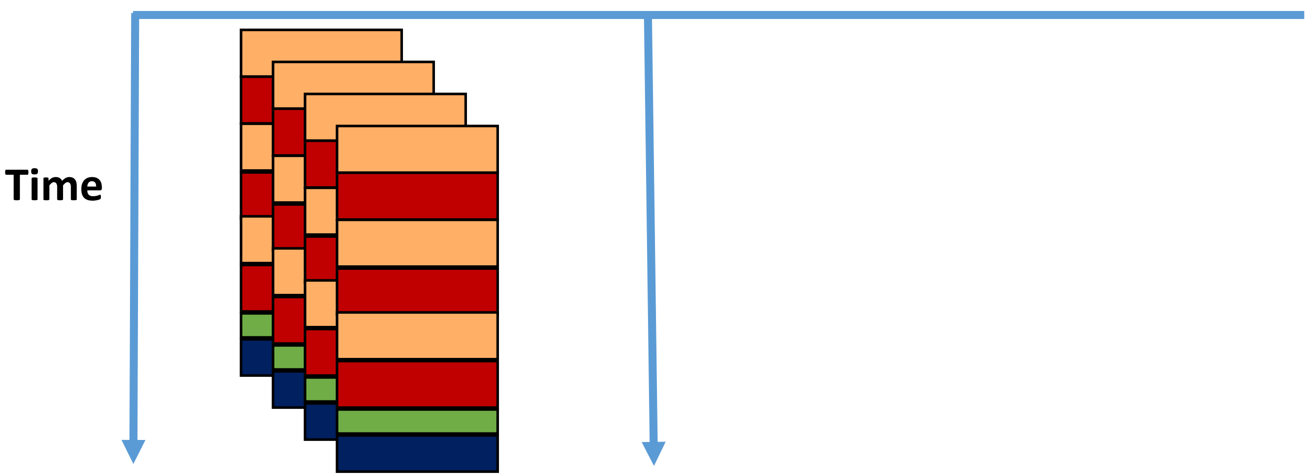
- Execution flow



Node #1

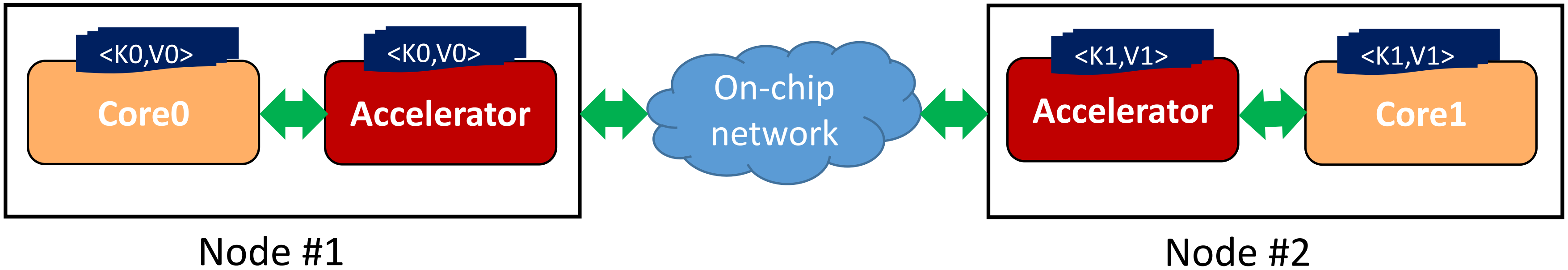
Node #2

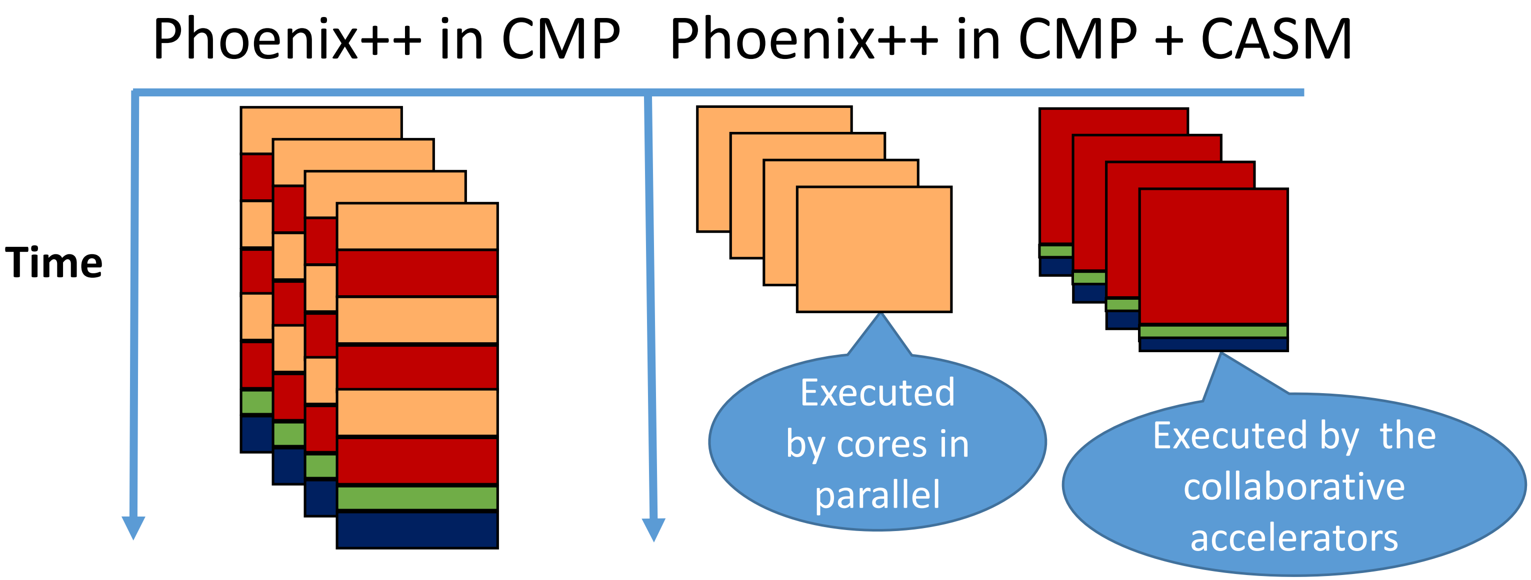
Phoenix++ in CMP   Phoenix++ in CMP + CASM



