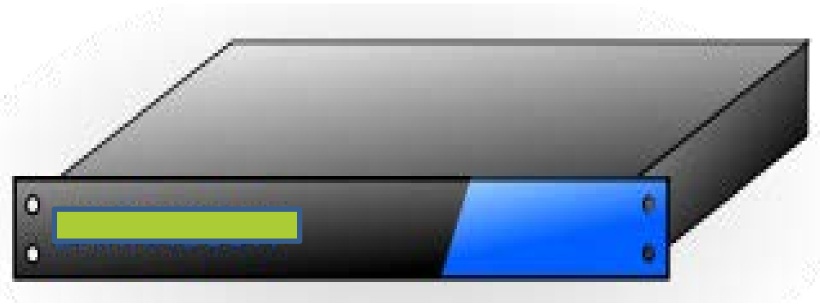
DVM: Towards a Datacenter-Scale Virtual Machine

Zhiqiang Ma[‡], Zhonghua Sheng[‡], Lin Gu[‡], Liufei Wen[†] and Gong Zhang[†]

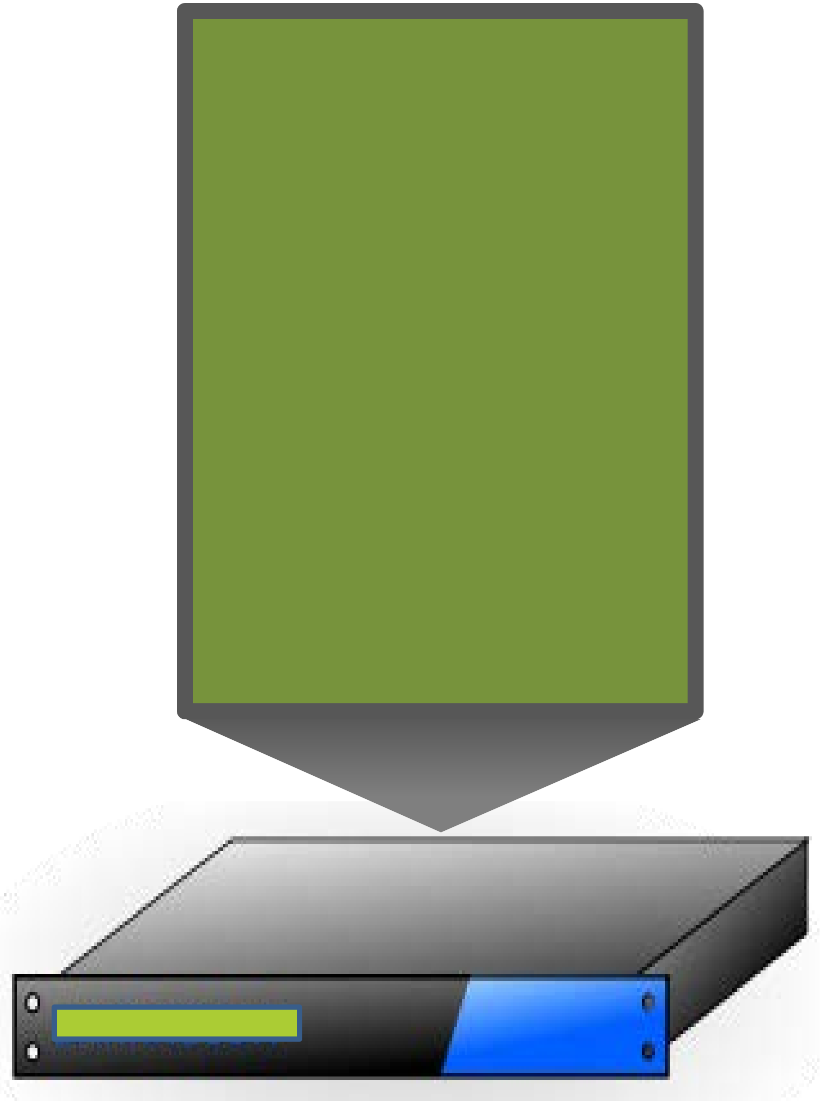
‡ Department of Computer Science and Engineering,
The Hong Kong University of Science and Technology, Hong Kong
† Huawei Technologies, Shenzhen, China

Eighth Annual International Conference on
Virtual Execution Environments (VEE 2012)
London, UK, March 3 - 4 2012

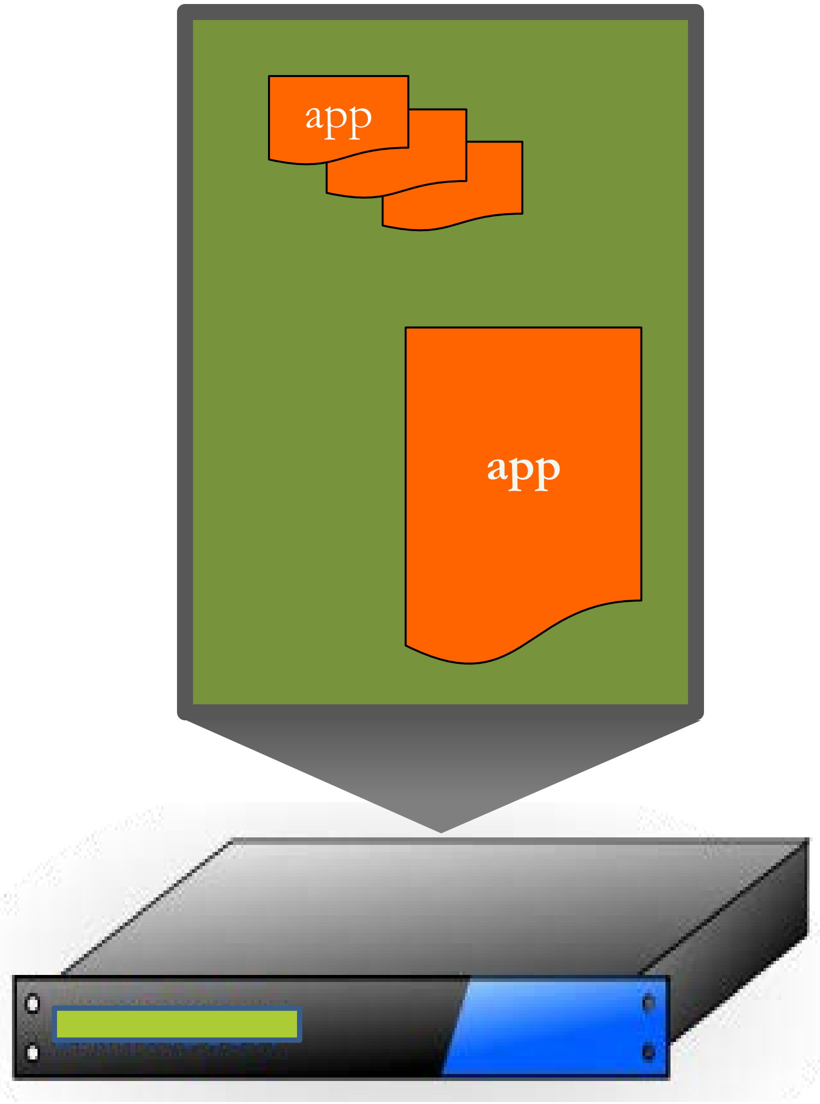
Virtualization technology



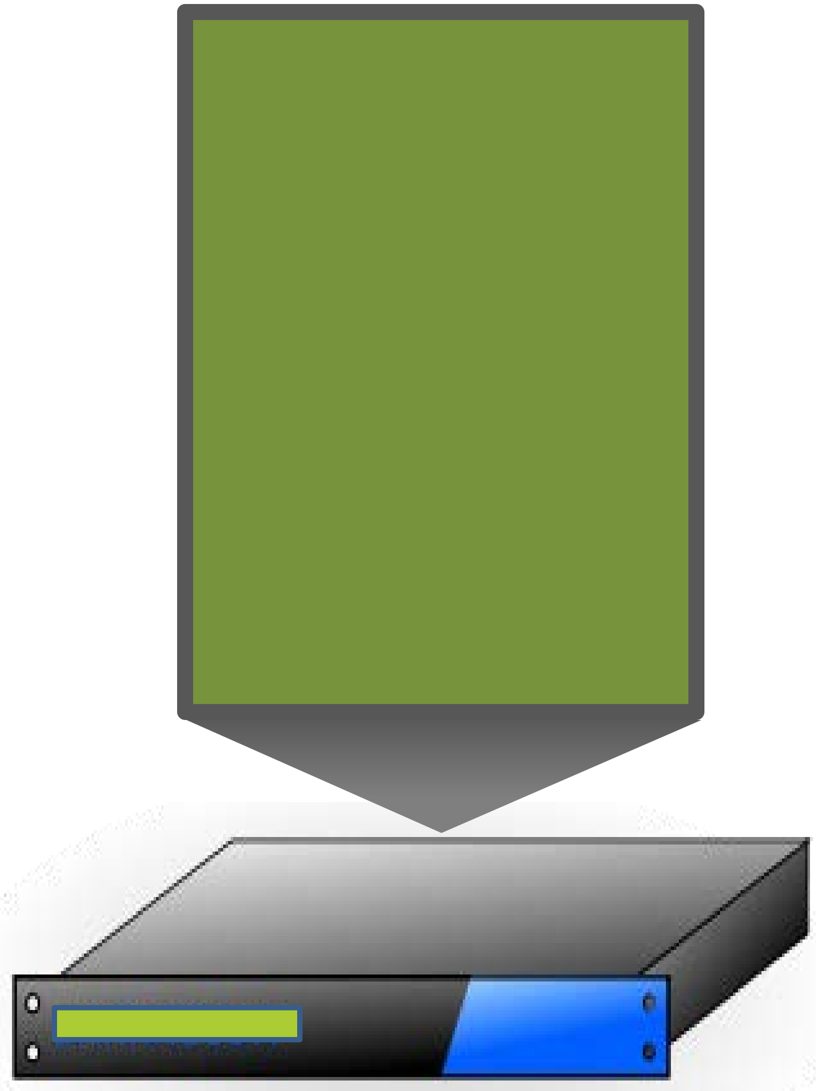
Virtualization technology



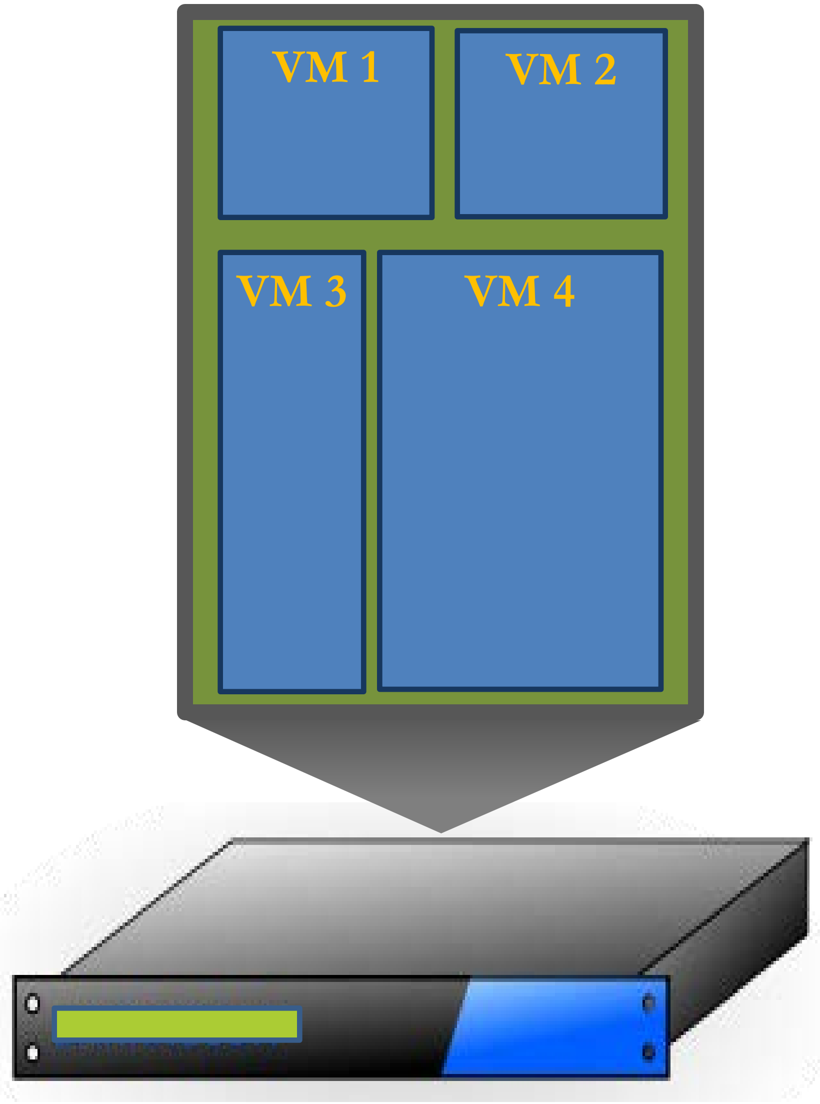
Virtualization technology



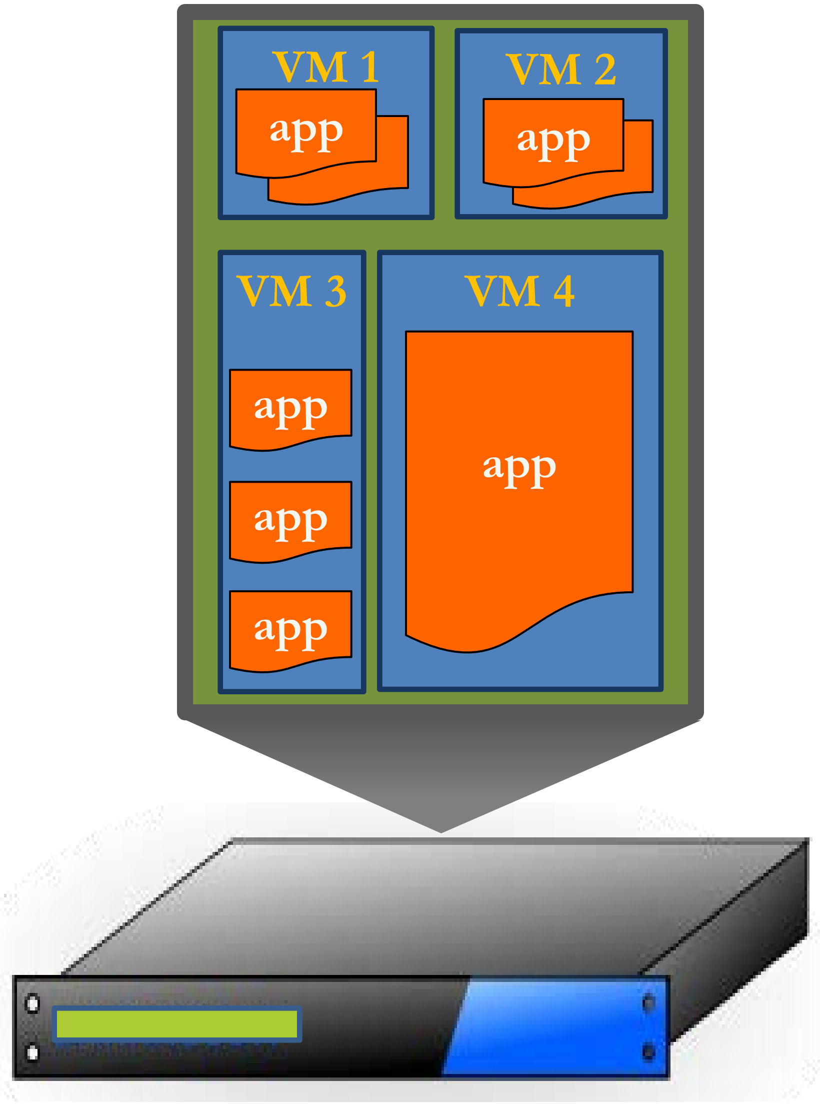
Virtualization technology



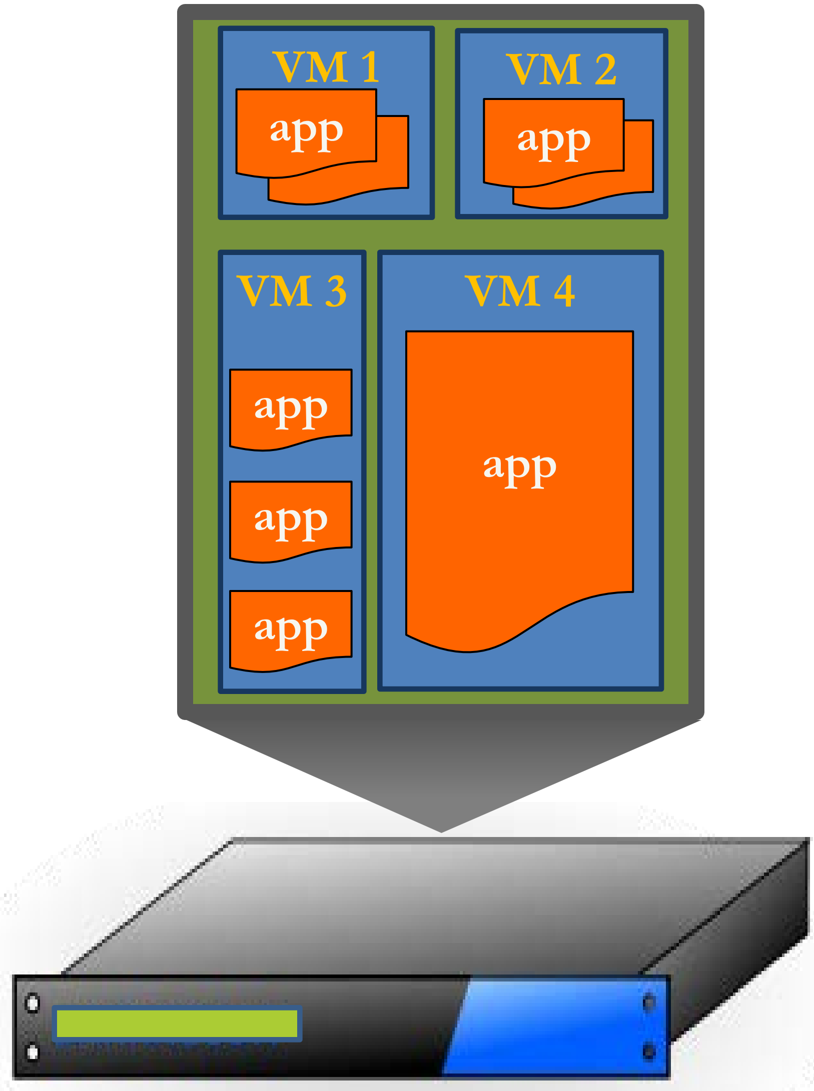
Virtualization technology



Virtualization technology

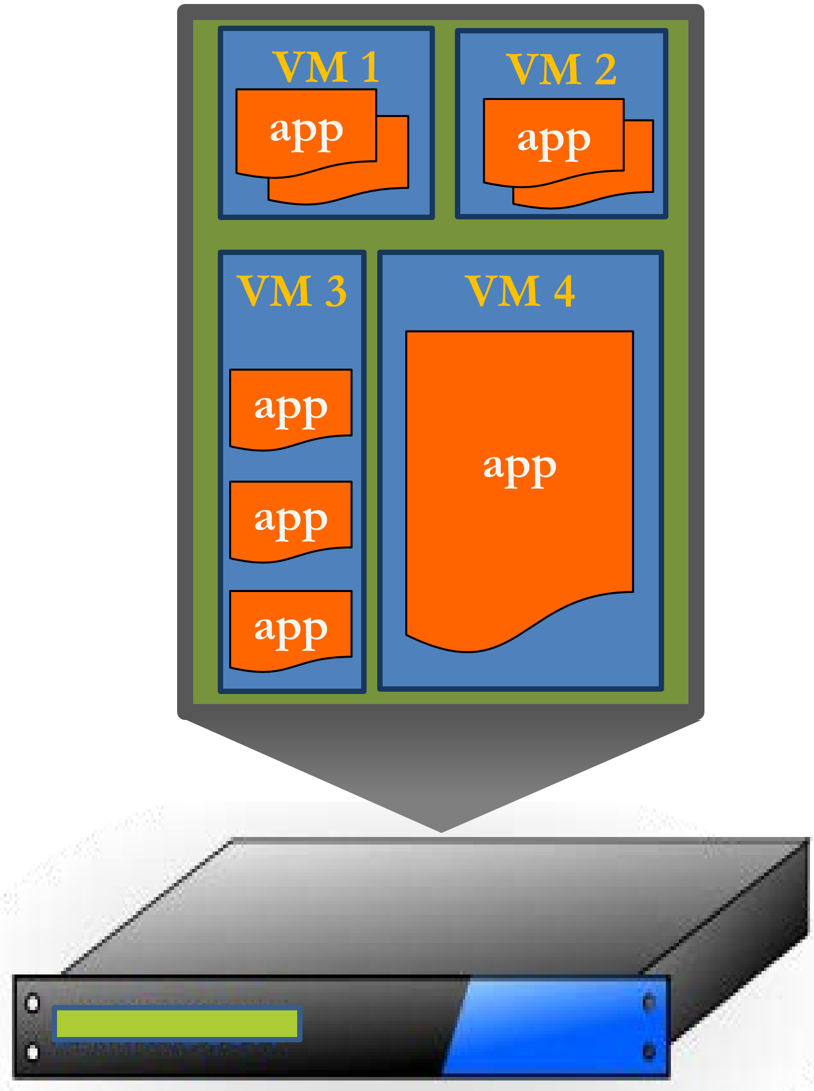


Virtualization technology



- Package resources
- Enforce isolation

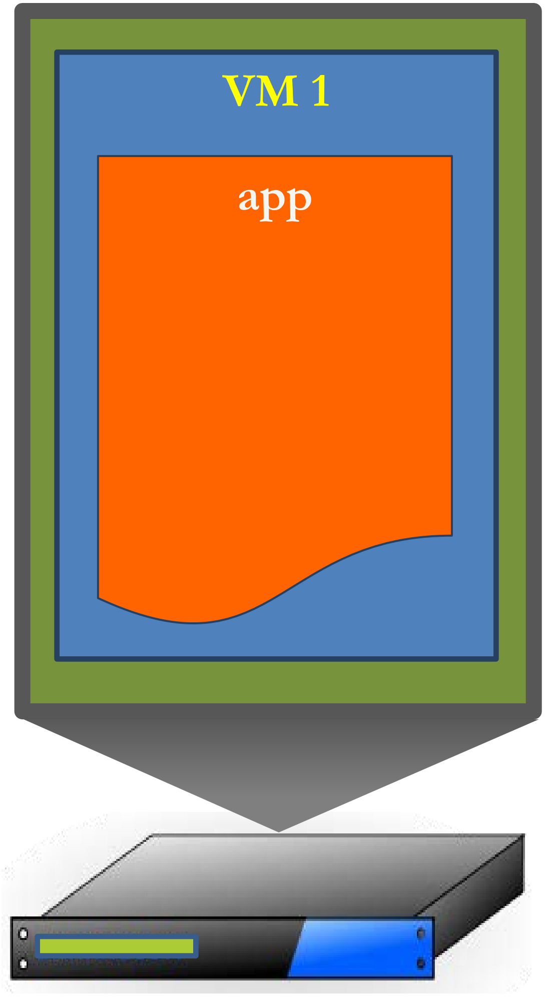
Virtualization technology



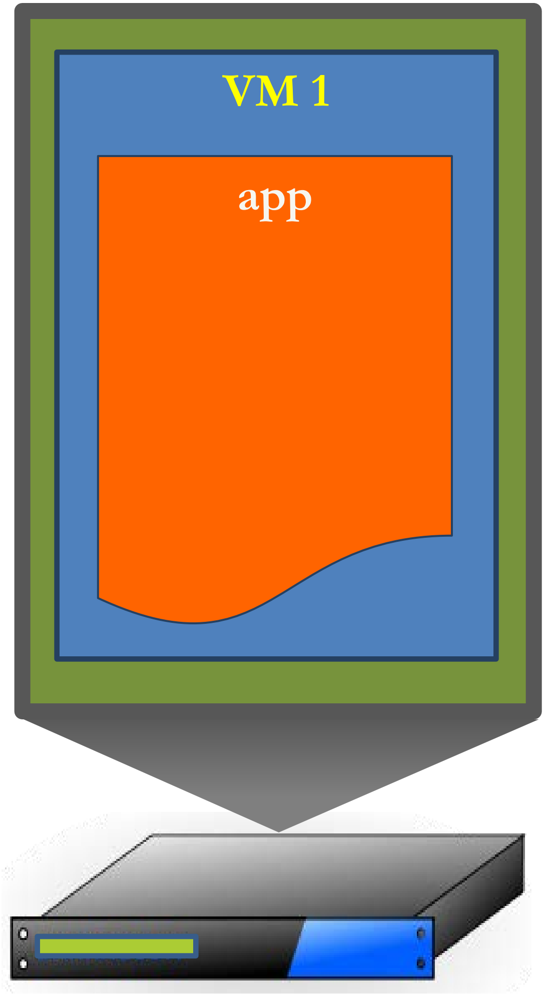
- Package resources
- Enforce isolation

- A fundamental component in cloud technology replying in datacenters

Computation in datacenters

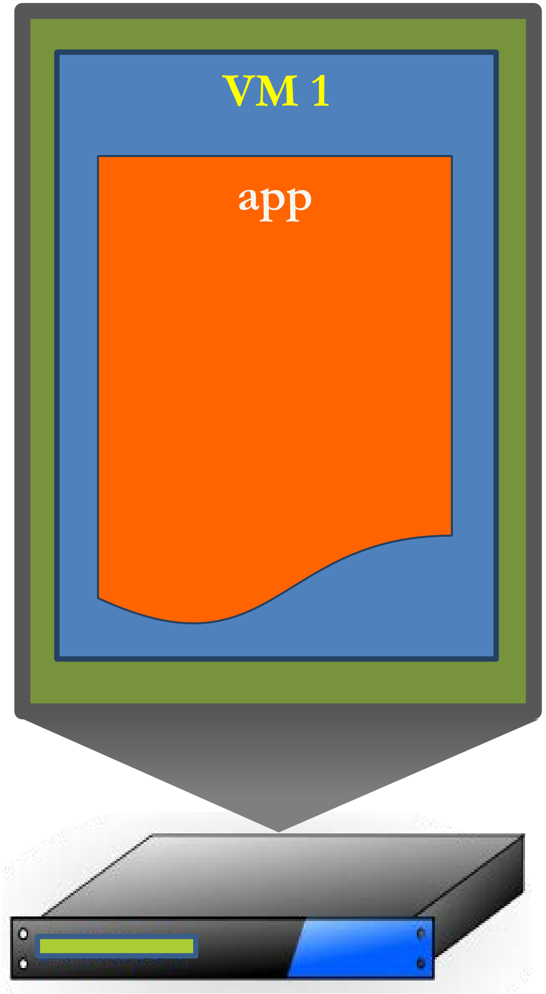


Computation in datacenters



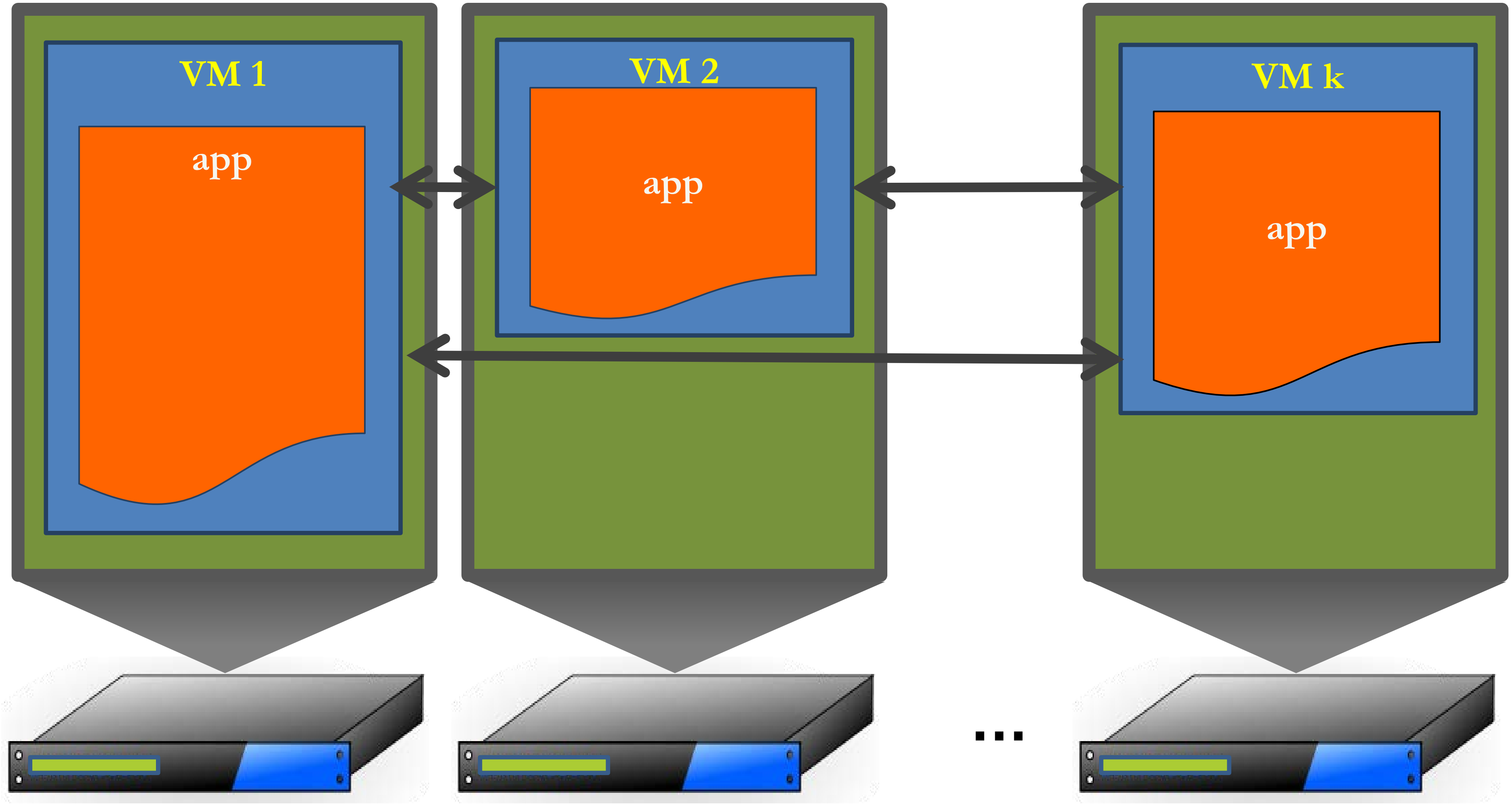
Google **3.2 ~ 12.8 TB data with 2,000 machines [Dean 2004]**

Computation in datacenters



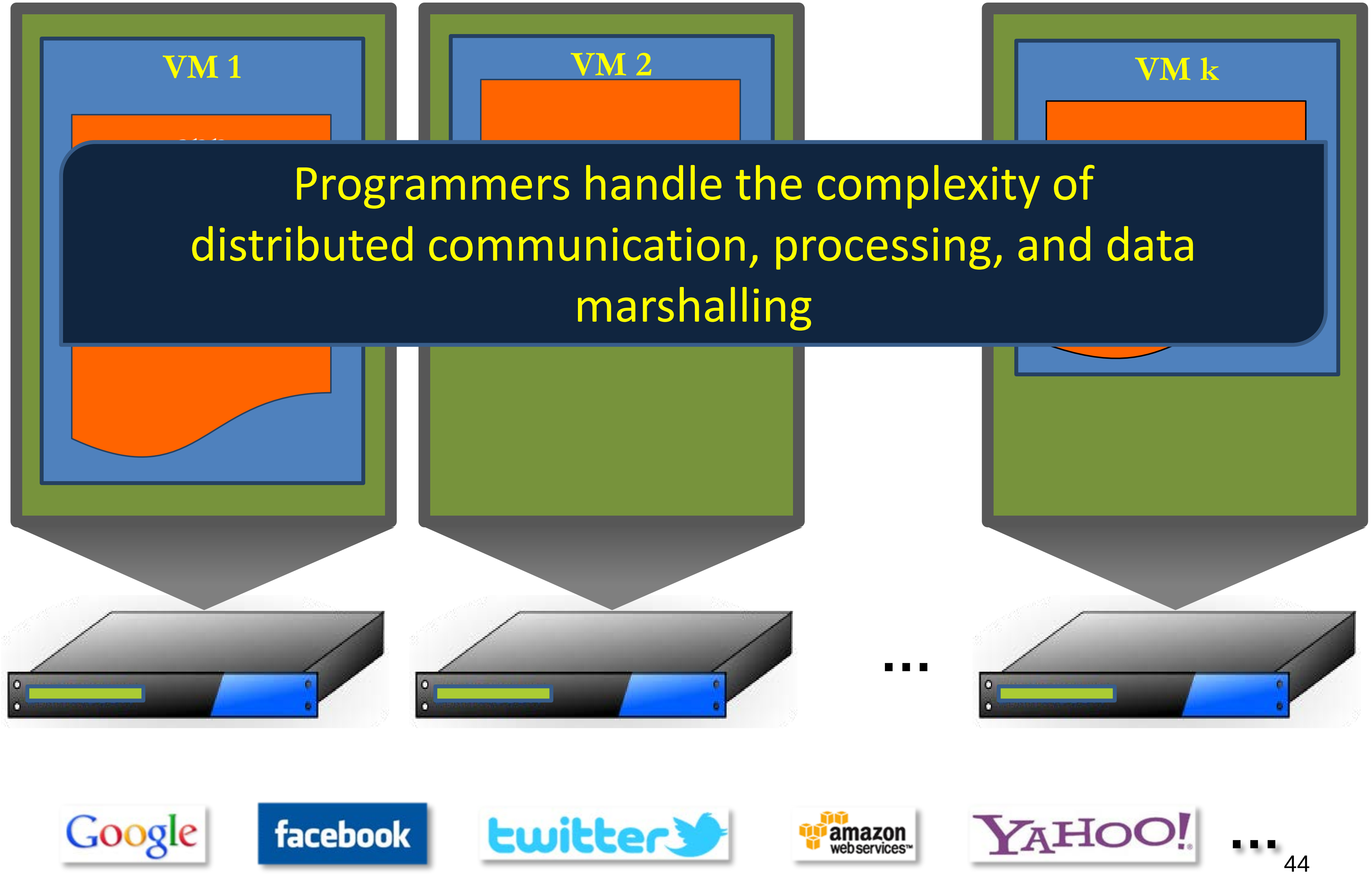
Google facebook twitter amazon webservices YAHOO! …

Computation in datacenters

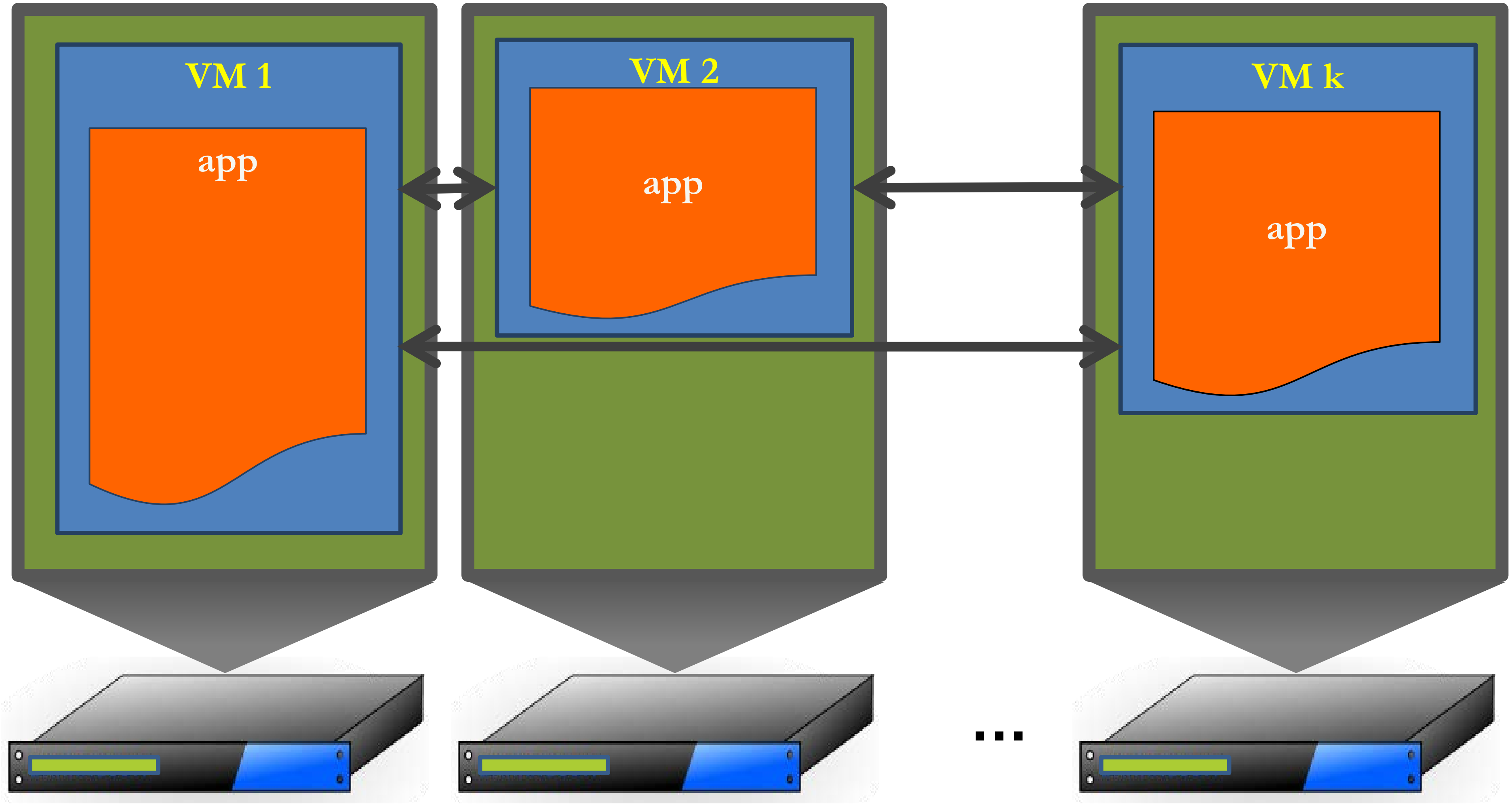


Google facebook twitter amazon webservices YAHOO! ...

Computation in datacenters



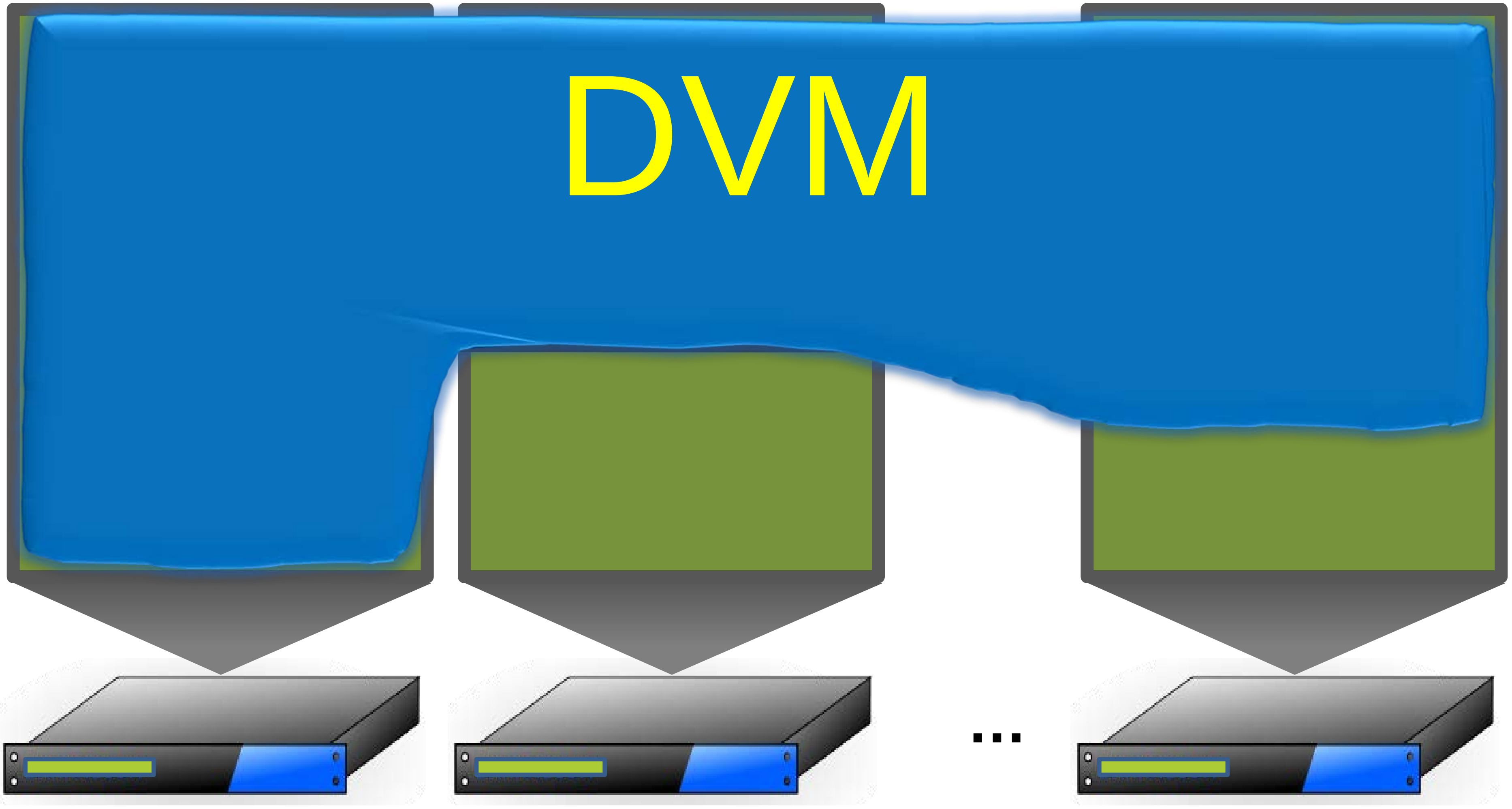
DVM: big virtual machine



DVM: DISA Virtual Machine

DISA: Datacenter Instruction Set Architecture

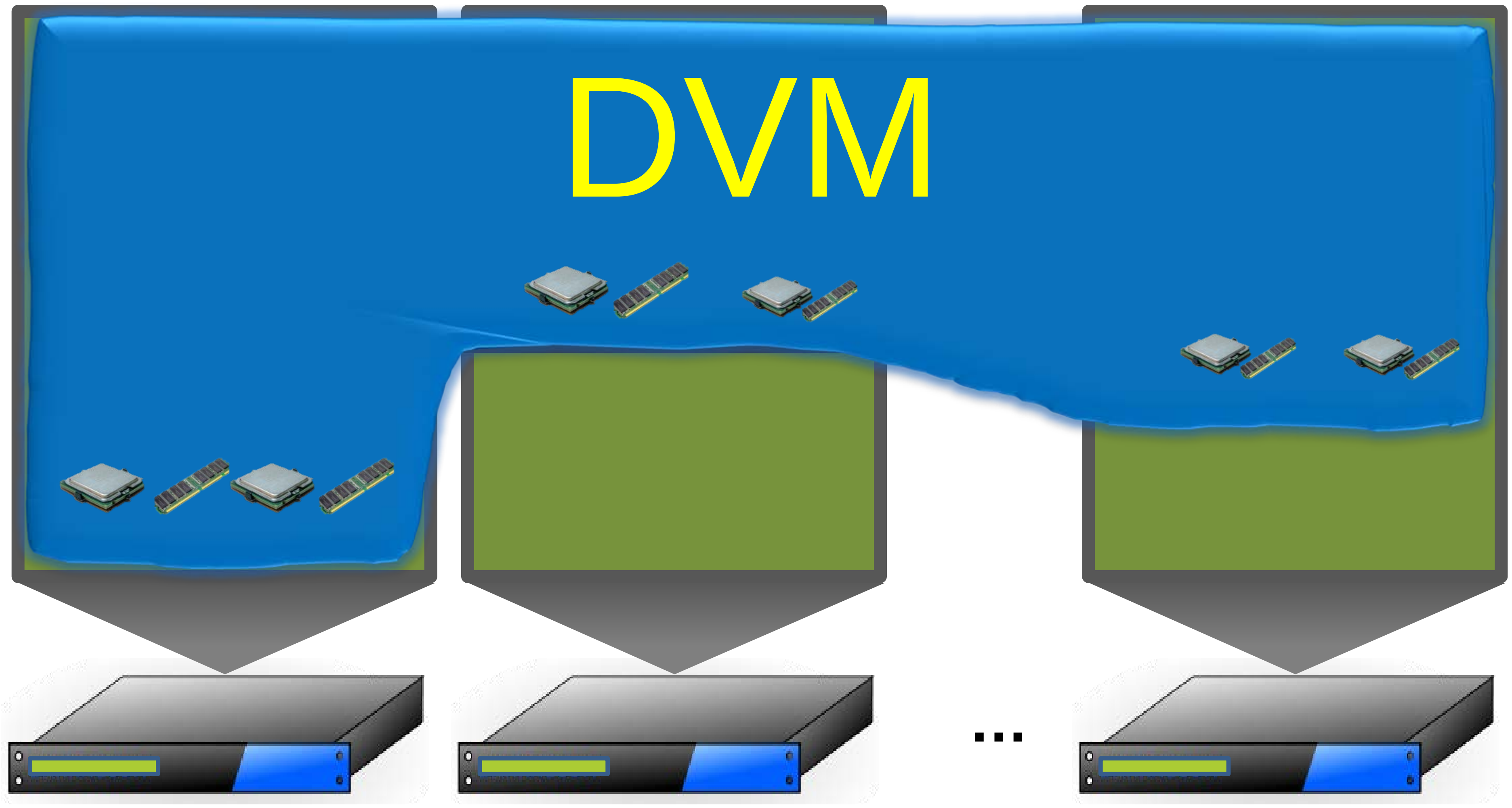
DVM: big virtual machine



DVM: DISA Virtual Machine

DISA: Datacenter Instruction Set Architecture

DVM: big virtual machine



DVM: DISA Virtual Machine

DISA: Datacenter Instruction Set Architecture

DVM: towards a datacenter-scale virtual machine

DVM: big virtual machine

- General
- Scalable (1000s of machines)
- Efficient
- Easy-to-program
- Portable

"The datacenter as a computer" **[Barroso 2009]**

Why not other approaches?