CASM overview

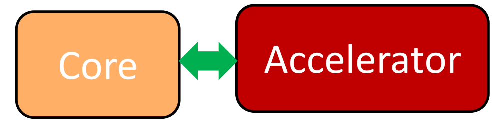
- Execution flow



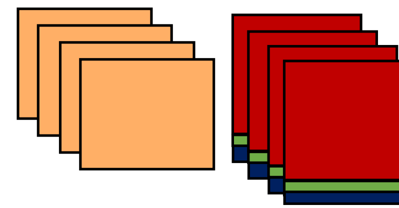


Key contributions

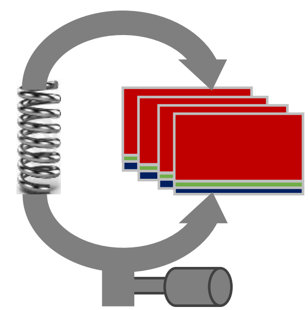
- Scalable and collaborative accelerators



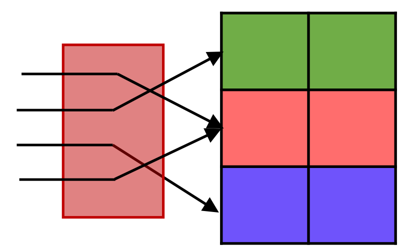
- Parallel execution of map and combine phases