- MapReduce (Hadoop) - application frameworks
- X10 - parallel programming languages
- MPI - System calls/APIs

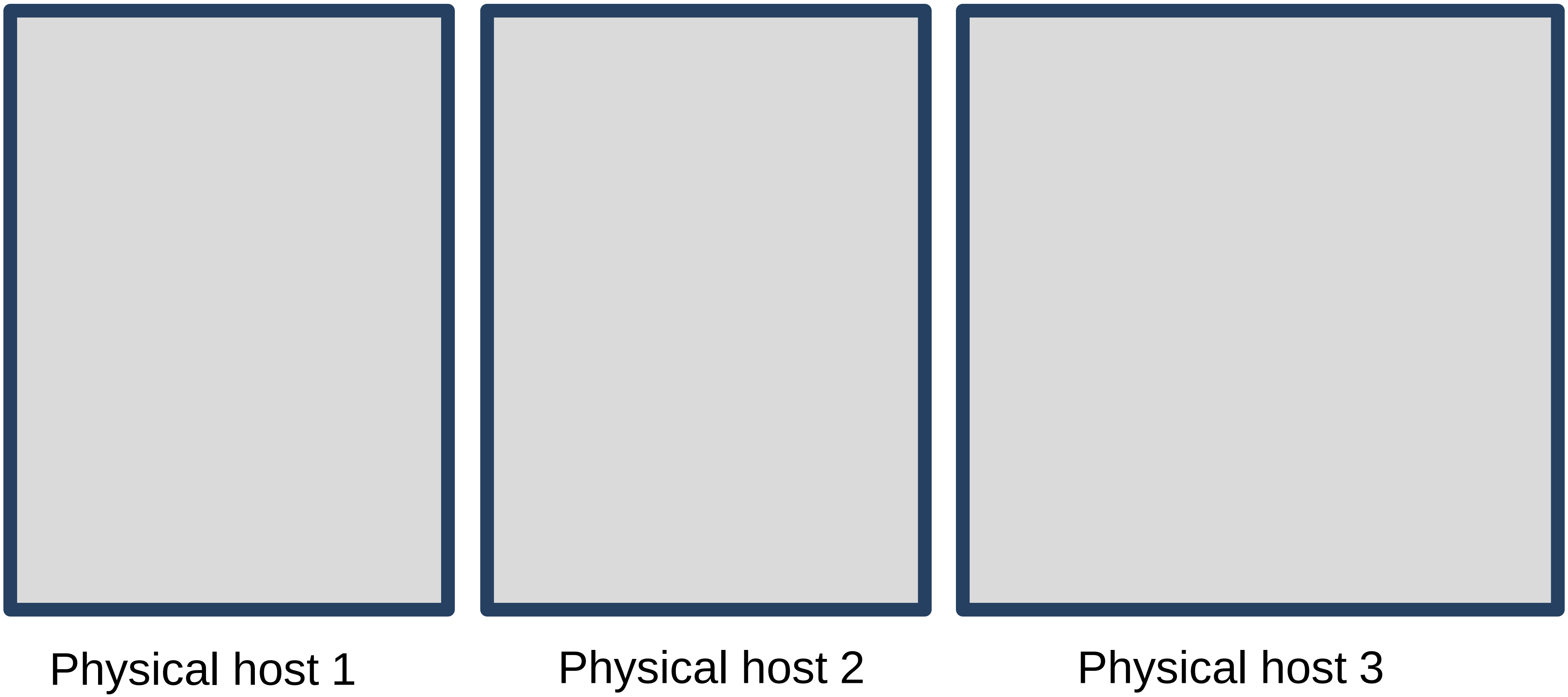
- Increased complexity
  - Partition program state (MapReduce)
  - Programmer specified synchronization (X10)
  - Semantic gaps (MPI)
- Decreased performance
  - 10X improvement is possible (*k*-means)
- Diminished generality
  - Specific control flow and dependence relation (MapReduce)

Talk outline

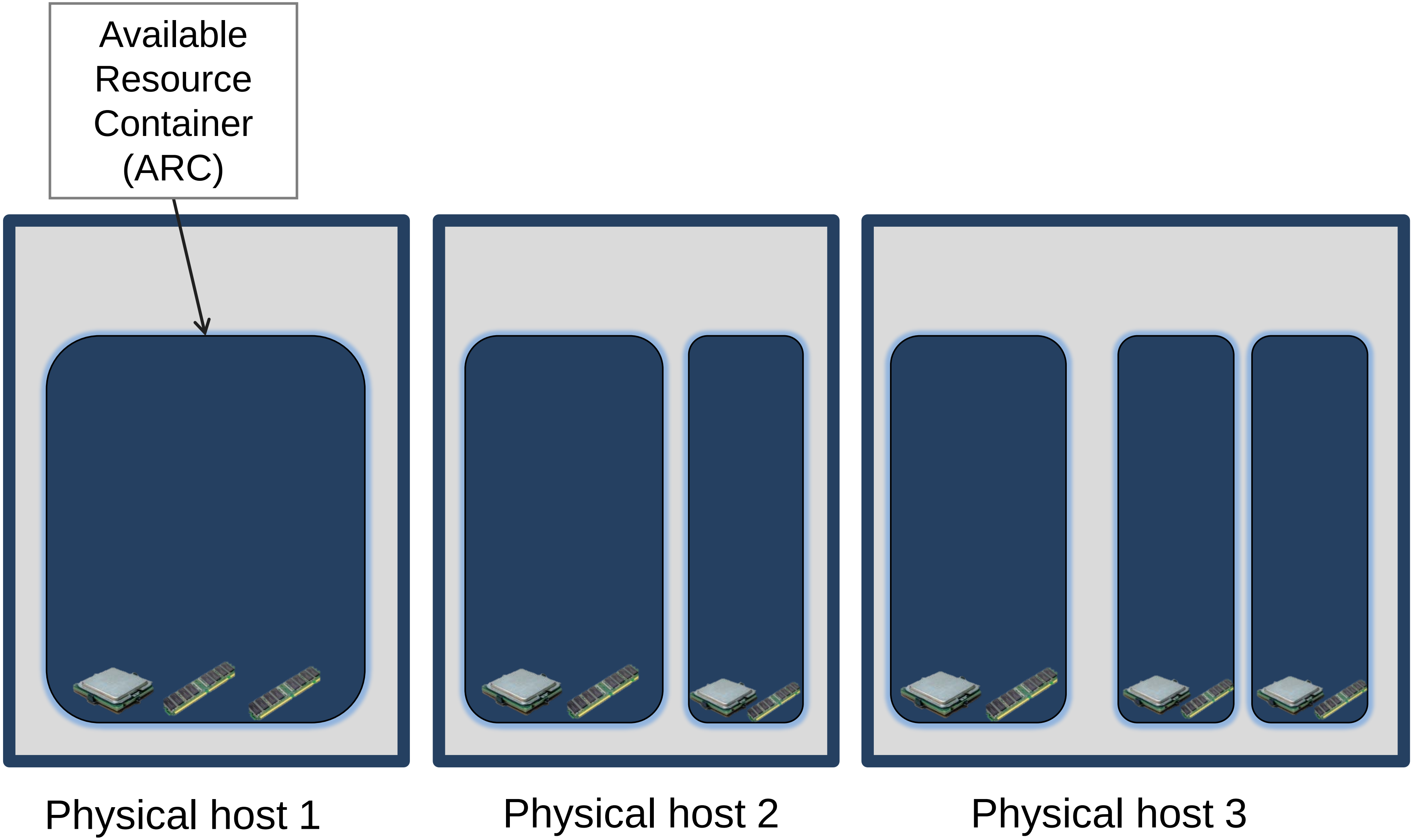
- Motivation
- System design
- Evaluation

DVM architecture

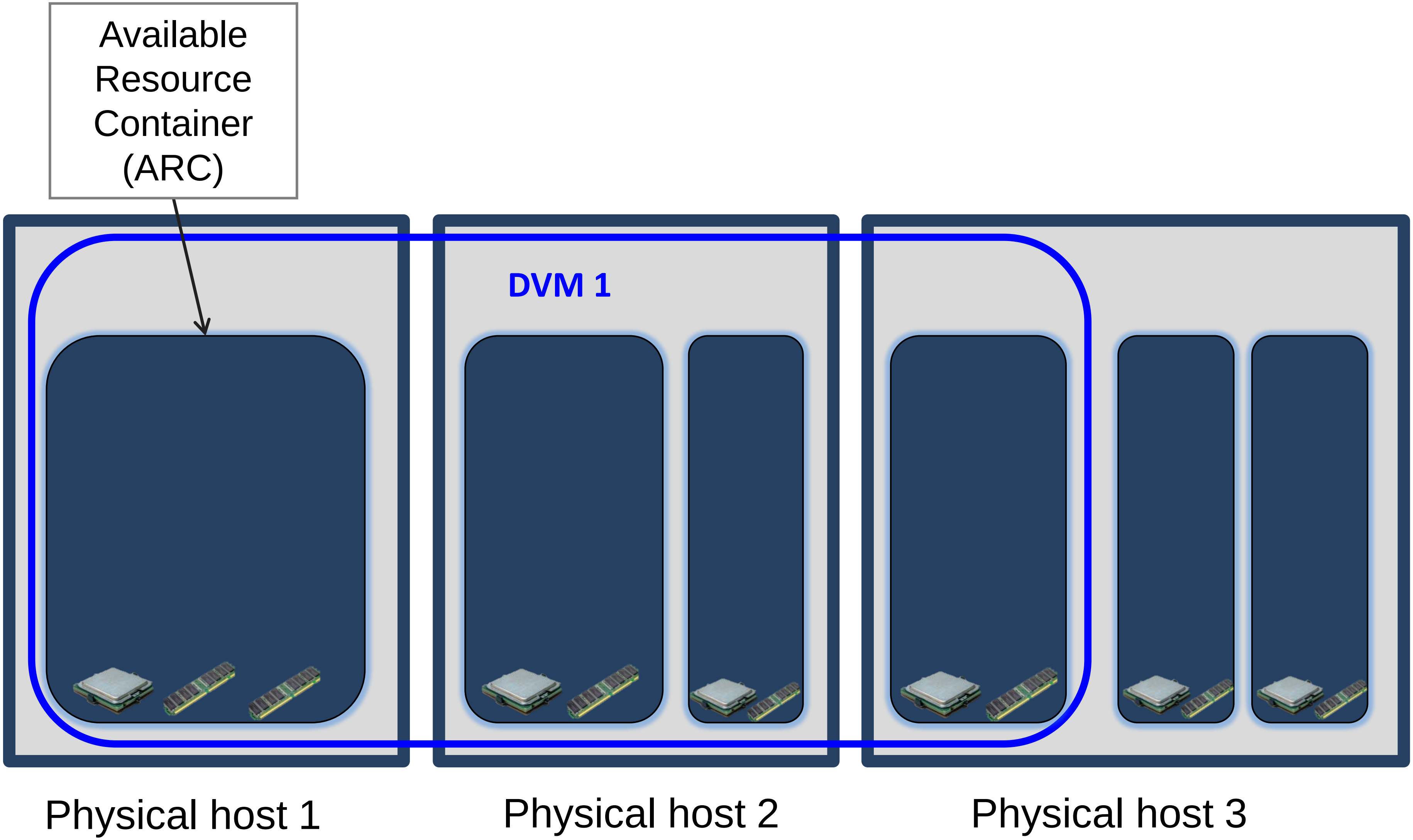
DVM architecture



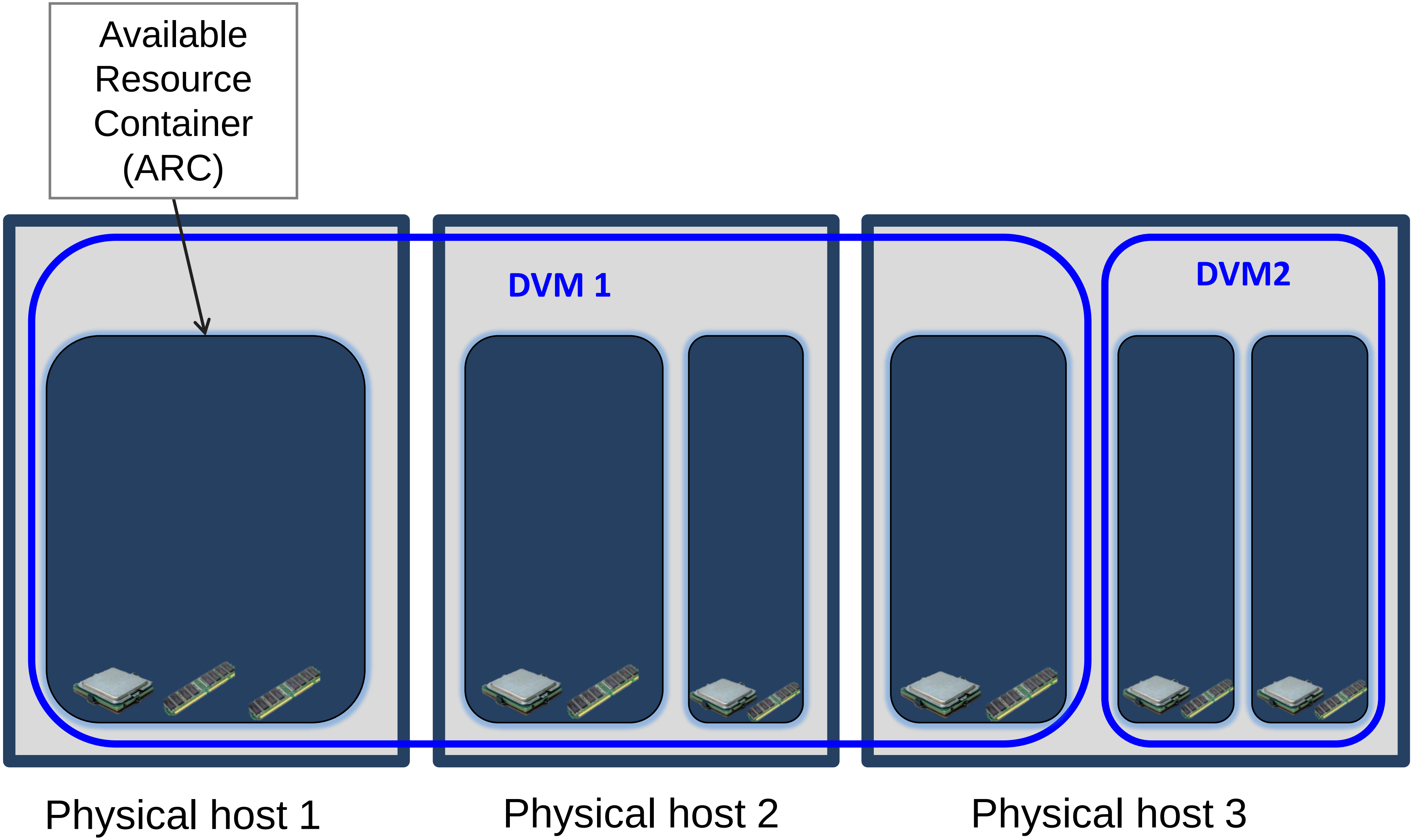
DVM architecture



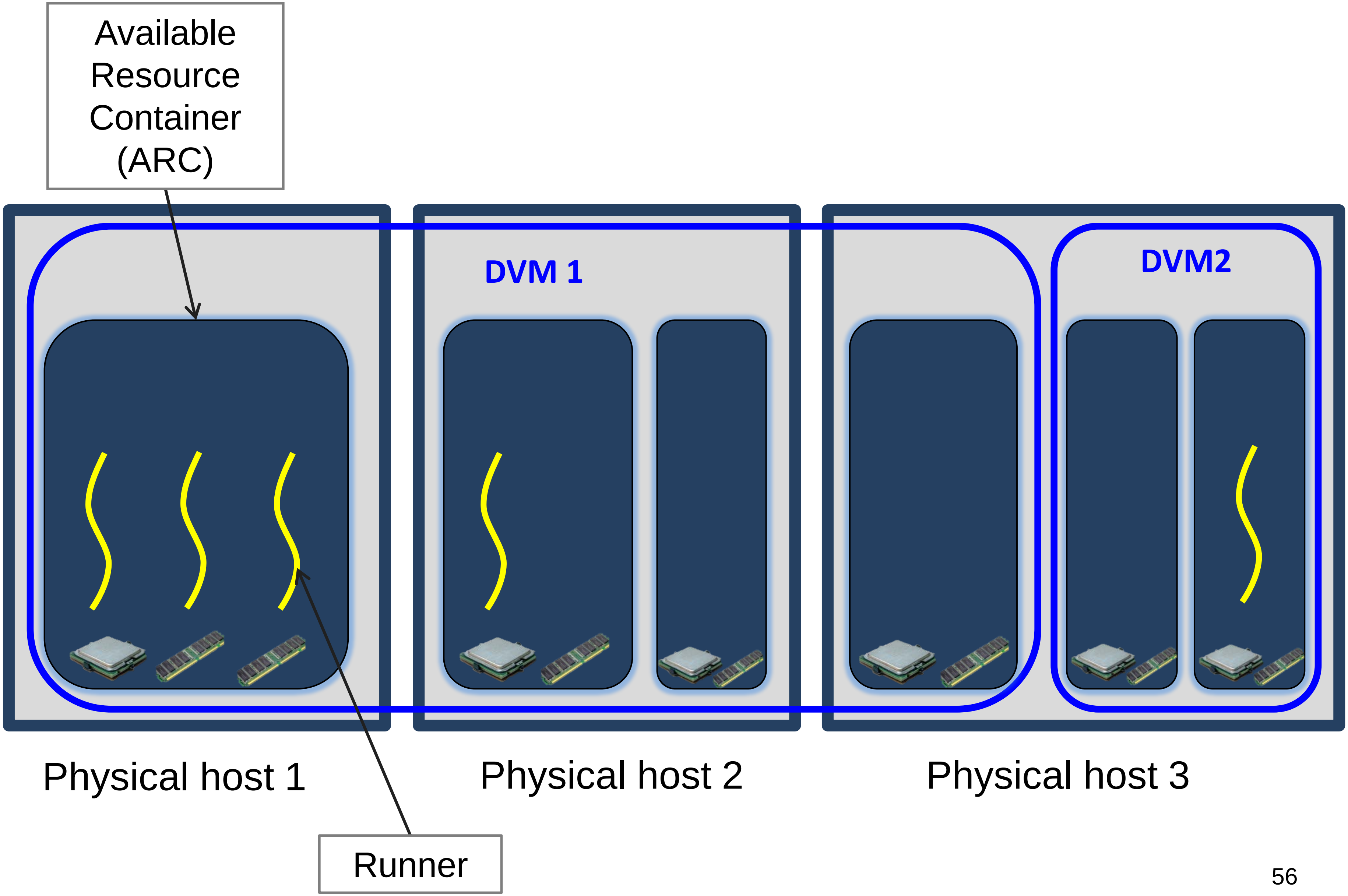
DVM architecture



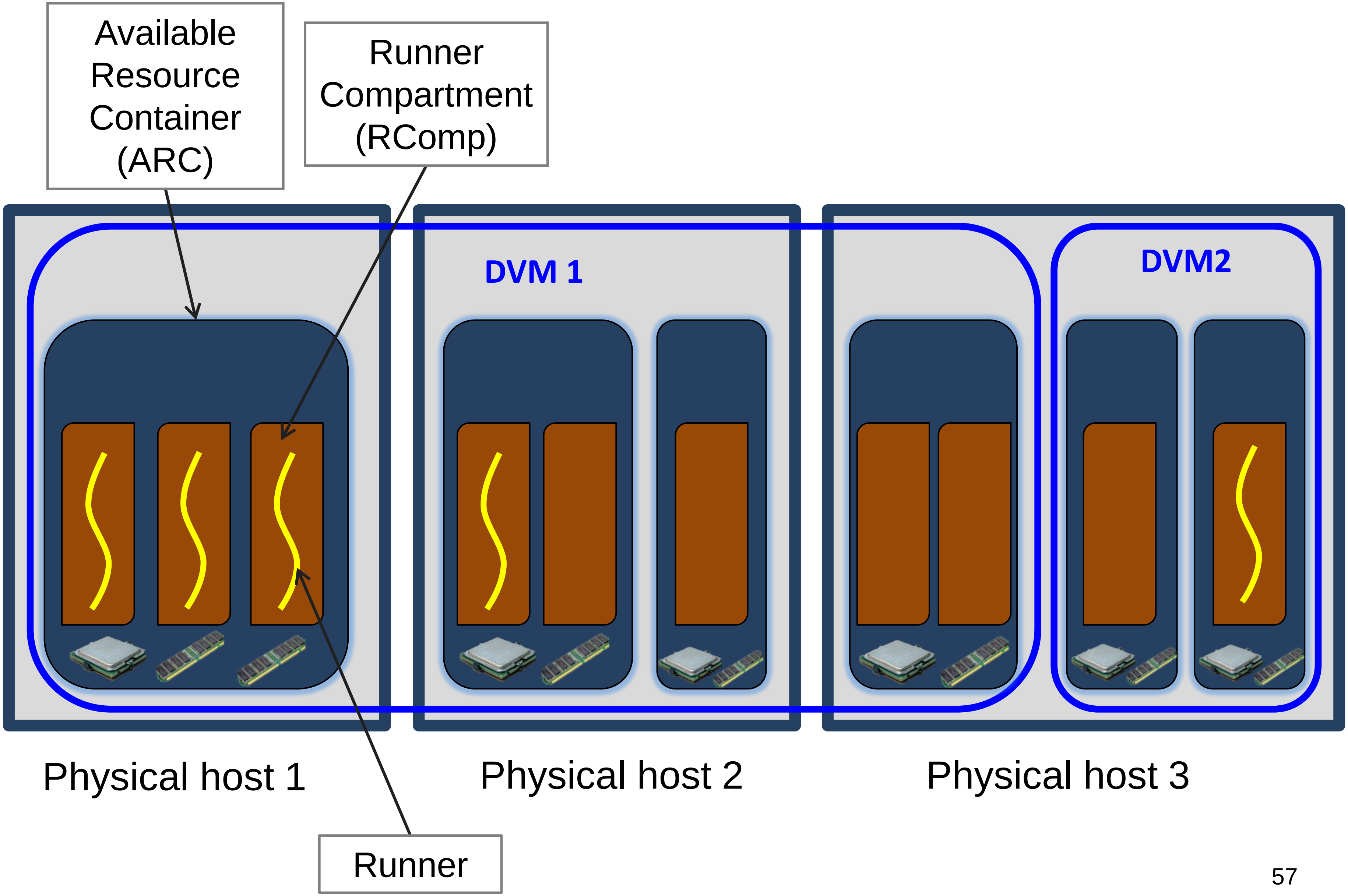
DVM architecture



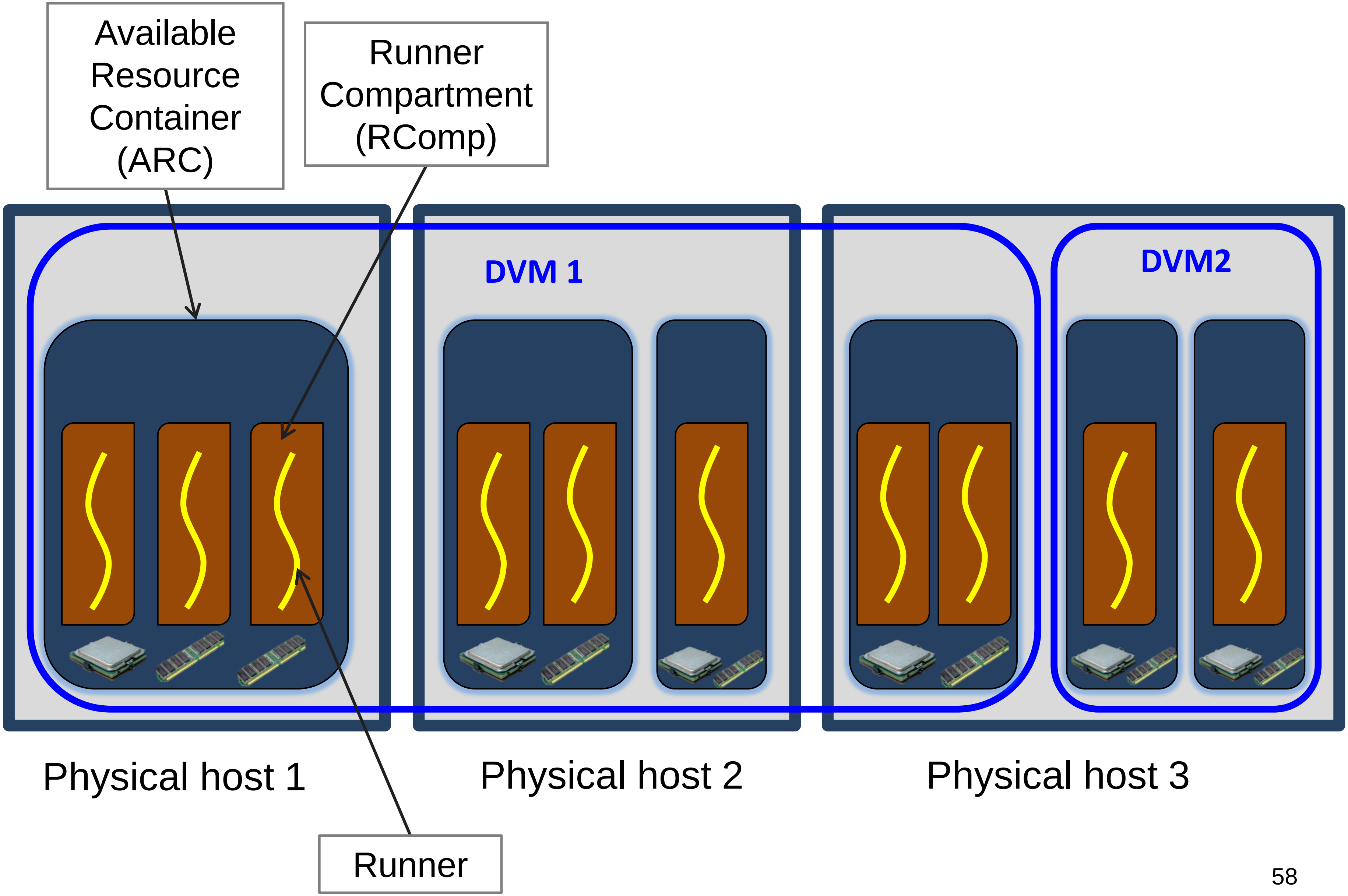
DVM architecture



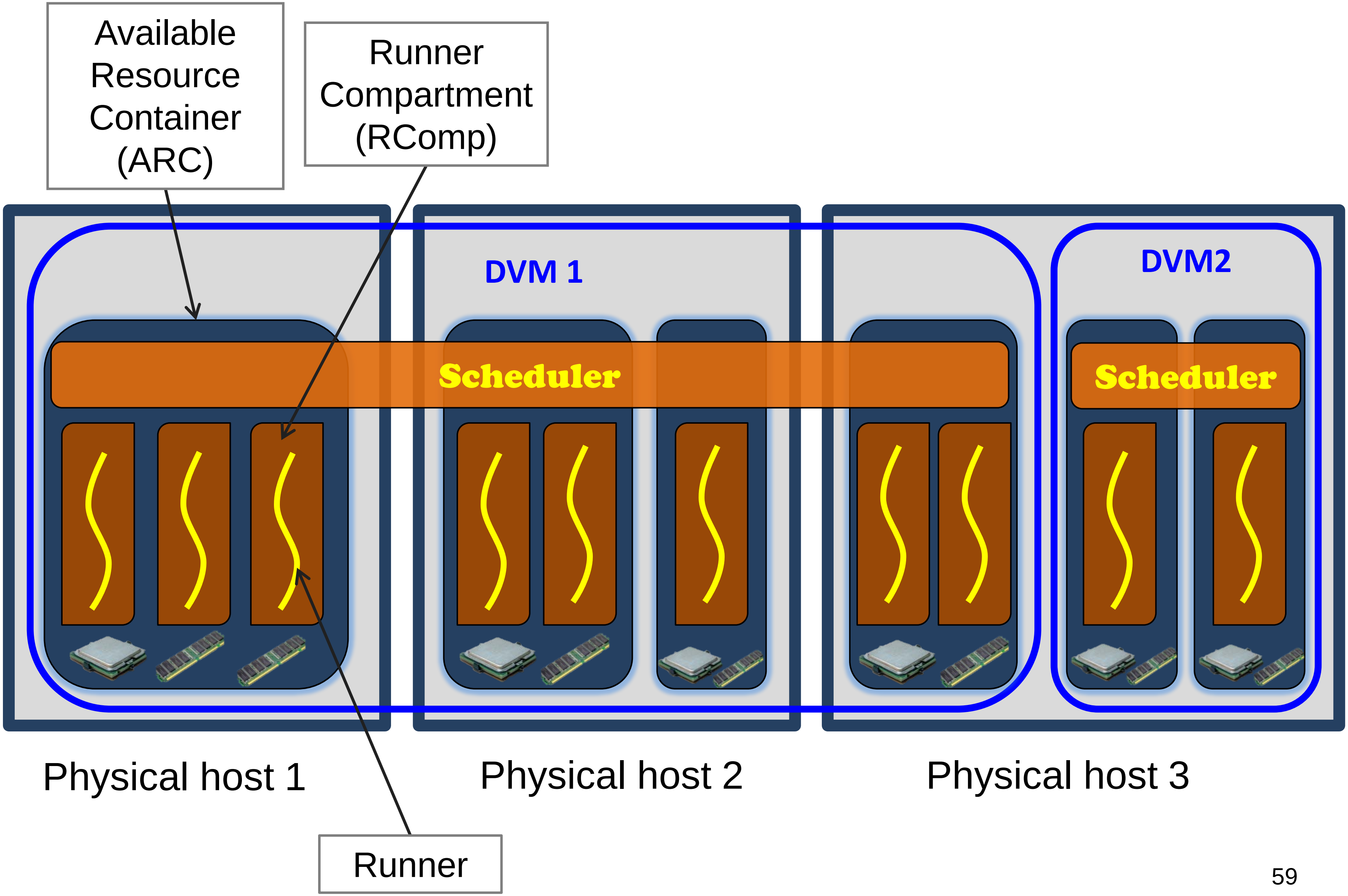
DVM architecture



DVM architecture



DVM architecture



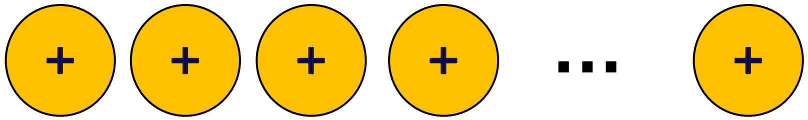
Runners – an example

Calculate the sums of
20,480 integers

Runners – an example

Calculate the sums of
20,480 integers

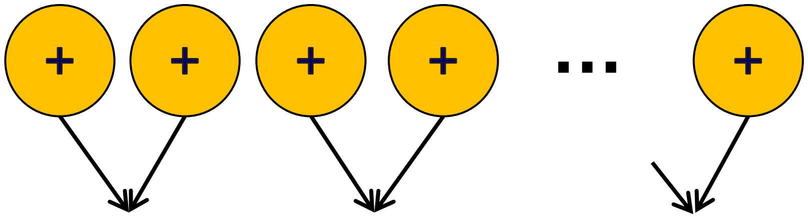
Each task sums two integers



Runners – an example

Calculate the sums of 20,480 integers

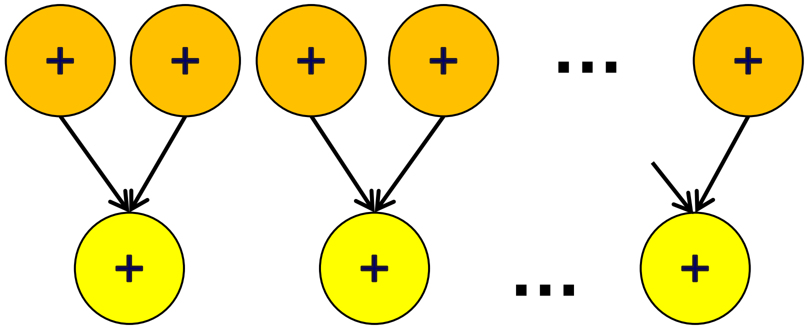
Each task sums two integers



Runners – an example

Calculate the sums of 20,480 integers

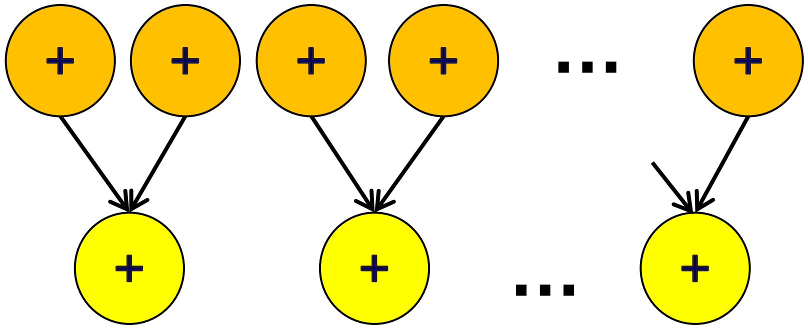
Each task sums two integers



Runners – an example

Calculate the sums of 20,480 integers

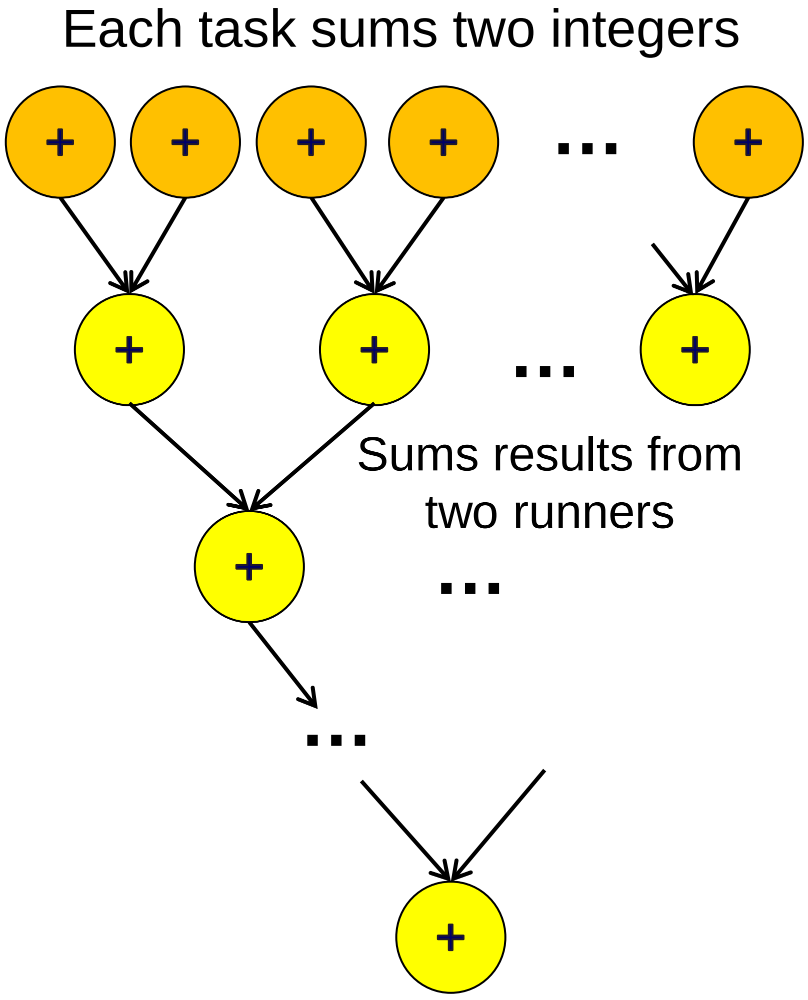
Each task sums two integers



Sums results from two runners

Runners – an example

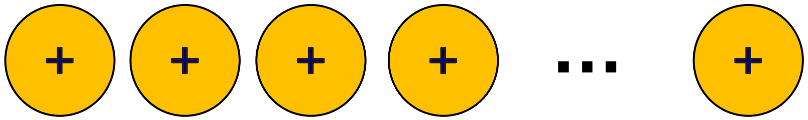
Calculate the sums of
20,480 integers



Runners – an example

Calculate the sums of
20,480 integers

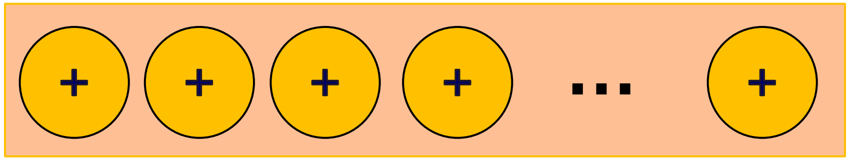
Each task sums two integers

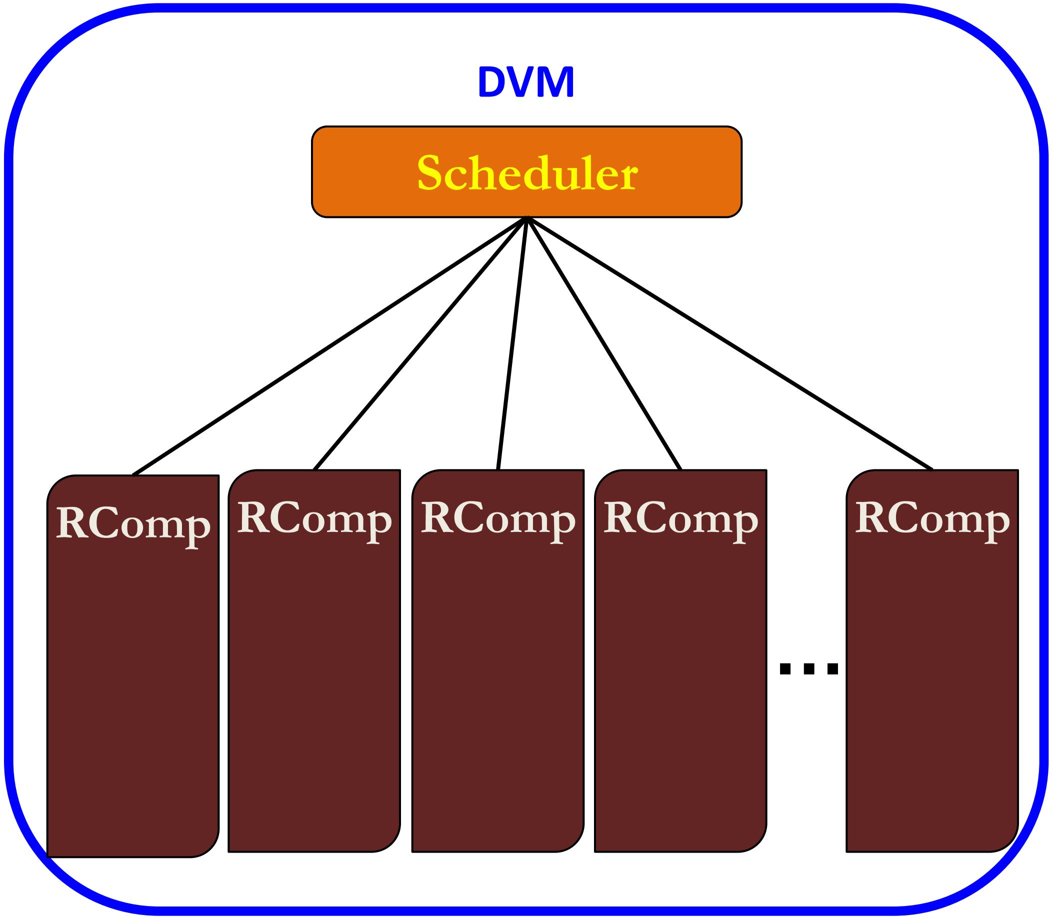


Runners – an example

Calculate the sums of 20,480 integers

Each task sums two integers

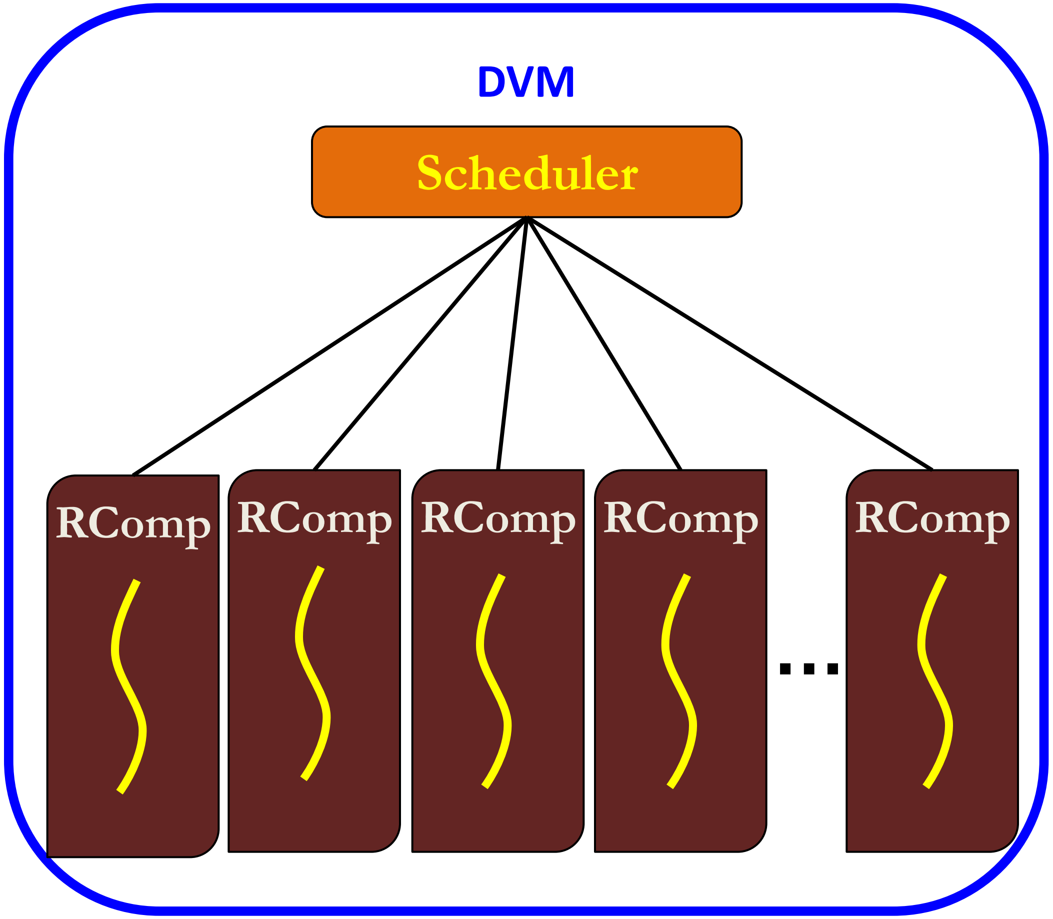




Runners – an example

Calculate the sums of
20,480 integers

Each task sums two integers



Interface between DVM and programs

- Traditional ISAs
  - Clear interface between hardware and software
- Traditional ISAs for DVM?
  - vNUMA: only for small cluster (8 nodes); unable to fully support Itanium's memory semantics (*mf*)
  - Not scalable to a datacenter

Datacenter Instruction Set Architecture

DISA retains the *generality* and *efficiency* of traditional ISAs, and enables the system to scale to many machines

Goals of DISA:

- Efficiently express logic
- Efficient on common hardware
- Easy to implement and port
- Scalable parallelization mechanism and memory model