- Faster execution of combine phase by the accelerators

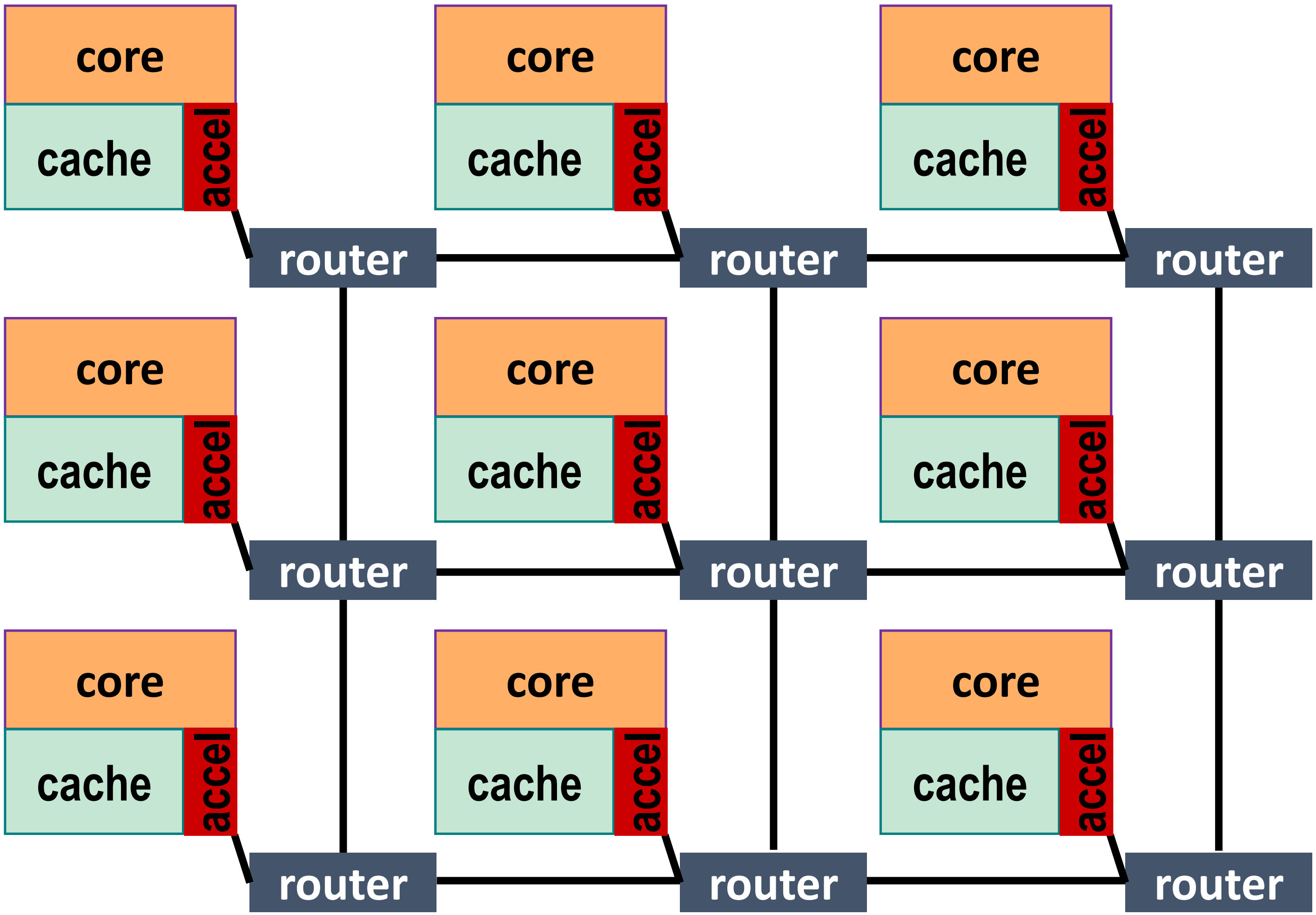


- In-hardware hash function



CASM high-level architecture

- Each accelerator has two main storage structures



CASM high-level architecture

- Each accelerator has two main storage structures

local scratchpad

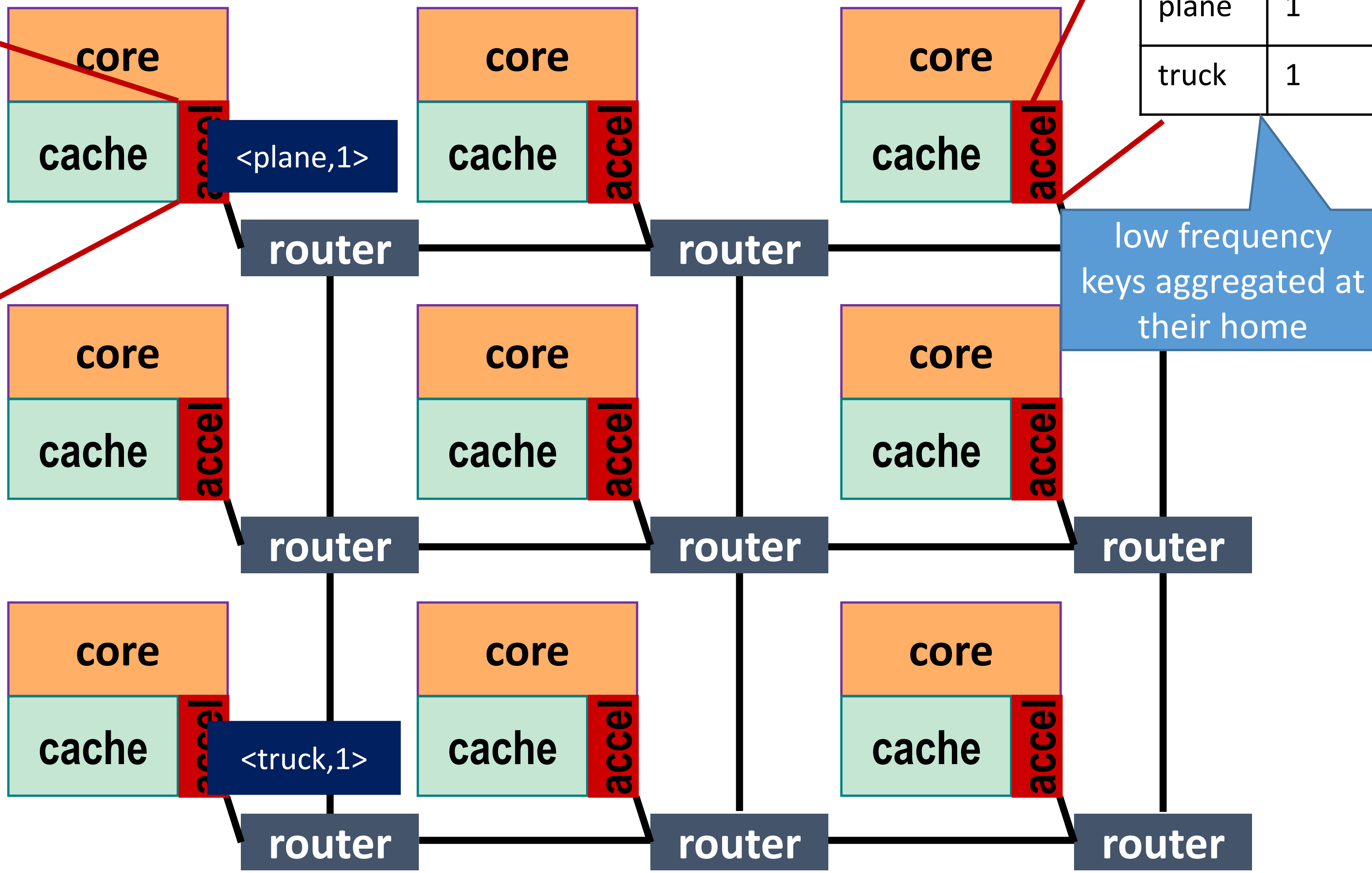
key	value
car	3
bus	2
train	2

high frequency keys aggregated locally

home scratchpad

key	value
plane	1
truck	1

low frequency keys aggregated at their home



Storage design space exploration

- **Option one**: local-only
  key-value pairs replicated across multiple accelerators

accel0

key	value
car	3
bus	2
train	2

accel1

key	value
car	3
bus	2
train	2

accel2

key	value
car	3
bus	2
train	2

- **Option two**: home-only
  a key-value pair exists only in one location

accel0

key	value
car	3
truck	1
plane	1

accel1

key	value
train	2
ship	1
bicycle	2

accel2

key	value
bus	2
motor	1
rocket	1

Storage design space exploration

- **Option one**: local-only
  key-value pairs replicated across multiple accelerators

accel0

key	value
car	3
bus	2
train	2

accel1

key	value
car	3
bus	2
train	2

accel2

key	value
car	3
bus	2
train	2

- **Option two**: home-only
  a key-value pair exists only in one location

key	value
car	3
truck	1
plane	1
train	2
ship	1
bicycle	2
bus	2
motor	1
rocket	1

unified large memory with no replication

Storage design space exploration

- **Option one**: local-only
  key-value pairs replicated across multiple accelerators

- **Option two**: home-only
  a key-value pair exists only in one location

Option one (local-only):

accel0

key	value
car	3
bus	2
train	2

accel1

key	value
car	3
bus	2
train	2

accel2

key	value
car	3
bus	2
train	2

Option two (home-only):

accel0

key	value
car	3
truck	1
plane	1

accel1

key	value
train	2
ship	1
bicycle	2

accel2

key	value
bus	2
motor	1
rocket	1

storage options	network traffic	memory traffic
local-only	low	high
home-only	high	low
local + home	low	low