DISA - instructions

add (0x100001000):q, 8(0x100001000), 0x100000000020

DISA - instructions

add (0x100001000):q, 8(0x100001000), 0x100000000020

opcode

DISA - instructions

add (0x100001000):q, 8(0x100001000), 0x100000000020

opcode operands

DISA - instructions

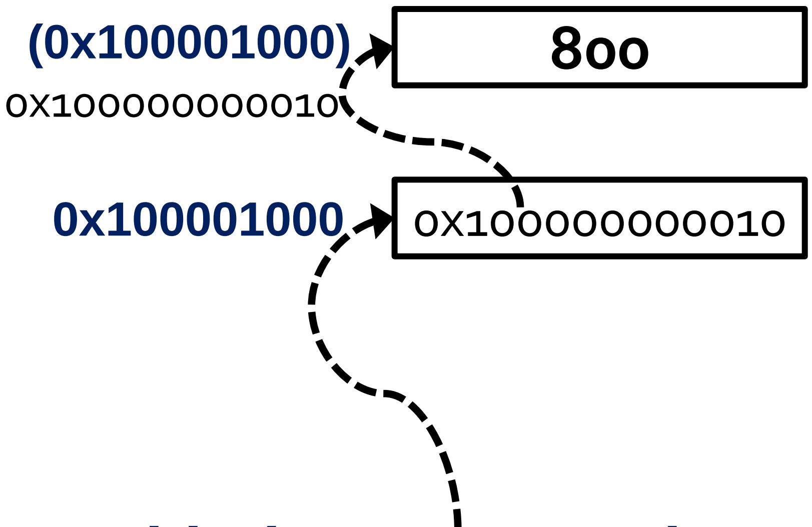
add (0x100001000):q, 8(0x100001000), 0x100000000020

opcode

operands

operand attributes:
64-bit integer

DISA - instructions



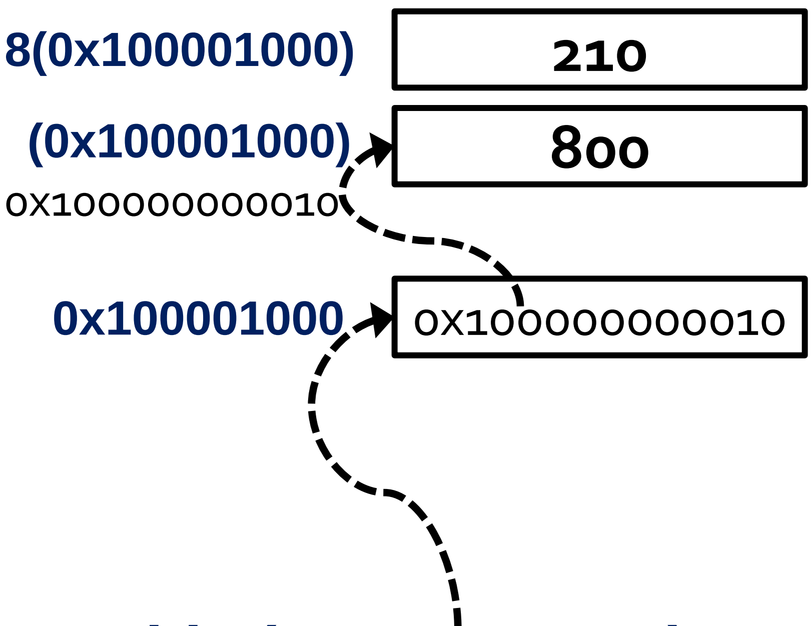
add **(0x100001000):q, 8(0x100001000), 0x100000000020**

opcode

operands

operand attributes:
64-bit integer

DISA - instructions



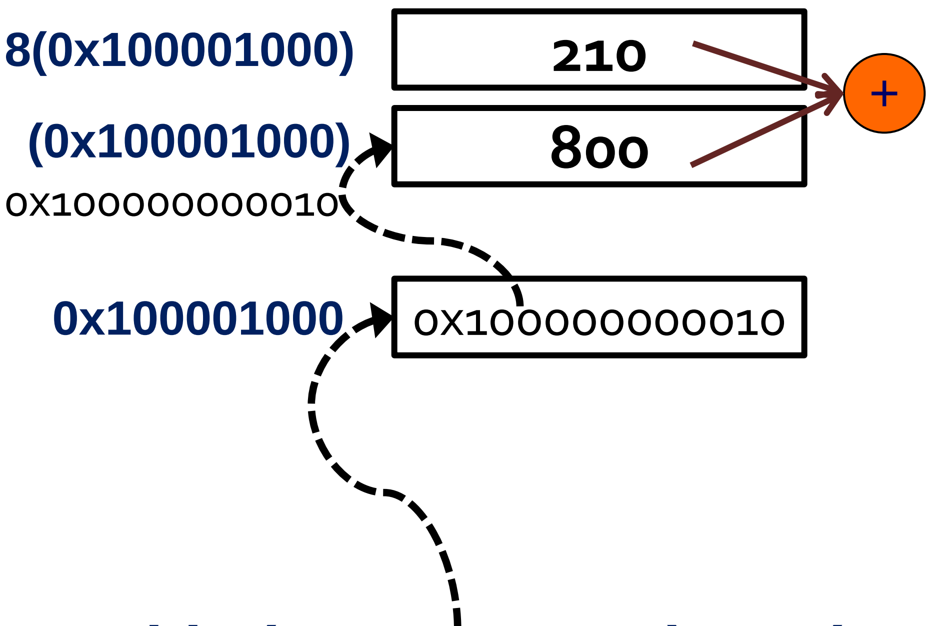
add **(0x100001000):q, 8(0x100001000), 0x10000000020**

opcode

operands

operand attributes:
64-bit integer

DISA - instructions



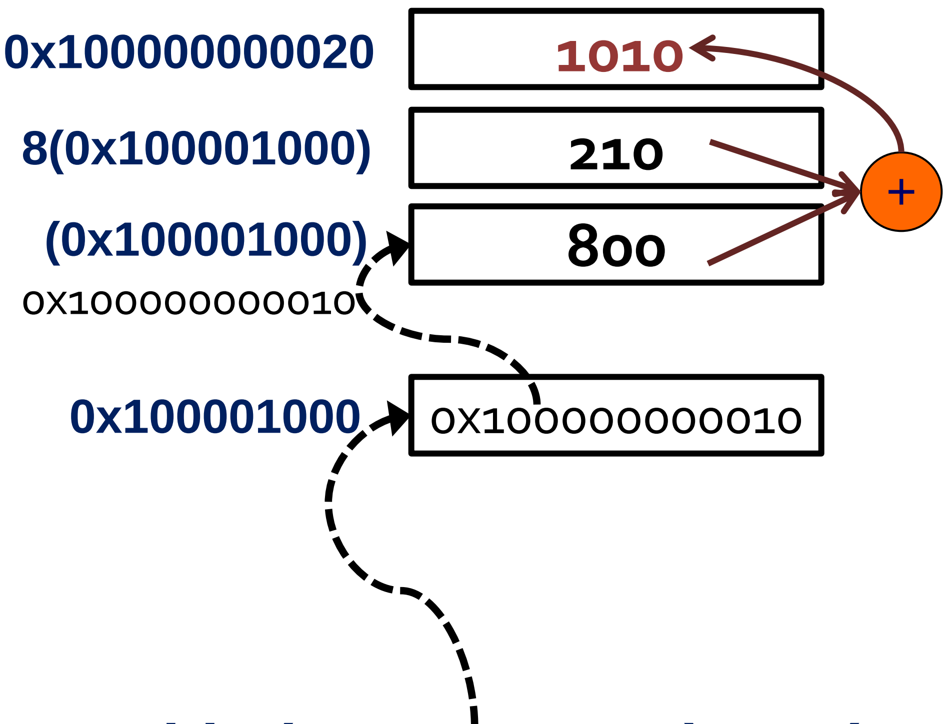
add (0x100001000):q, 8(0x100001000), 0x10000000020

opcode

operands

operand attributes:
64-bit integer

DISA - instructions



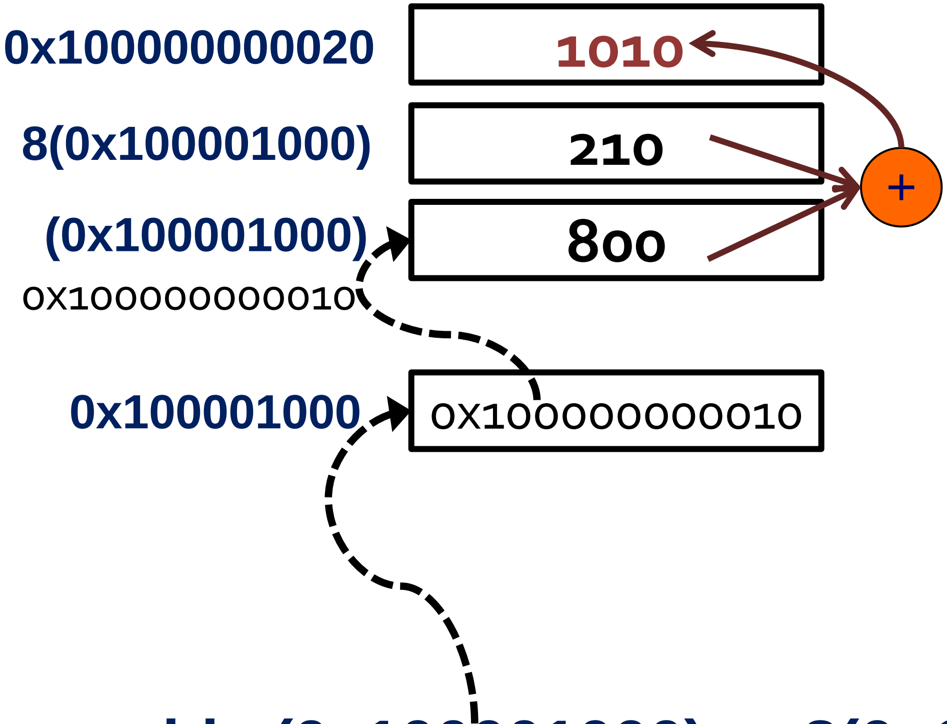
add (0x100001000):q, 8(0x100001000), 0x100000000020

opcode

operands

operand attributes:
64-bit integer

DISA - instructions



- Unified operand address based on memory
- Orthogonality in instruction design
- Selected group of frequently used instructions for efficiency
- Support for massive, flexible and efficient parallel processing

add (0x100001000):q, 8(0x100001000), 0x100000000020

opcode

operands

operand attributes:
64-bit integer

DISA - instructions

Instruction	Operands	Effect
mov	D1, M1	Move [D1] to M1
add	D1, D2, M1	Add [D1] and [D2]; store the result in M1
sub		
mul		
div		
and	D1, D2, M1	Store the bitwise AND of [D1] and [D2] in M1
or	D1, D2, M1	Store the bitwise inclusive OR of [D1] and [D2] in M1
xor	D1, D2, M1	Store the bitwise exclusive OR of [D1] and [D2] in M1
br	D1, D2, M1	Compare [D1] and [D2]; jump to M1 depending on the comparing result
bl	M1, M2	Branch and link (procedure call)
newr	M1, M2, M3, M4	Create a new runner
exit		Exit and commit or abort

Selected group of frequently used instructions

DISA - instructions

Instruction	Operands	Effect
mov	D1, M1	Move [D1] to M1
add	D1, D2, M1	Add [D1] and [D2]; store the result in M1
sub	[illegible]	[illegible]
mul	[illegible]	[illegible]
div	D1, [illegible]	[illegible] [D1] by [D2]; store the result in M1
and	D1, D2, M1	Store the bitwise AND of [D1] and [D2] in M1
or	D1, D2, M1	Store the bitwise inclusive OR of [D1] and [D2] in M1
xor	D1, D2, M1	Store the bitwise exclusive OR of [D1] and [D2] in M1
br	D1, D2, M1	Compare [D1] and [D2]; jump to M1 depending on the comparing result
bl	M1, M2	Branch and link (procedure call)
newr	M1, M2, M3, M4	Create a new runner
exit		Exit and commit or abort

Selected group of frequently used instructions

Instructions for massive, flexible and efficient parallel processing

Store runner state

Programming on a big single computer

- Large, flat, and unified memory space
  - Shared region (SR) and private region (PR) ~64 TBs and 4 GBs

Challenge: thousands of runners access SR concurrently

- A snapshot on interested ranges for a runner
  - Updates affect associated snapshot => concurrent accesses
  - Most accesses handled at native speed
  - Coordination only needed for committing memory ranges

Manage runners

Parent runner creates
10,240 child runners

Share data

Manage runners

Parent runner creates
10,240 child runners

Commit 10,240 times?

Share data

Manage runners

Parent runner creates 10,240 child runners

Commit 10,240 times?

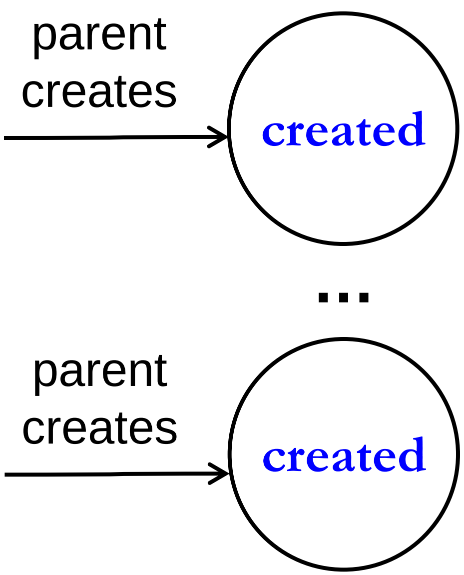
Share data

Manage runners

Parent runner creates
10,240 child runners

Commit 10,240 times?

Share data

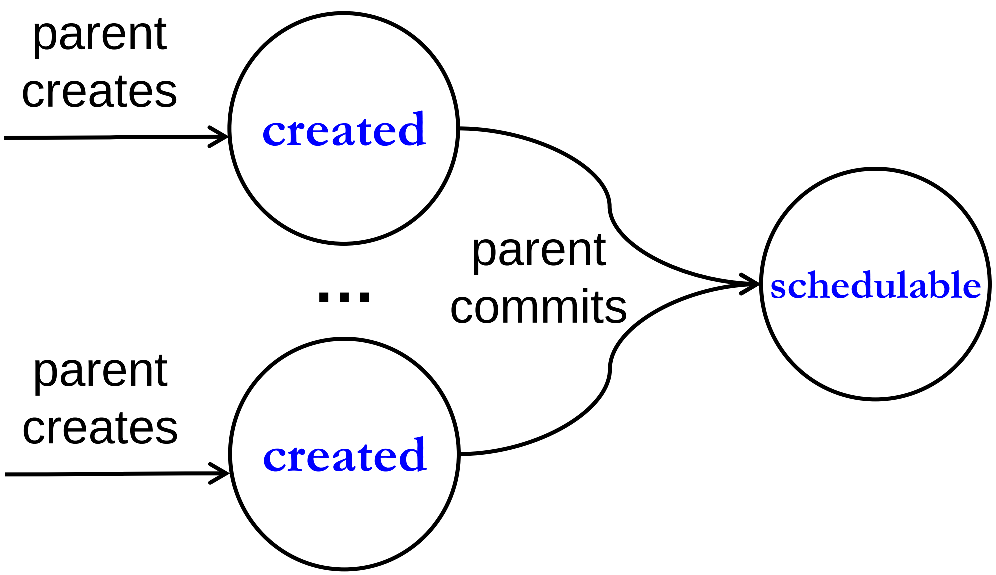


Manage runners

Parent runner creates 10,240 child runners

Commit 10,240 times?

Share data

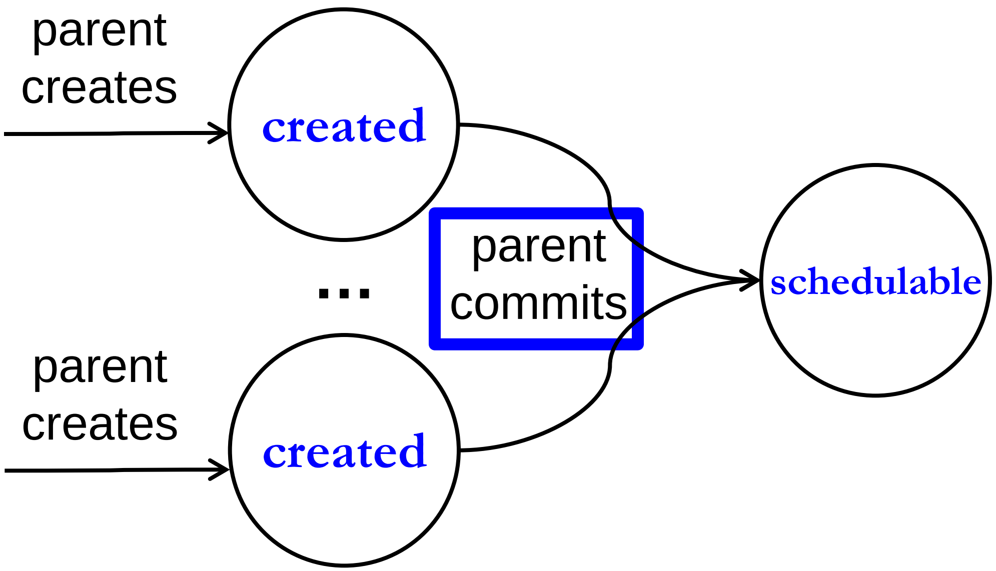


Manage runners

Parent runner creates
10,240 child runners

Commit 10,240 times?

Share data

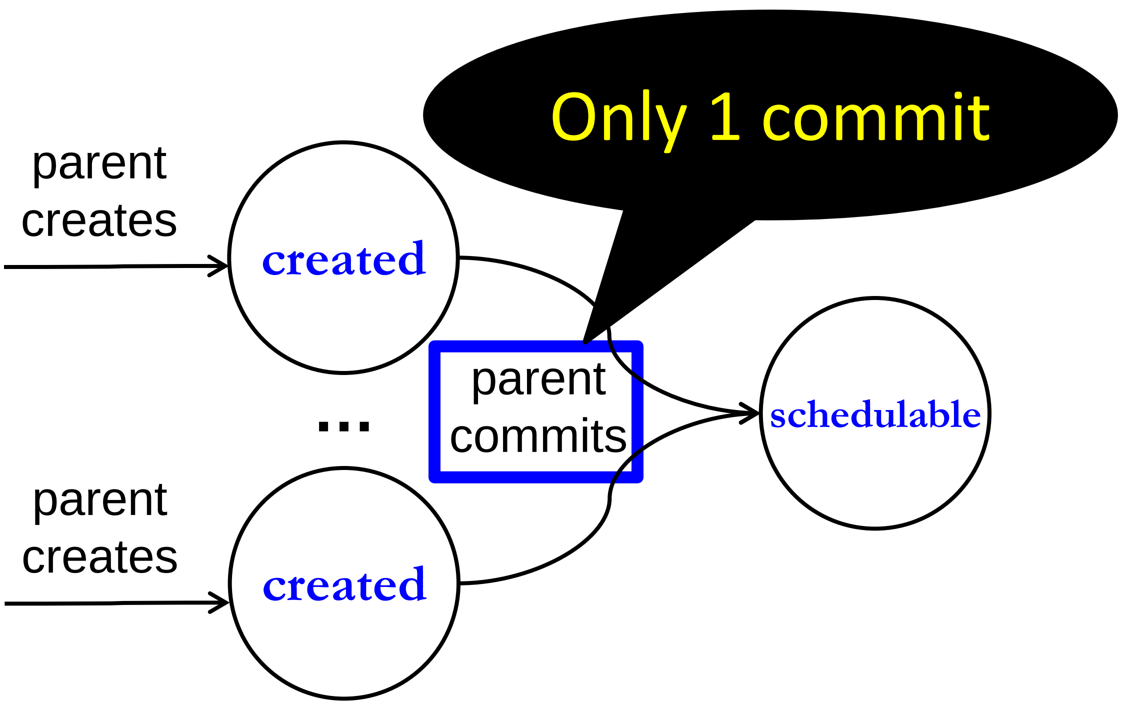


Manage runners

Parent runner creates 10,240 child runners

Commit 10,240 times?

Share data

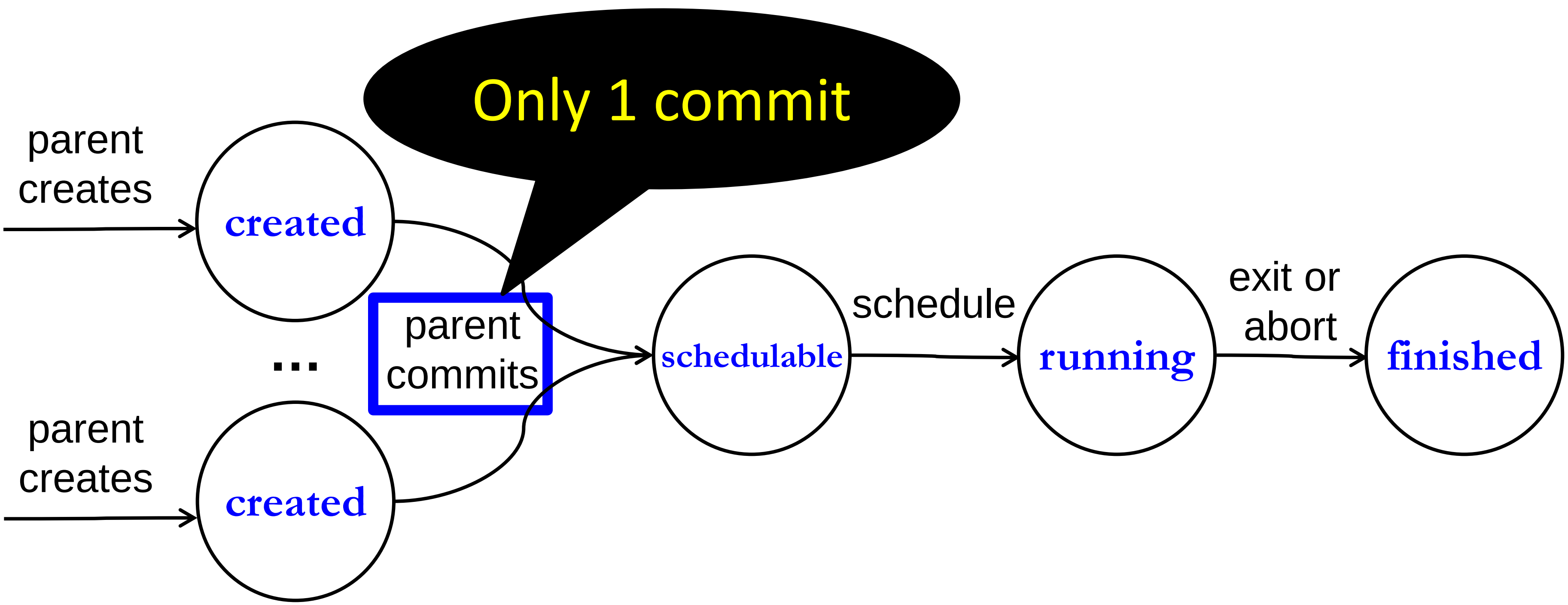


Manage runners

Parent runner creates
10,240 child runners

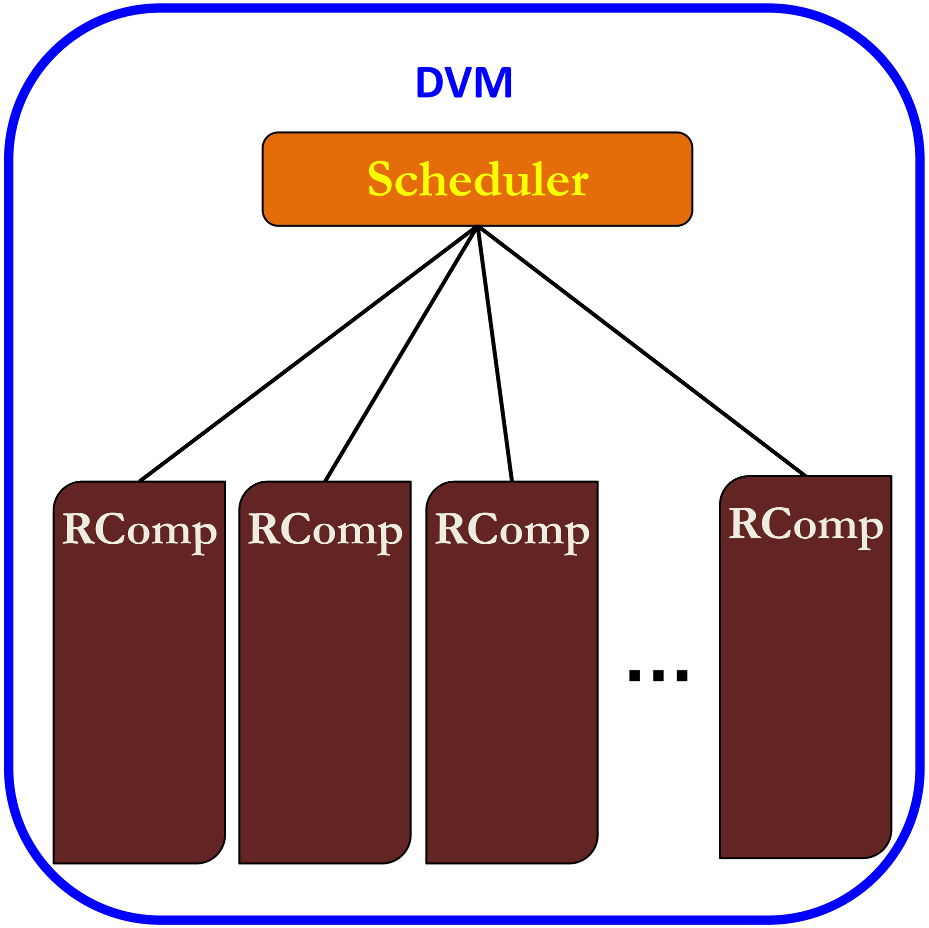
Commit 10,240 times?