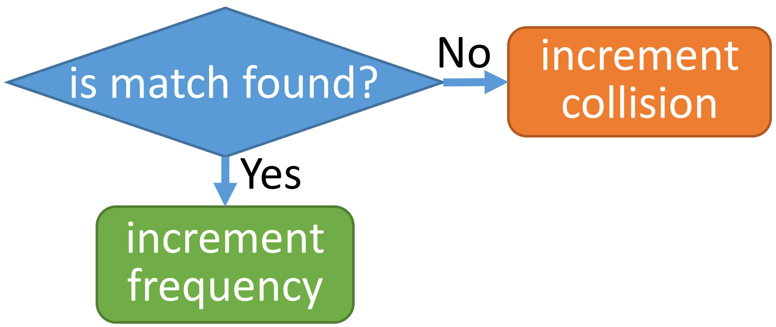
Key-value pair eviction policy

- "frequency" and "collision" bits are stored in scratchpads

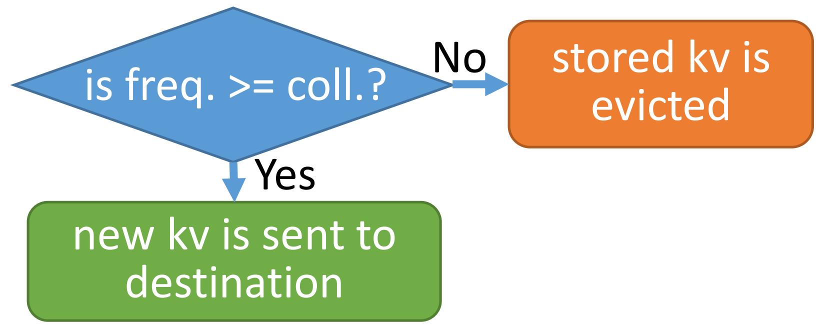
  Scratchpad structure:

key	value	**frequency**	**collision**

  - Frequency and collision update units:



  - Simple heuristic function to identify frequently occurring keys:



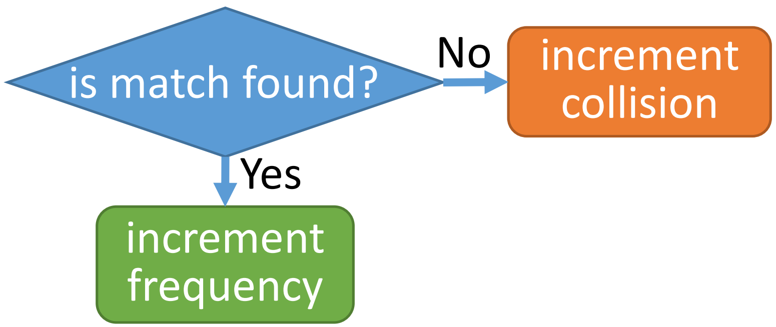
Key-value pair eviction policy

- “frequency” and “collision” bits are stored in scratchpads

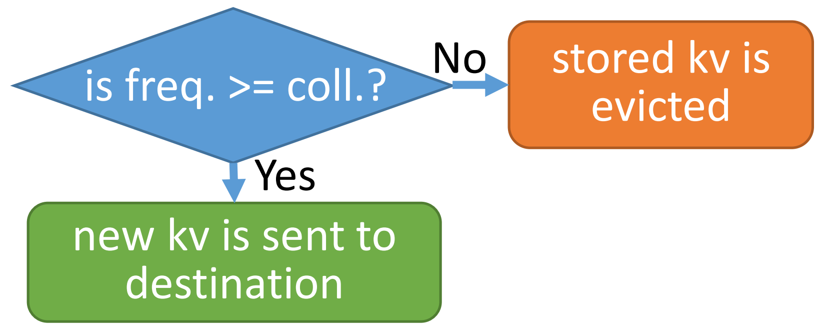
Scratchpad structure:

key	value	frequency	collision

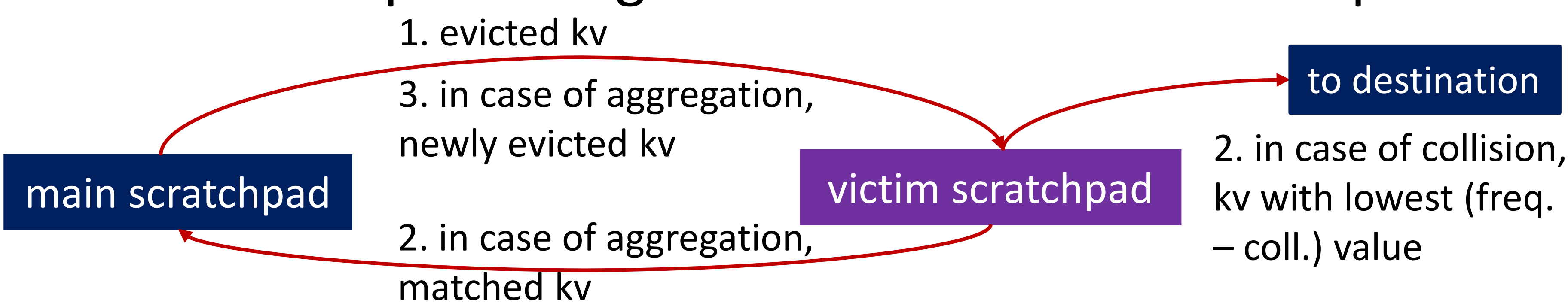
- ➢ Frequency and collision update units:



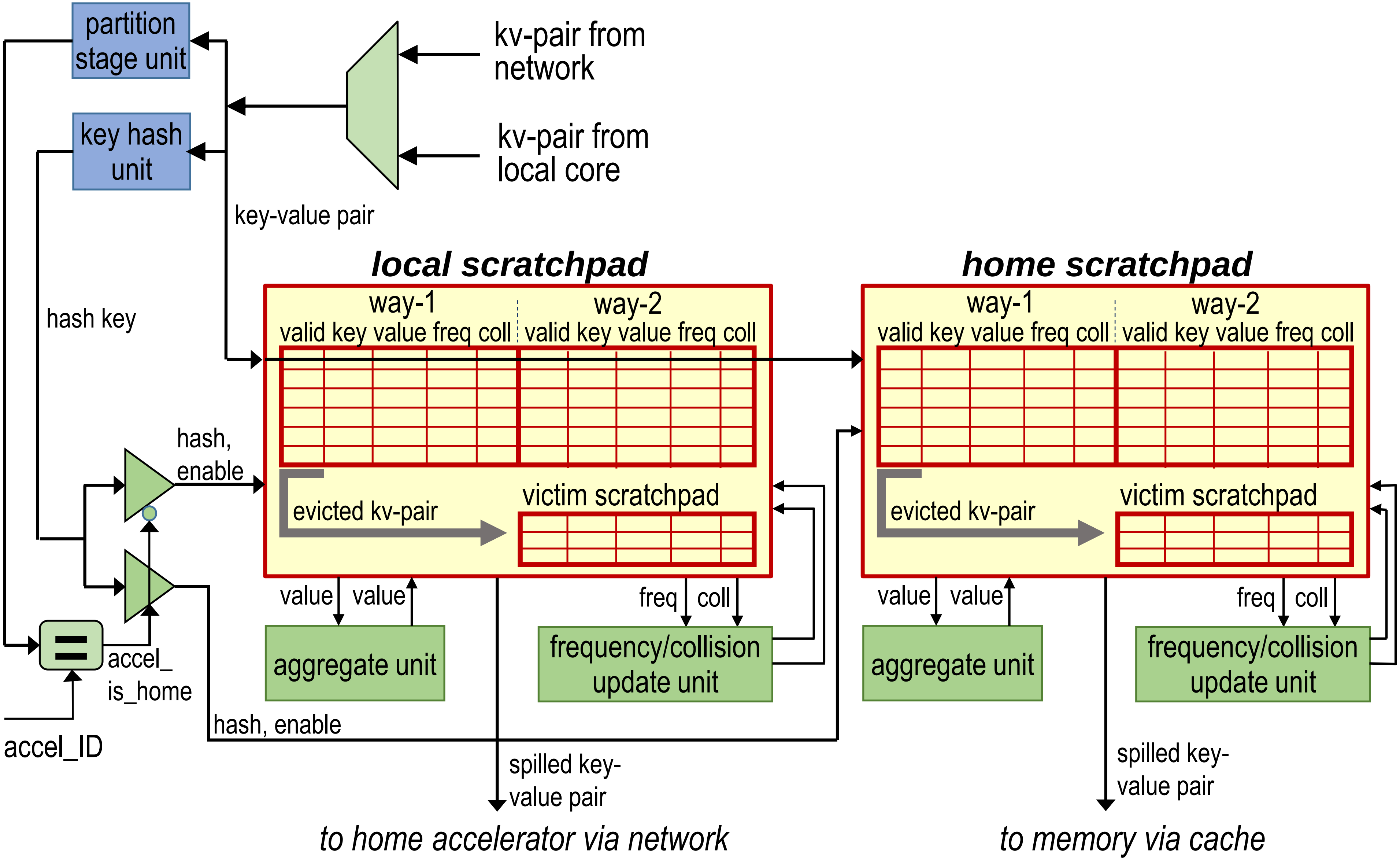
- ➢ Simple heuristic function to identify frequently occurring keys:



- Each scratchpad is augmented with victim scratchpad



Accelerator architecture



Experimental framework

- Scale-up CMP configuration (Gem5/Garnet)

Parameter	Value
Core	64 cores, OoO, 8-wide
L1 D&I caches	16KB
L2 cache (shared)	128KB per core/slice
Coherence	MOESI directory-based
Memory	4xDDR3-1600, 12GB/s

- CASM configuration (Gem5/Garnet)

Parameter	Value
Scratchpad size	16KB
#entries per victim scratchpad	8
Max. key size	64 bits
Max. value size	64 bits
Freq. & coll. size	8 bits

Experimental framework

- Scale-up CMP configuration (Gem5/Garnet)

Parameter	Value
Core	64 cores, OoO, 8-wide
L1 D&I caches	16KB
L2 cache (shared)	128KB per core/slice
Coherence	MOESI directory-based
Memory	4xDDR3-1600, 12GB/s

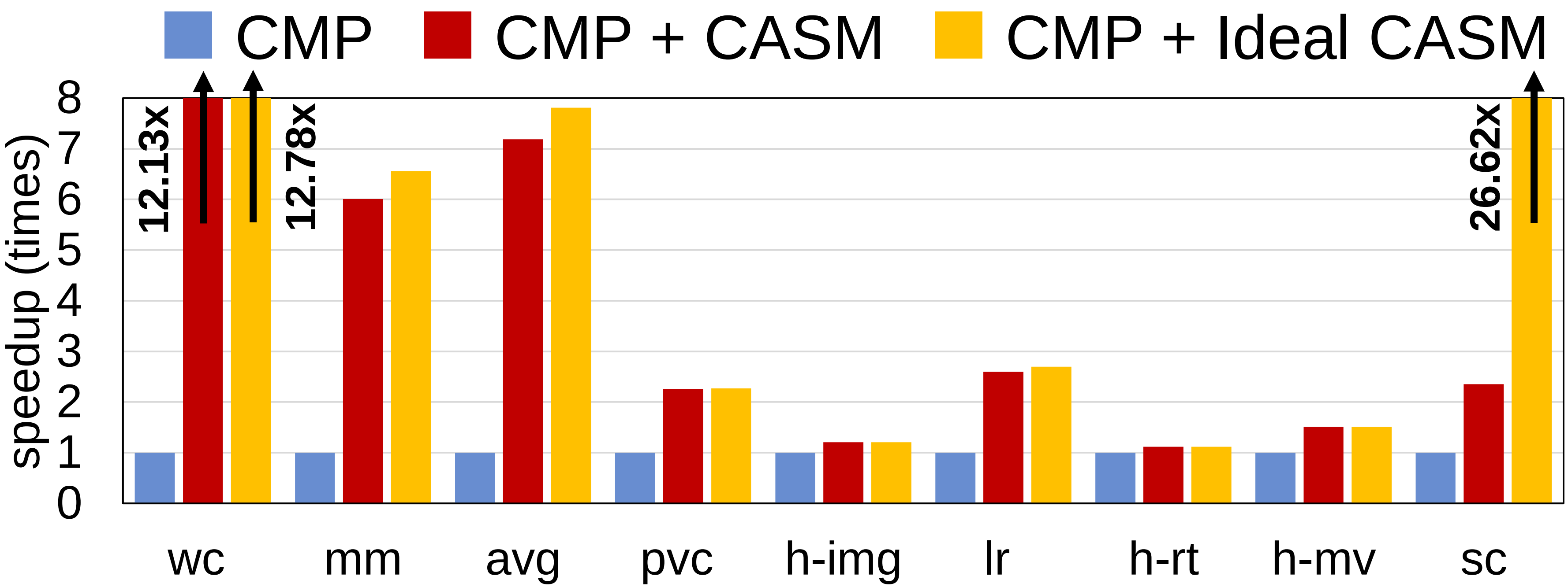
- CASM configuration (Gem5/Garnet)

Parameter	Value
Scratchpad size	16KB
#entries per victim scratchpad	8
Max. key size	64 bits
Max. value size	64 bits
Freq. & coll. size	8 bits

- Workload characteristics

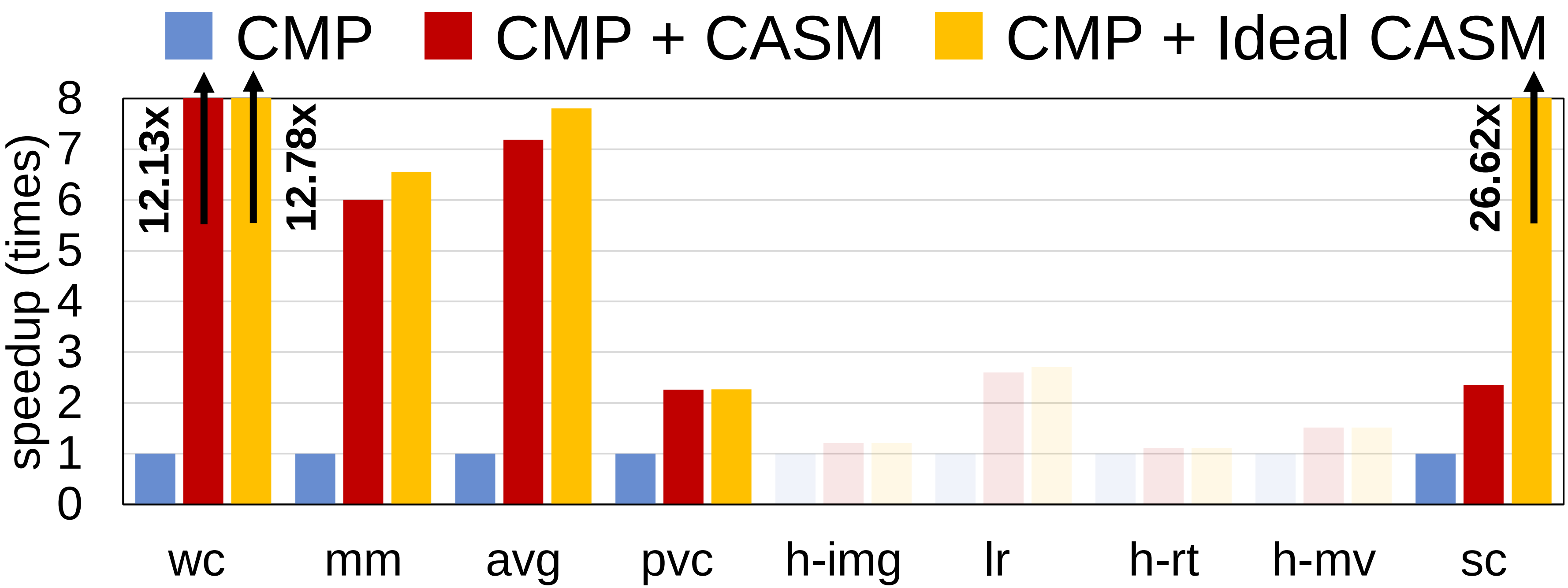
workload	wc	mm	avg	pvc	h-img	lr	h-rt	h-mv	sc
#unique keys	257K	28K	28K	10K	768	5	5	20K	3.5M
cache locality	low	low	low	low	high	high	high	high	low