Share data

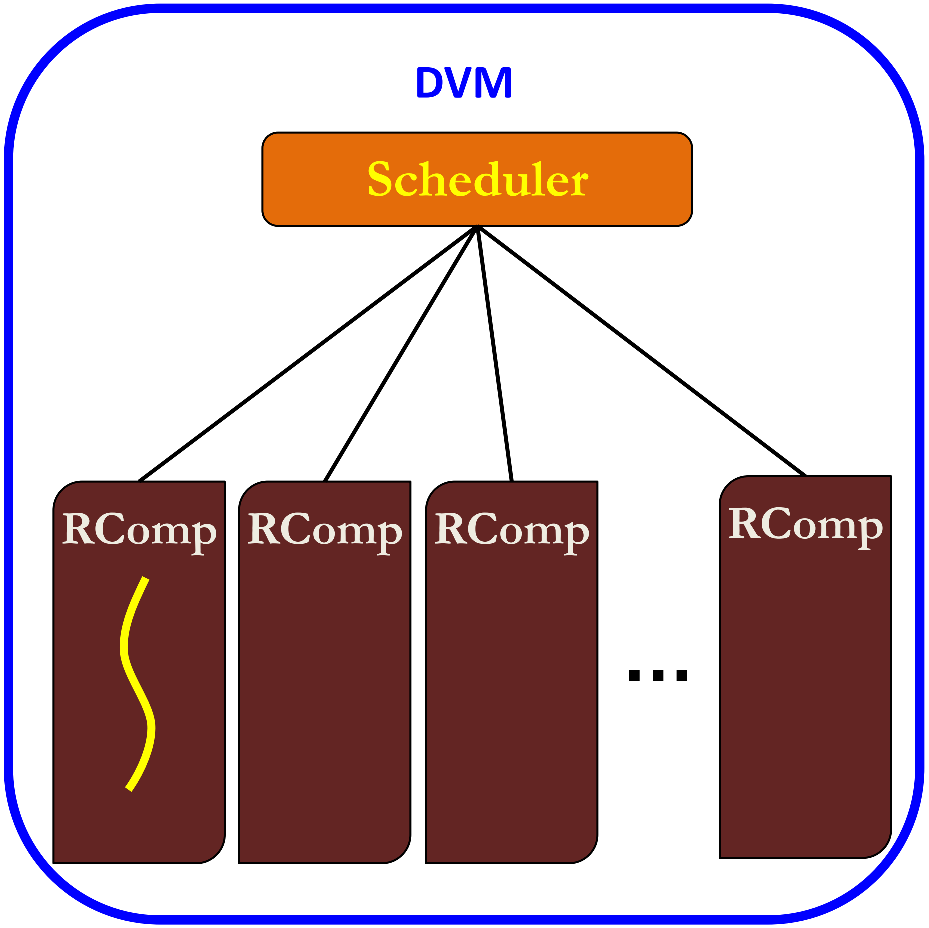


Many-runner parallel execution

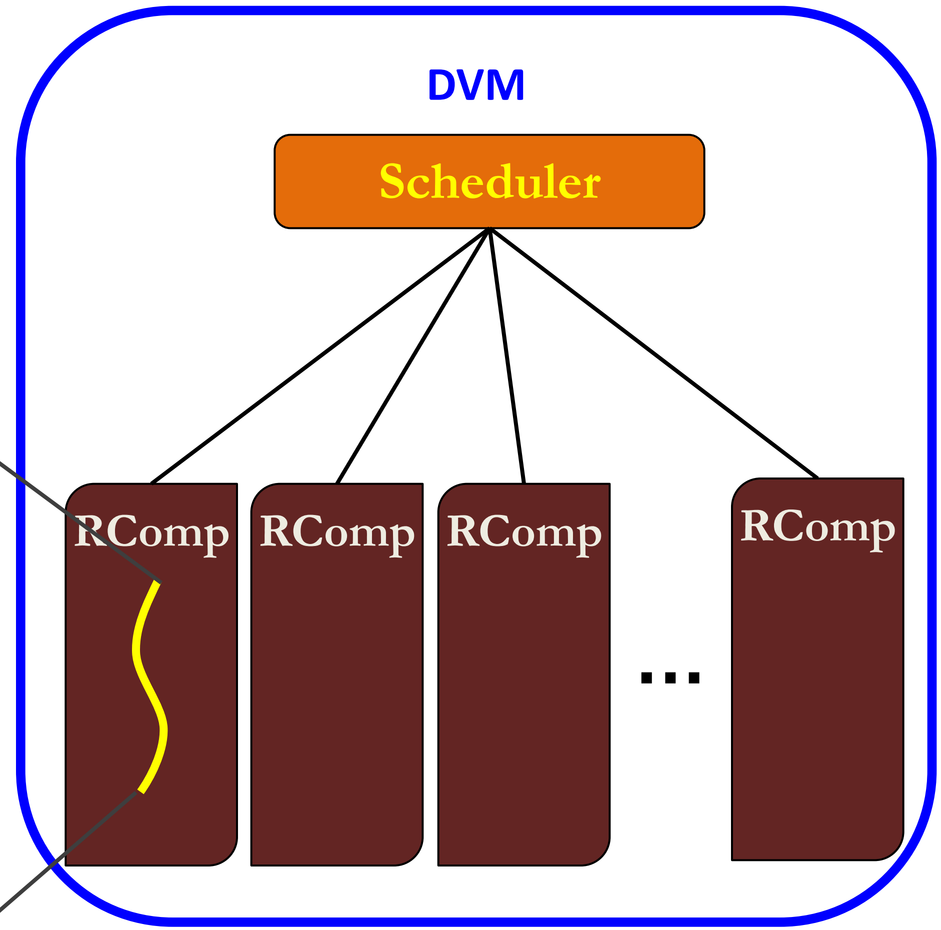
Many-runner parallel execution



Many-runner parallel execution



Many-runner parallel execution

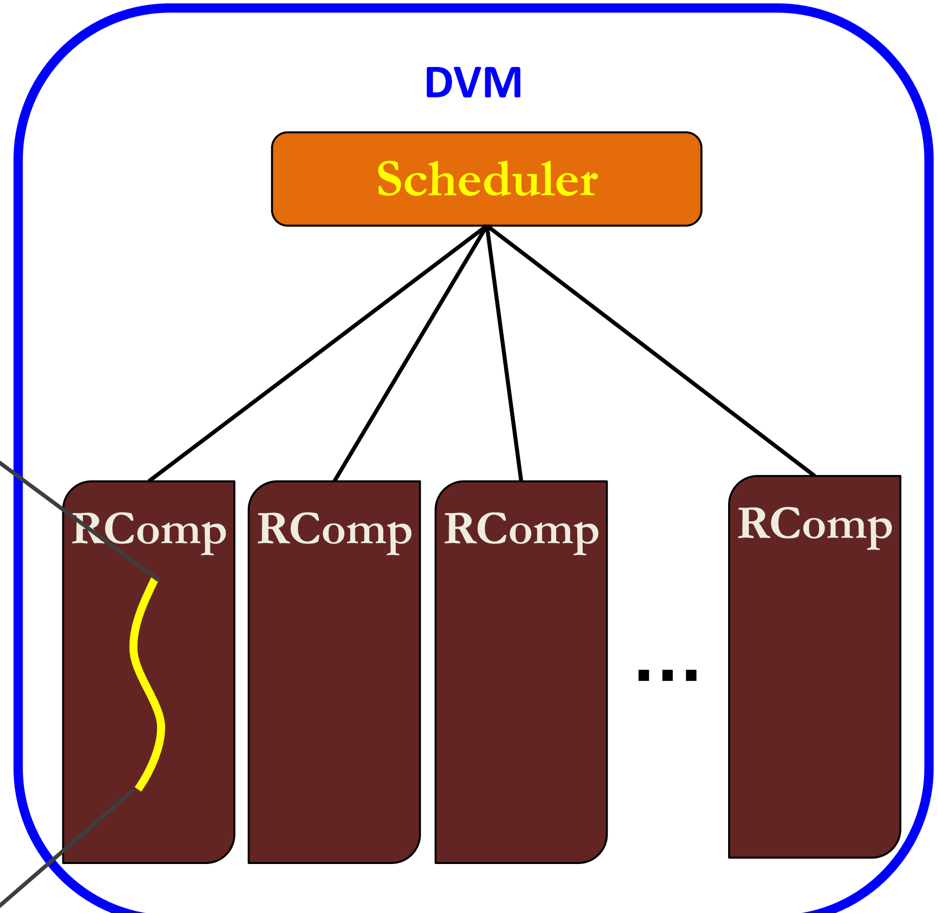


newr stack, heap, watched, fi
newr stack, heap, watched, fi
newr stack, heap, watched, fi
...
newr stack, heap, watched, fi
exit:c

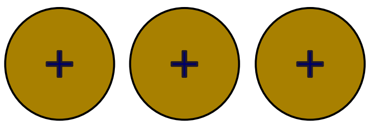
Many-runner parallel execution

+

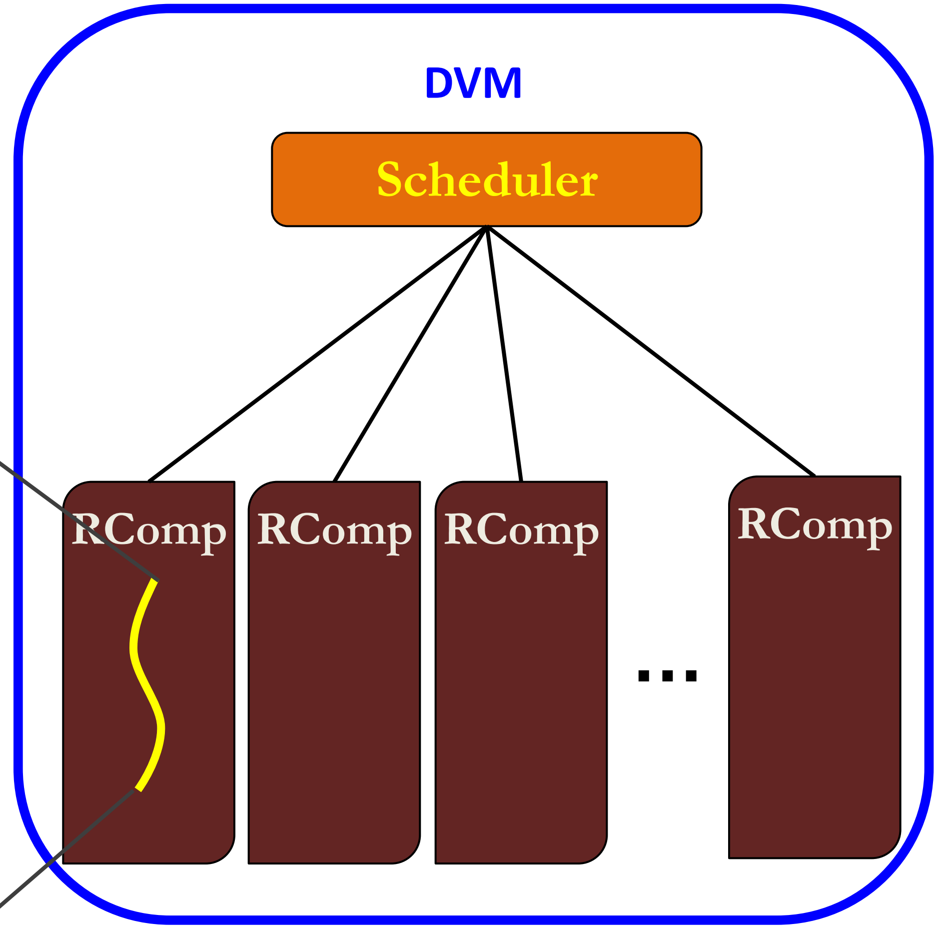
newr stack, heap, watched, fi
newr stack, heap, watched, fi
newr stack, heap, watched, fi
...
newr stack, heap, watched, fi
exit:c



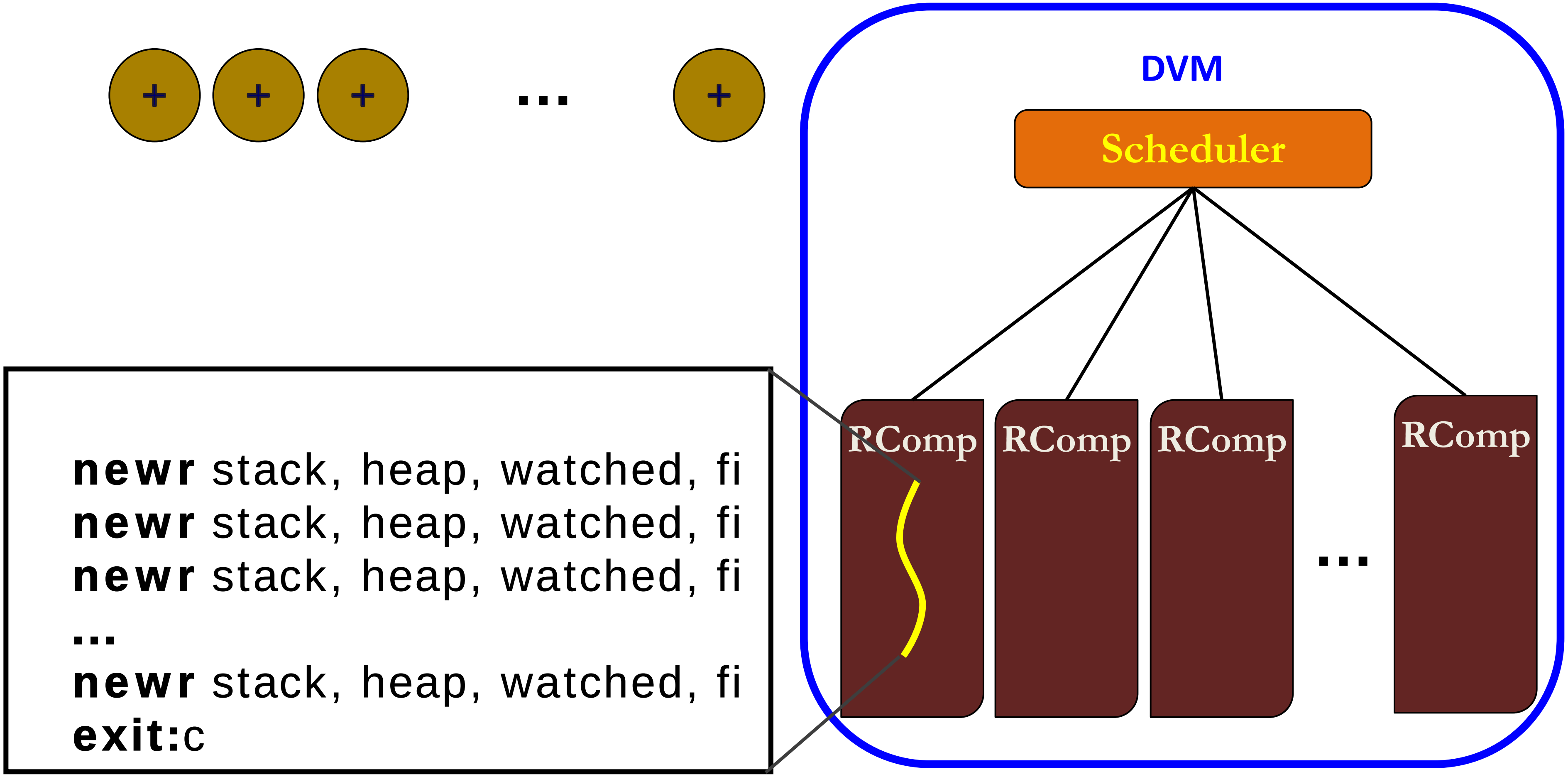
Many-runner parallel execution



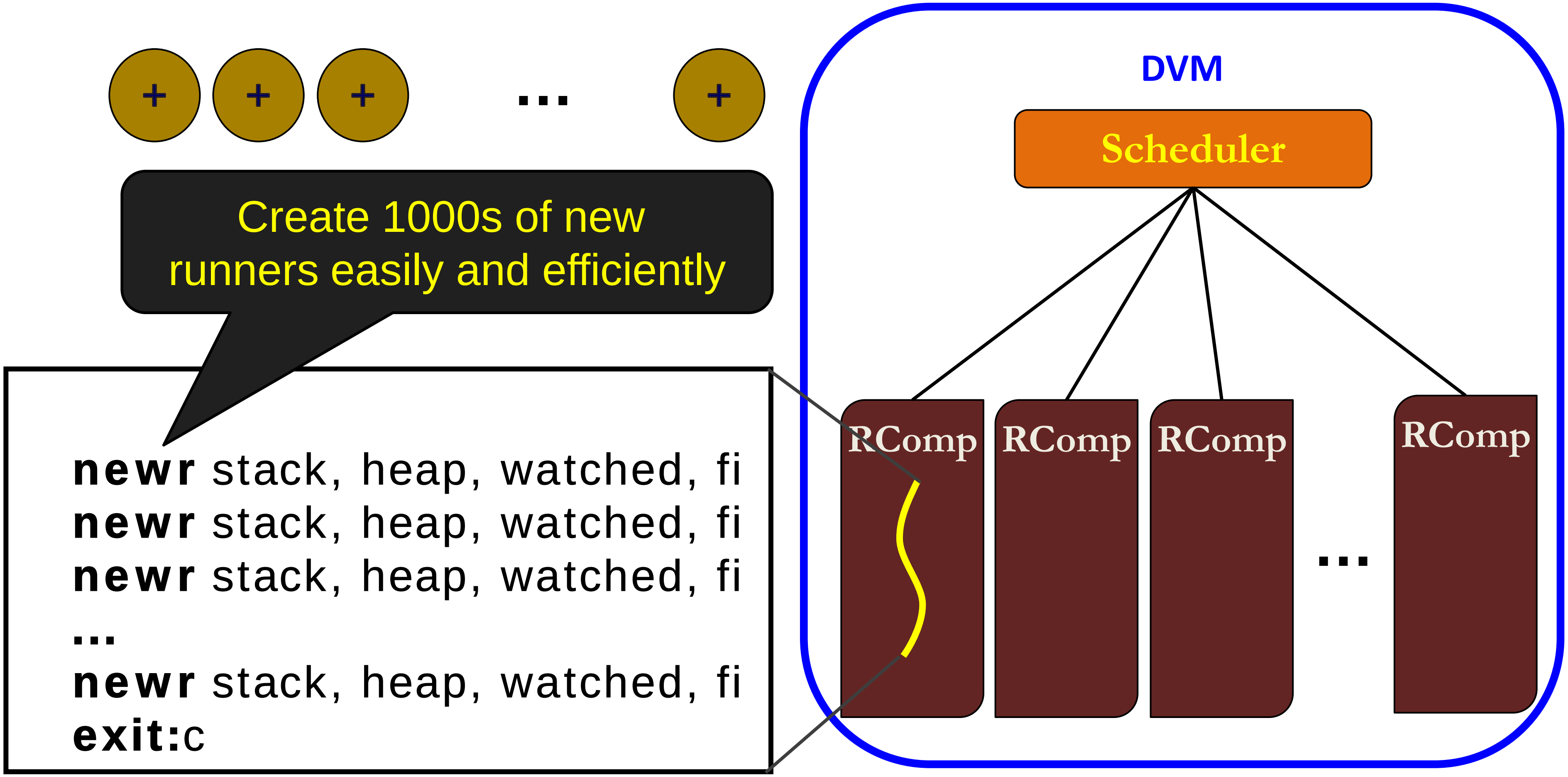
newr stack, heap, watched, fi
newr stack, heap, watched, fi
newr stack, heap, watched, fi
...
newr stack, heap, watched, fi
exit:c



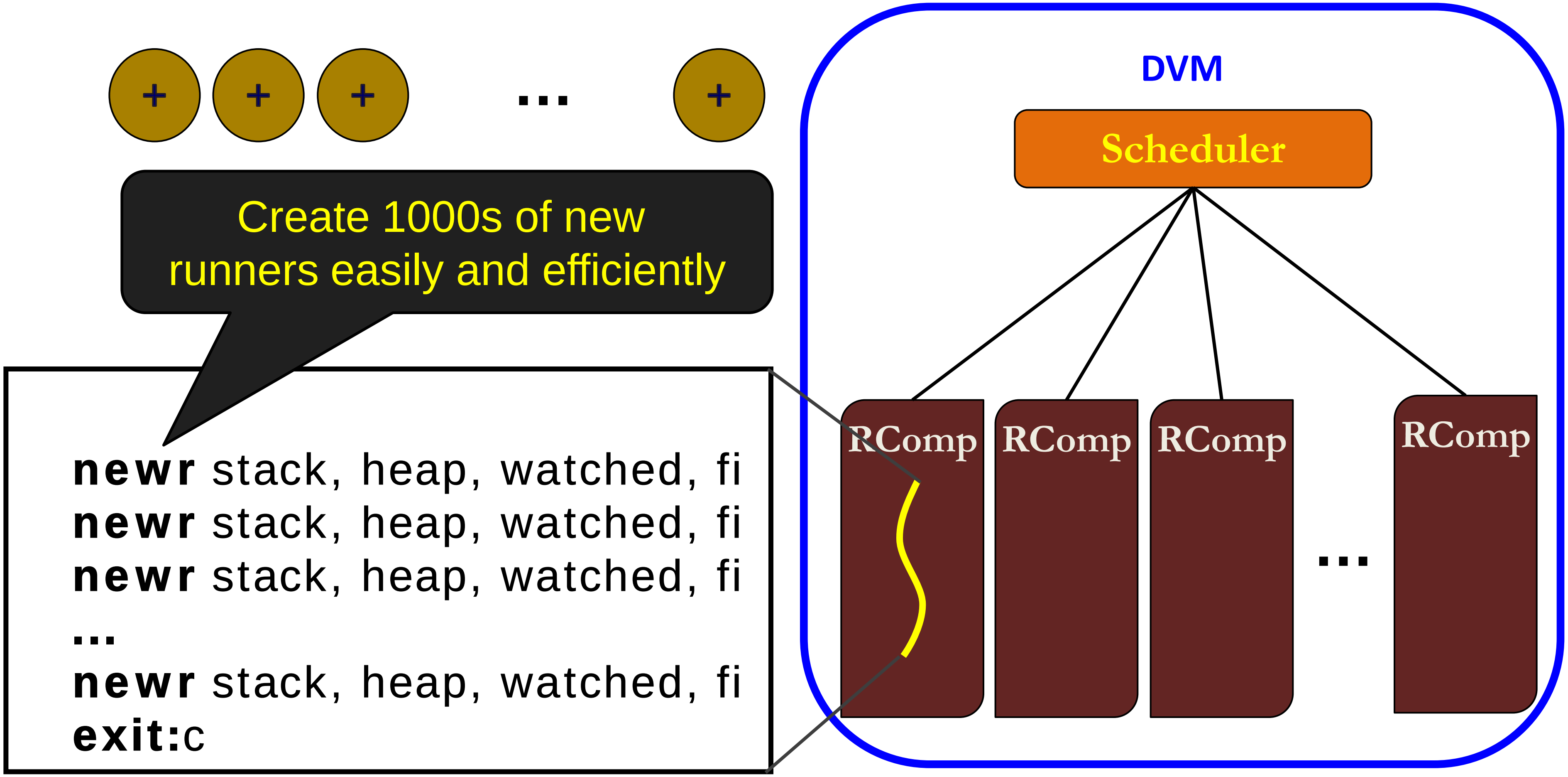
Many-runner parallel execution



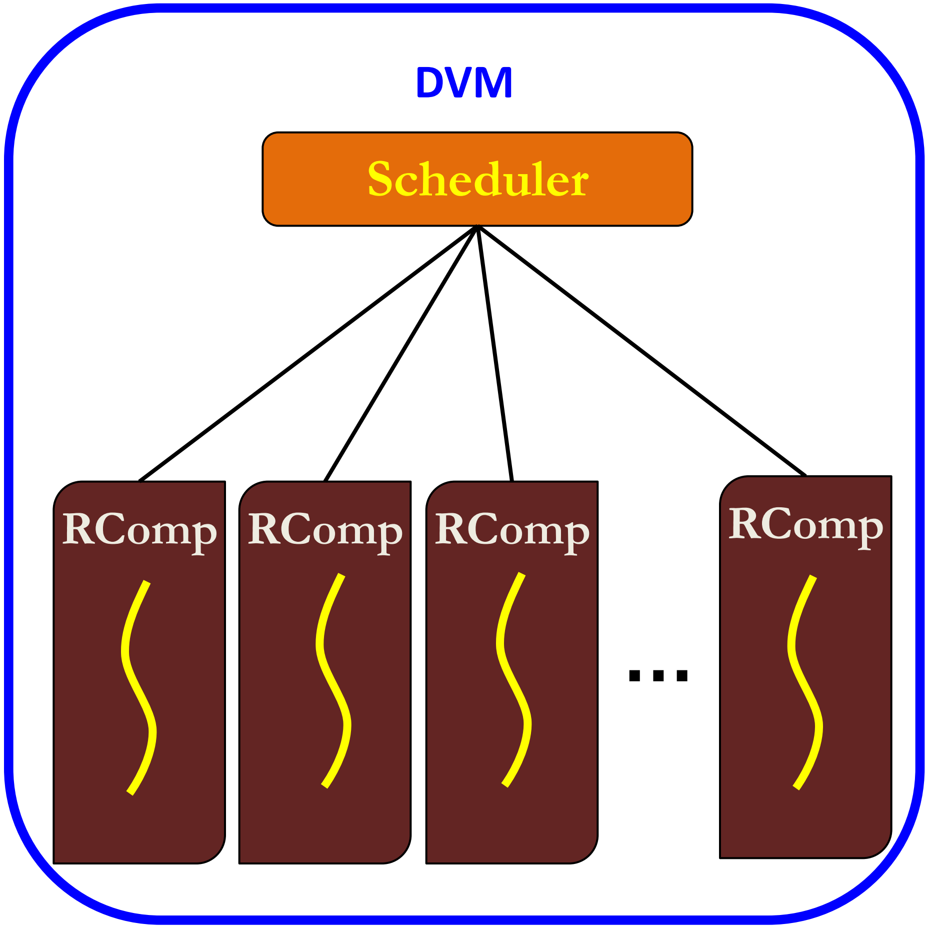
Many-runner parallel execution



Many-runner parallel execution



Many-runner parallel execution



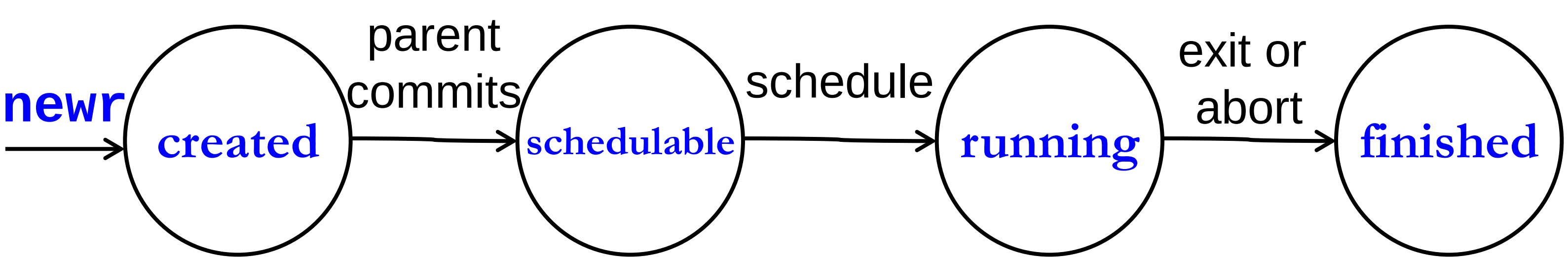
Task dependency

- Task dependency control is a key issue in concurrent program execution

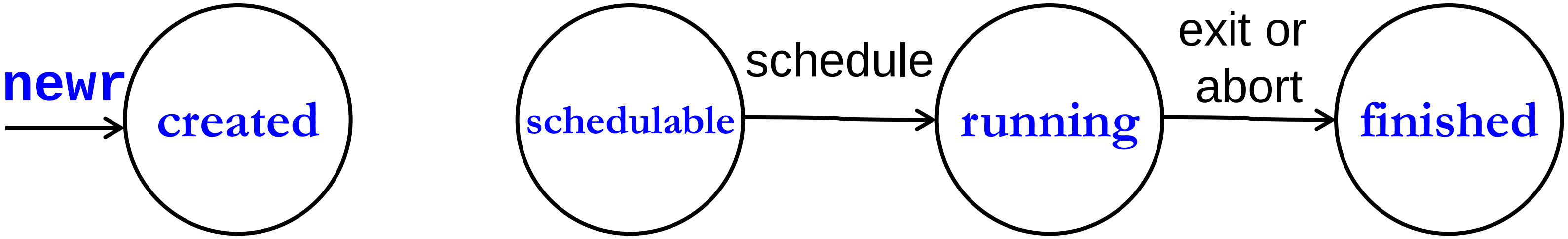
- X10 – synchronization mechanisms
  - Need to synchronize concurrent execution
- MapReduce – Restricted programming model
- Dryad – DAG-based
  - Non-trivial burden in programming
  - Automatic DAG generation only implemented for certain high- level languages

Watcher

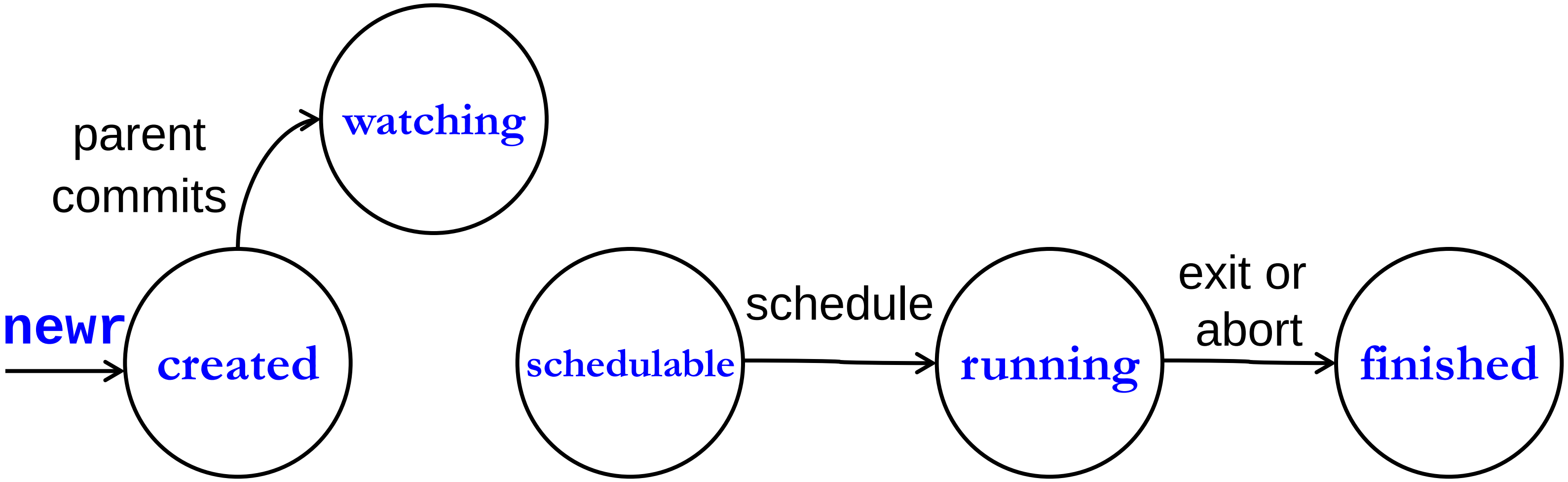
Watcher


newr
created
parent
commits
schedulable
schedule
running
exit or
abort
finished

Watcher

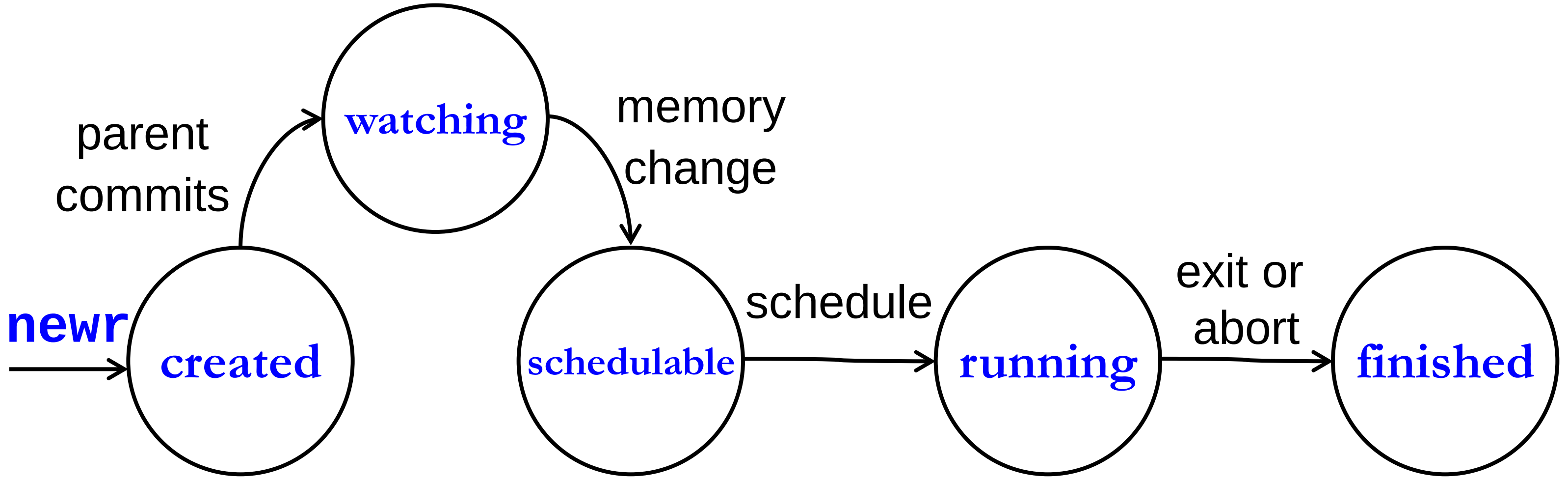

newr
created
schedulable
schedule
running
exit or
abort
finished

Watcher



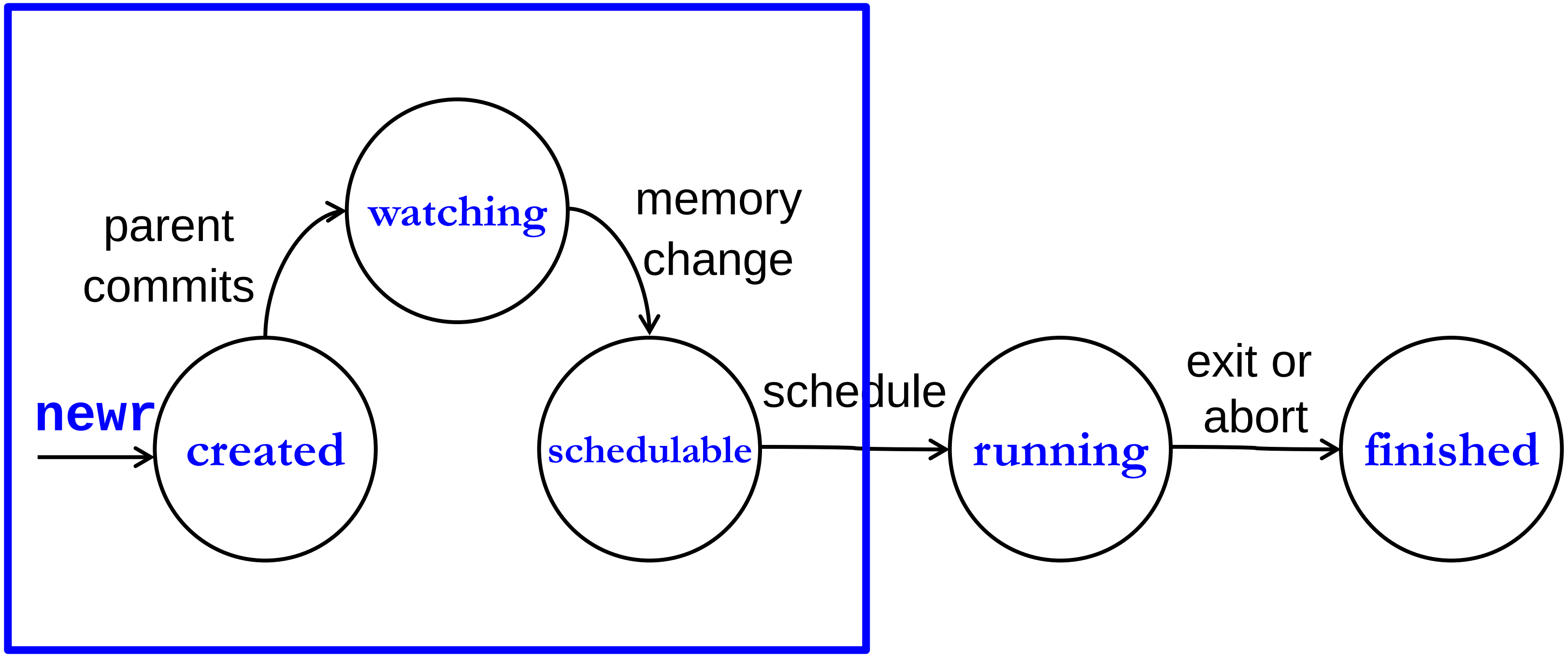
watching
parent
commits
newr
created
schedulable
schedule
running
exit or
abort
finished

Watcher



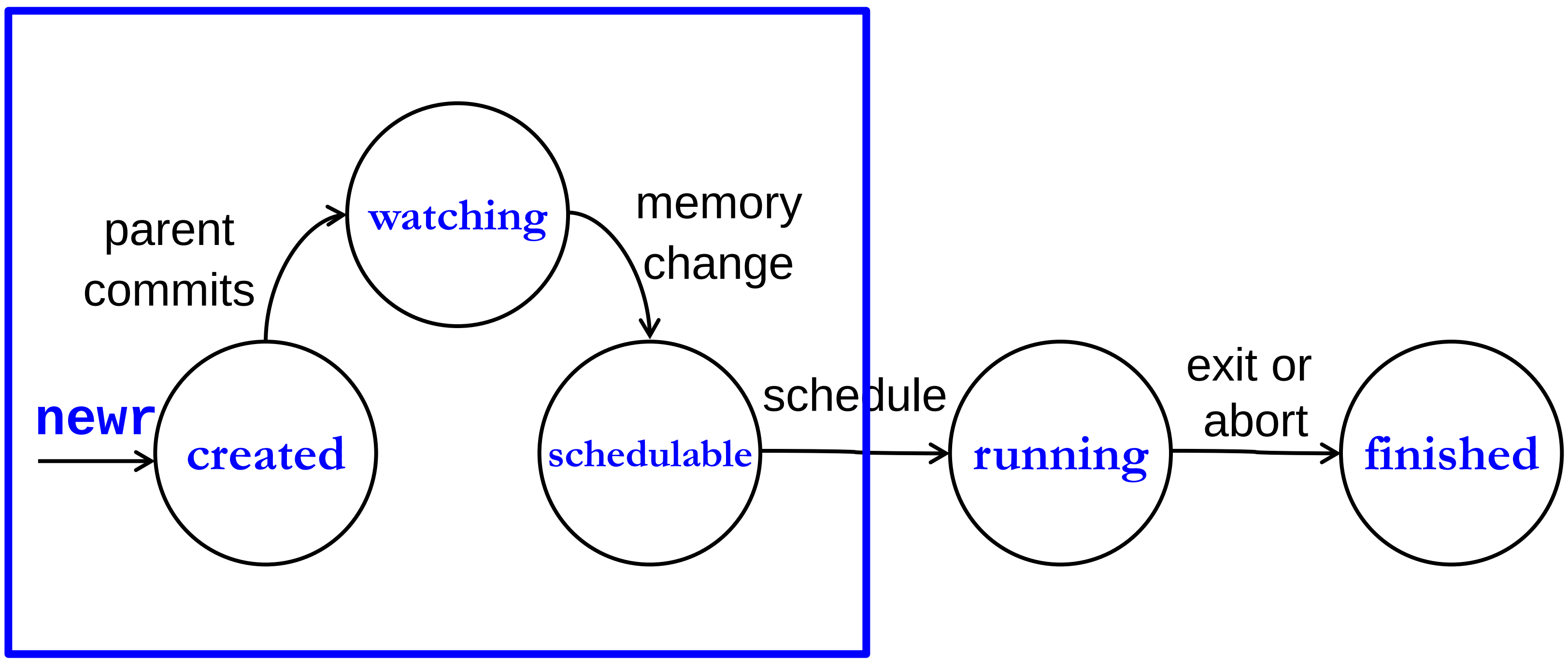
watching
parent
commits
memory
change
newr
created
schedulable
schedule
running
exit or
abort
finished

Watcher

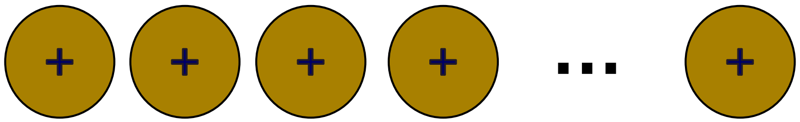

parent
commits
watching
memory
change
newr
created
schedulable
schedule
running
exit or
abort
finished

Watcher

- Watcher – explicitly express data dependence
  - Data dependence: “watched ranges” e.g. [0x1000, 0x1010)
  - Flexible way to declare dependence
  - Automatic dependence resolution