Performance and energy analysis



- > 4x speedup on average
- > 3.5x energy saving on average

Performance and energy analysis

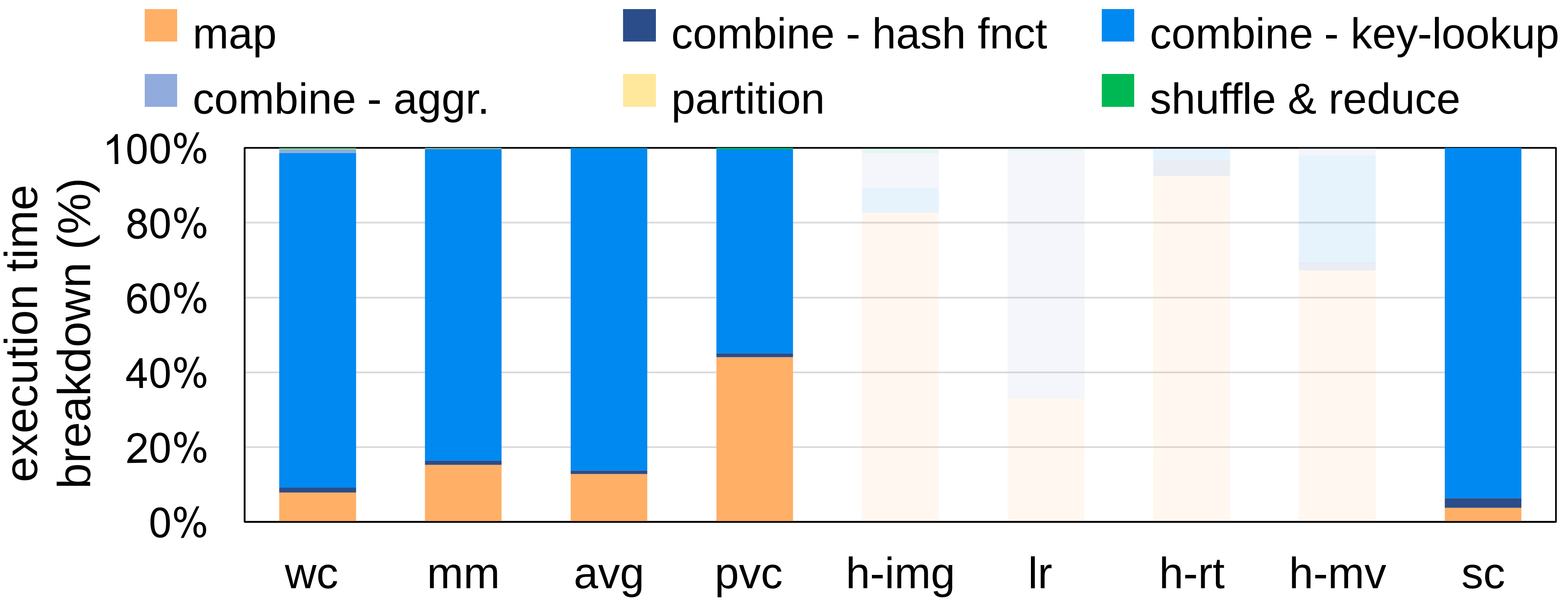


- > 4x speedup on average
- > 3.5x energy saving on average

Large #unique keys & no cache locality ➡ high speedup

Sources of performance benefits

- Mainly due to offloading the combine phase to CASM

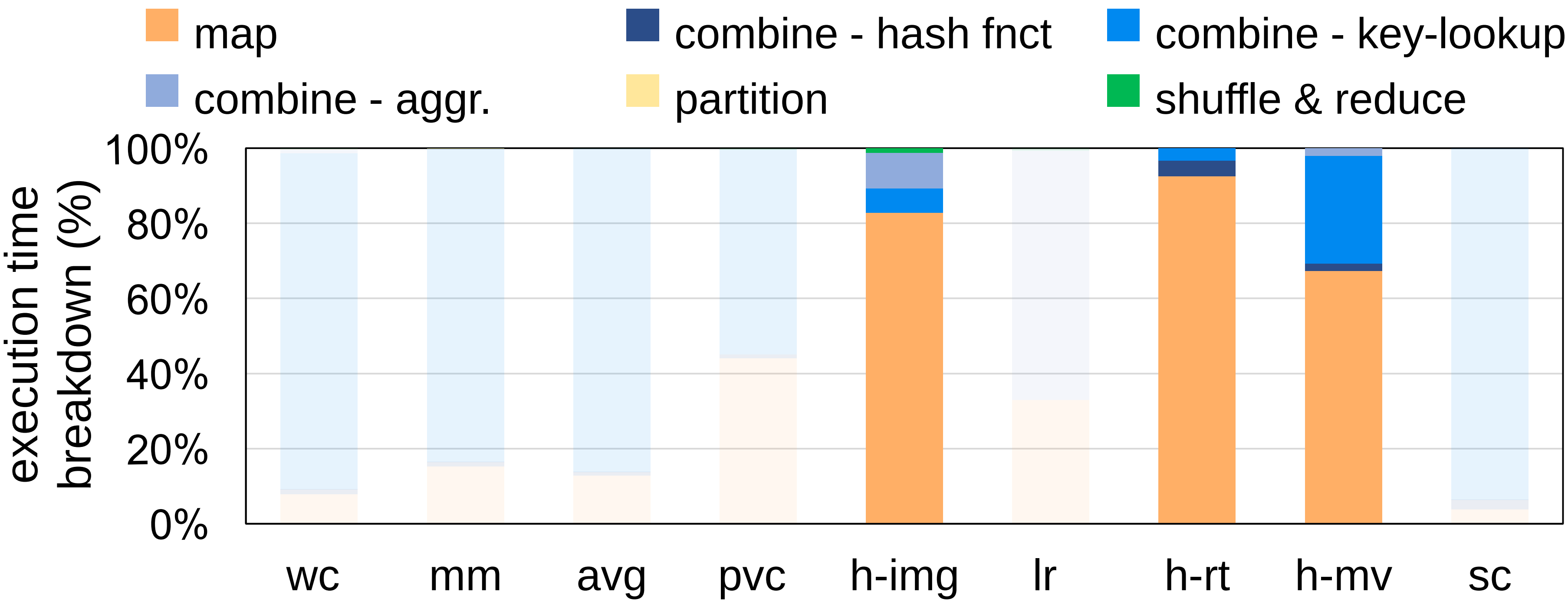


Dominant phase:

combine – key-lookup

Sources of performance benefits

- Mainly due to offloading the combine phase to CASM

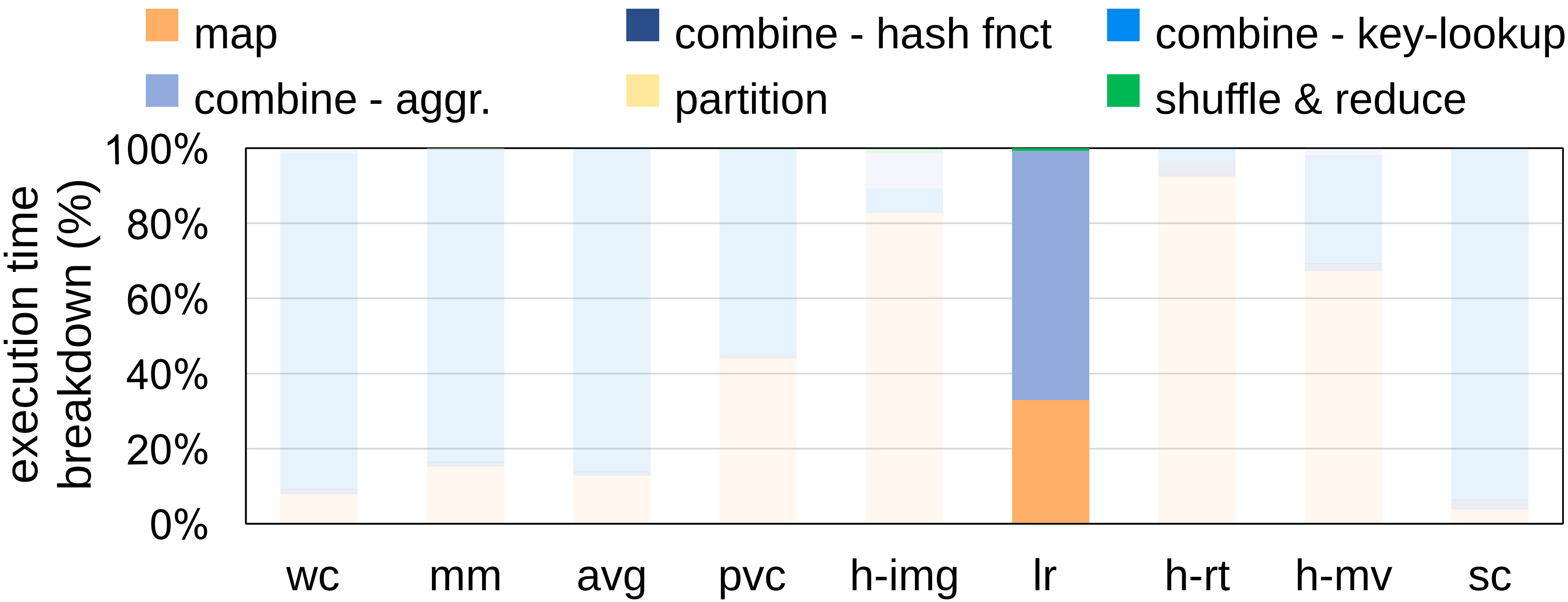


Dominant phase:

map

Sources of performance benefits

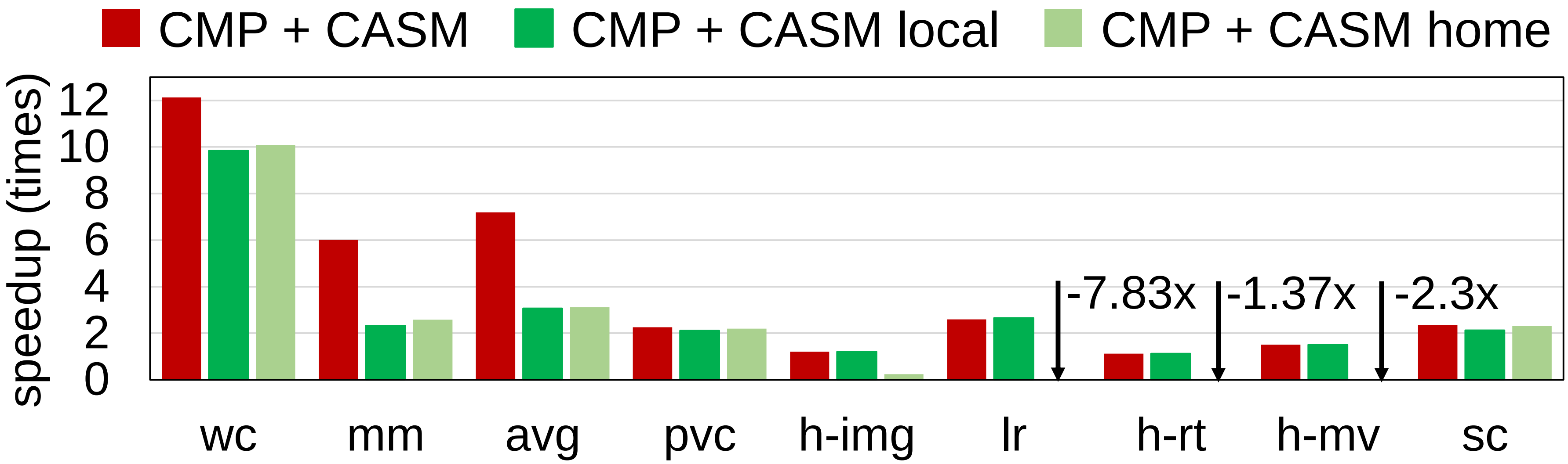
- Mainly due to offloading the combine phase to CASM



Dominant phase:

combine – aggregation

Speedup contribution: local vs home



- A hybrid of local and home accelerators provides significant benefits across applications

Conclusion

- MapReduce on scale-up machines suffers from:
  - serial execution of map and combine phases
  - inefficient key-value lookup
- Solution:
  - Parallel execution of map and combine phases
  - Local/home partitioned on-chip storage
  - Aggregation near on-chip storage
- CASM provides:
  - >4x in performance on average
  - >3.5x in energy saving on average
  - < 6% of area overhead