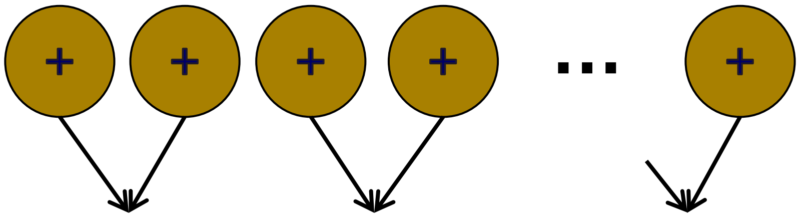


Watcher example



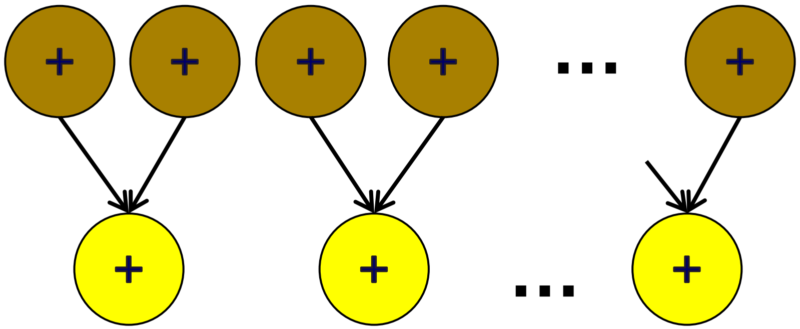
Initial value in 0x1000 and 0x1008 is 0

Watcher example



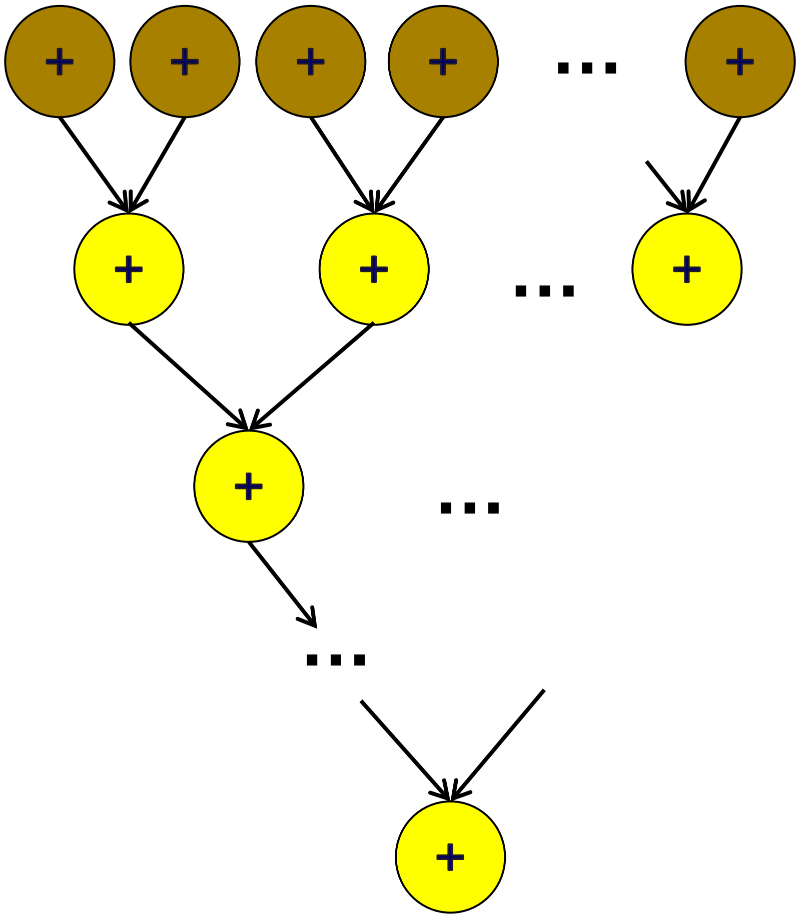
Initial value in 0x1000 and 0x1008 is 0

Watcher example



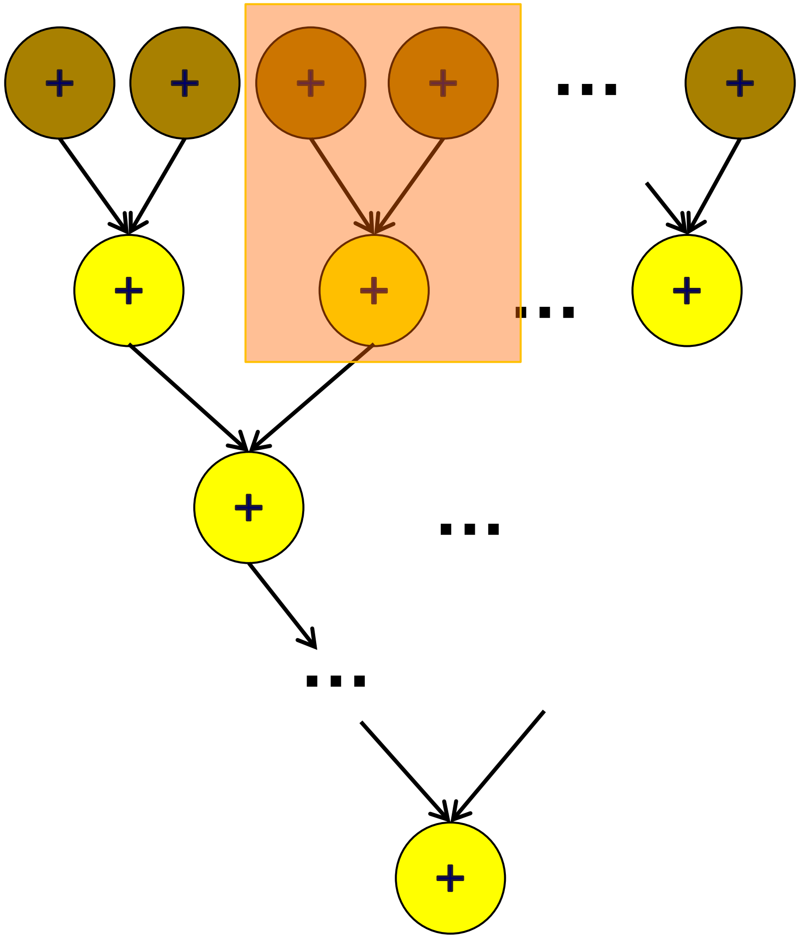
Initial value in 0x1000 and 0x1008 is 0

Watcher example



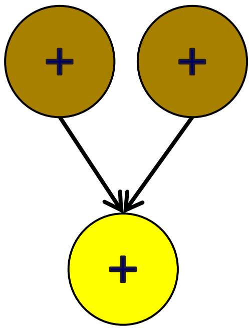
Initial value in 0x1000 and 0x1008 is 0

Watcher example



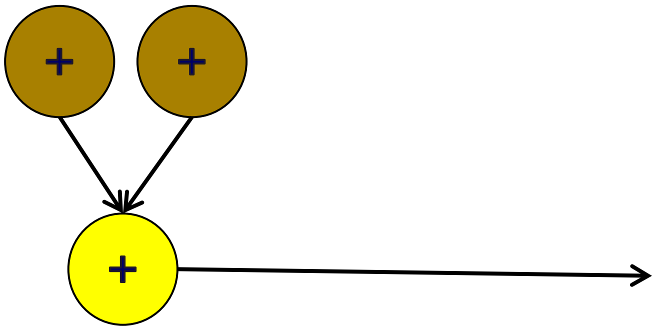
Initial value in 0x1000 and 0x1008 is 0

Watcher example



Initial value in 0x1000 and 0x1008 is 0

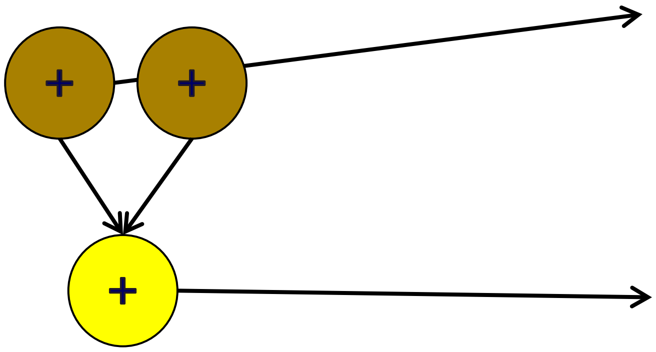
Watcher example



watch shared
memory range
[0x1000, 0x1010)

Initial value in 0x1000 and 0x1008 is 0

Watcher example

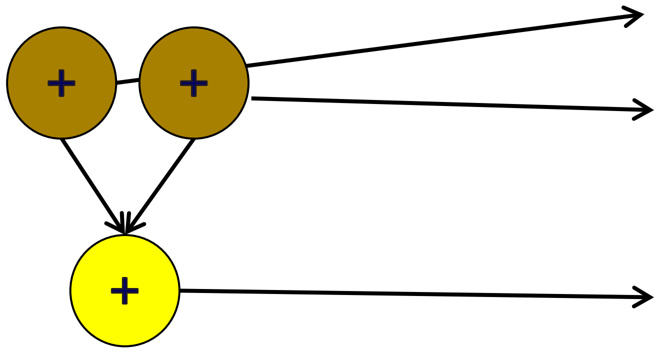


store result in 0x1000

watch shared memory range [0x1000, 0x1010)

Initial value in 0x1000 and 0x1008 is 0

Watcher example



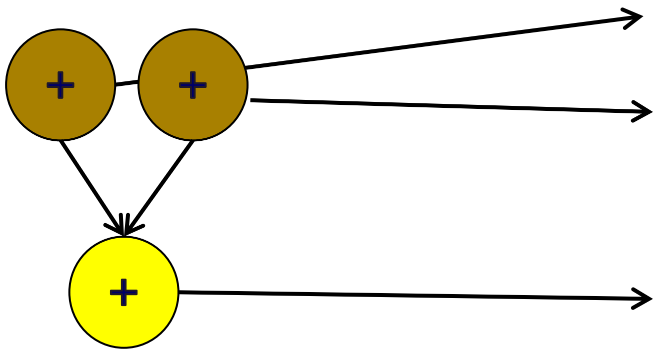
store result in 0x1000

store result in 0x1008

watch shared
memory range
[0x1000, 0x1010)

Initial value in 0x1000 and 0x1008 is 0

Watcher example



store result in 0x1000

store result in 0x1008

watch shared memory range [0x1000, 0x1010)

```
if (*((long*)0x1000) != 0 &&
    *((long*)0x1008) != 0) {
    // add the sum produced by two
    // runners together
} else {
    // create itself and keep watching
```

Initial value in 0x1000 and 0x1008 is 0

Talk outline

- Motivation
- System design
- Evaluation

Implementation and evaluation

- Emulate DISA on x86-64
  - Dynamic binary translation
- Implement DVM
  - CCMR – a research testbed
  - An industrial testbed
  - Amazon Elastic Compute Cloud (EC2)
- Microbenchmarks, prime-checker and *k*-means clustering
  - Compare with Xen, VMware, Hadoop and X10

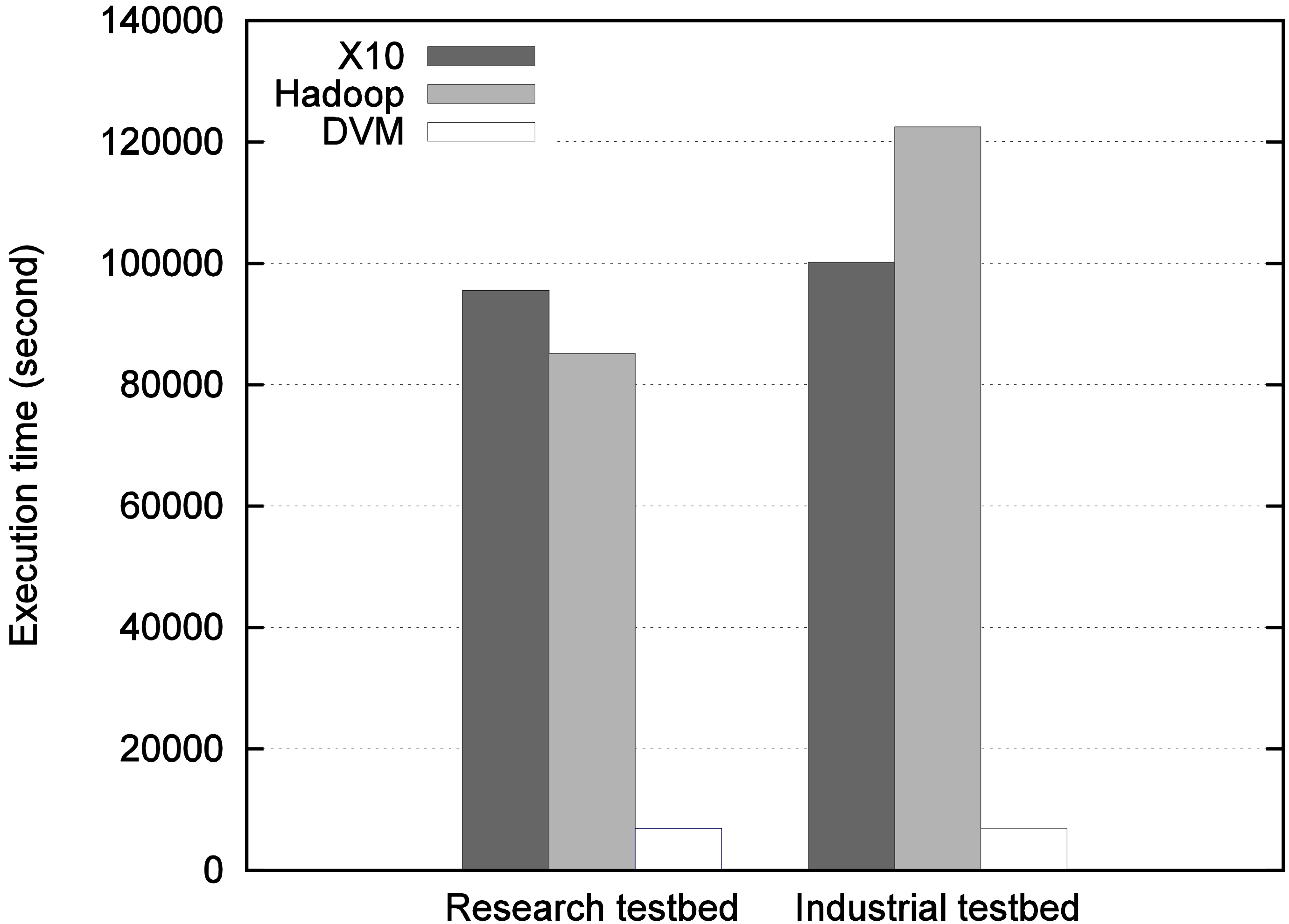
Goals of DVM:
General, scalable, efficient, portable, easy-to-program

Implementation and evaluation

- Emulate DISA on x86-64
  - Dynamic binary translation
- Implement DVM
  - CCMR – a research testbed
  - An industrial testbed
  - Amazon Elastic Compute Cloud (EC2)
- Microbenchmarks, prime-checker and *k*-means clustering
  - Compare with Xen, VMware, Hadoop and X10

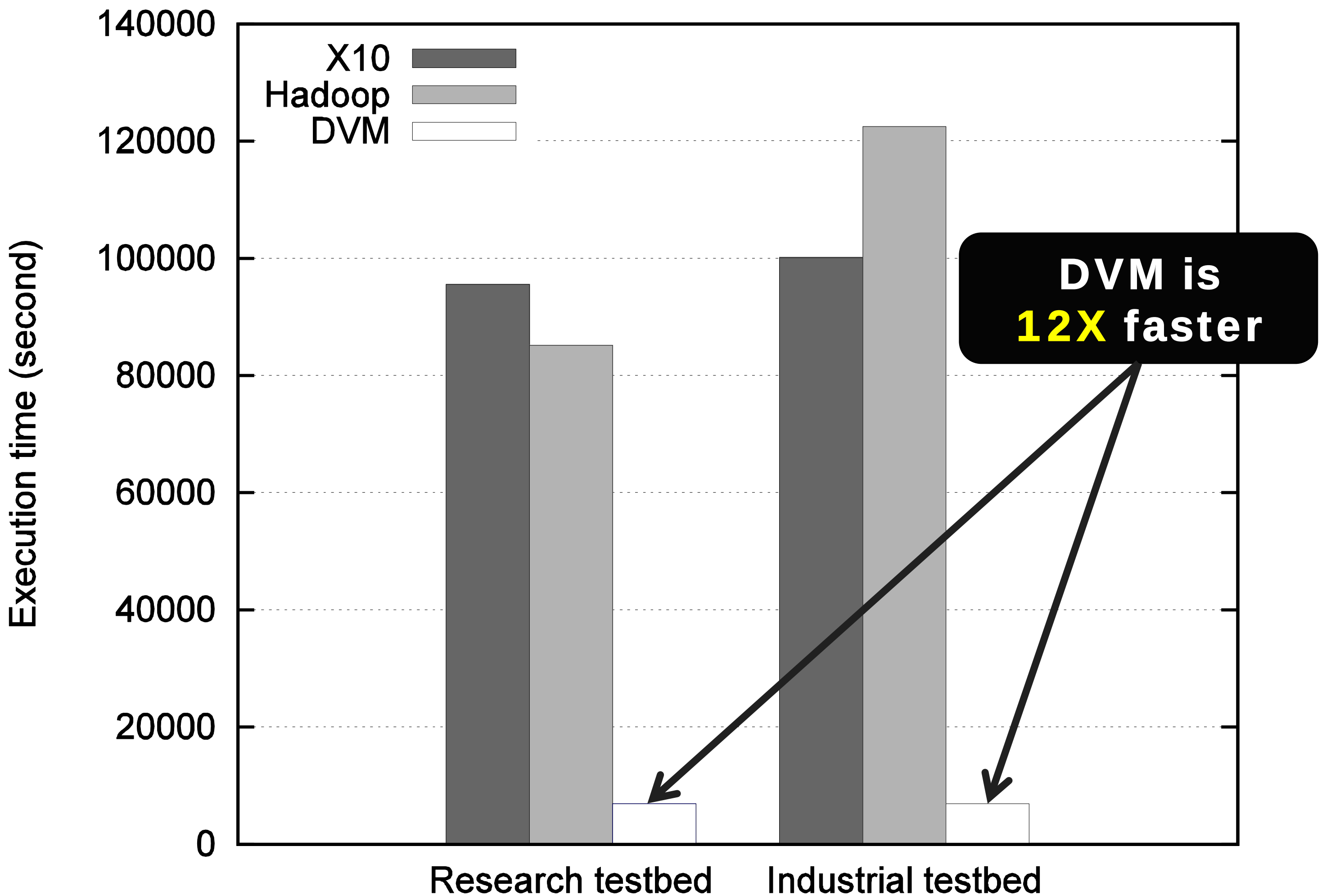
Goals of DVM:
General, scalable, efficient, portable, easy-to-program

Performance comparison – *k*-means on 1 node



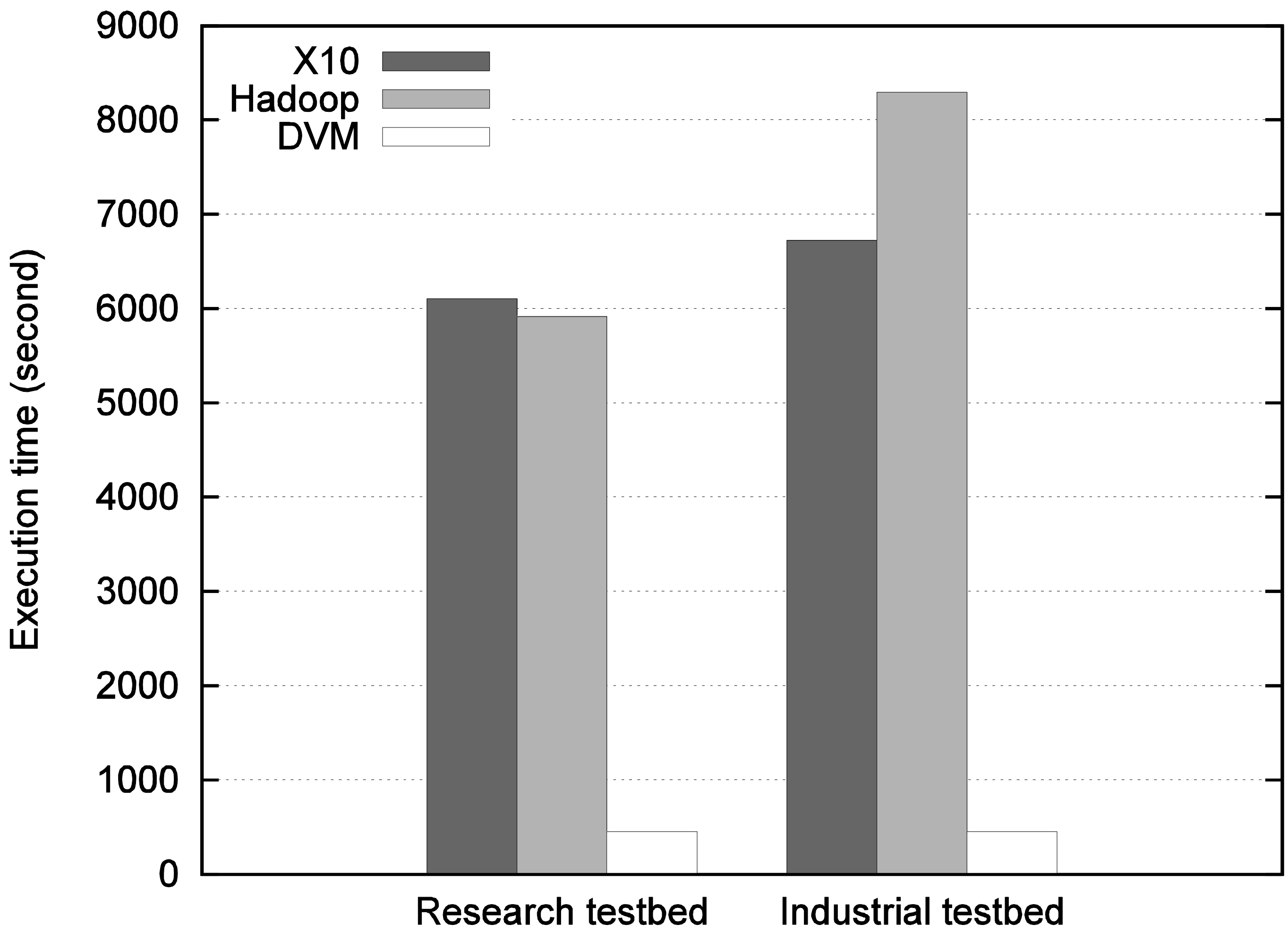
Execution time of *k*-means on 1 working node
R: research testbed. I: industrial testbed.

Performance comparison – *k*-means on 1 node



Execution time of *k*-means on 1 working node
R: research testbed. I: industrial testbed.

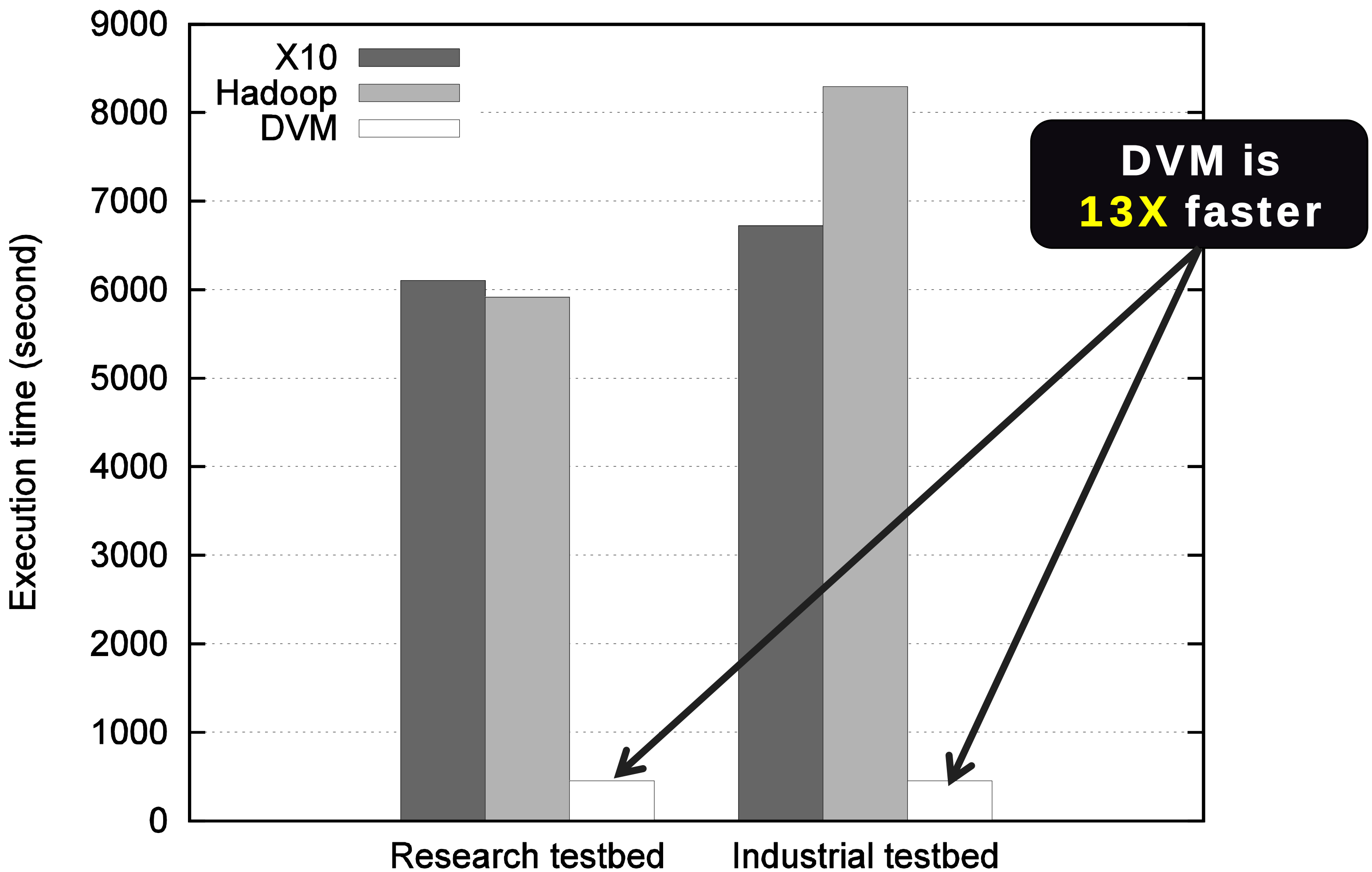
Performance comparison – *k*-means on 16 nodes



Execution time of *k*-means on 16 working nodes

General, scalable, efficient, portable, easy-to-program

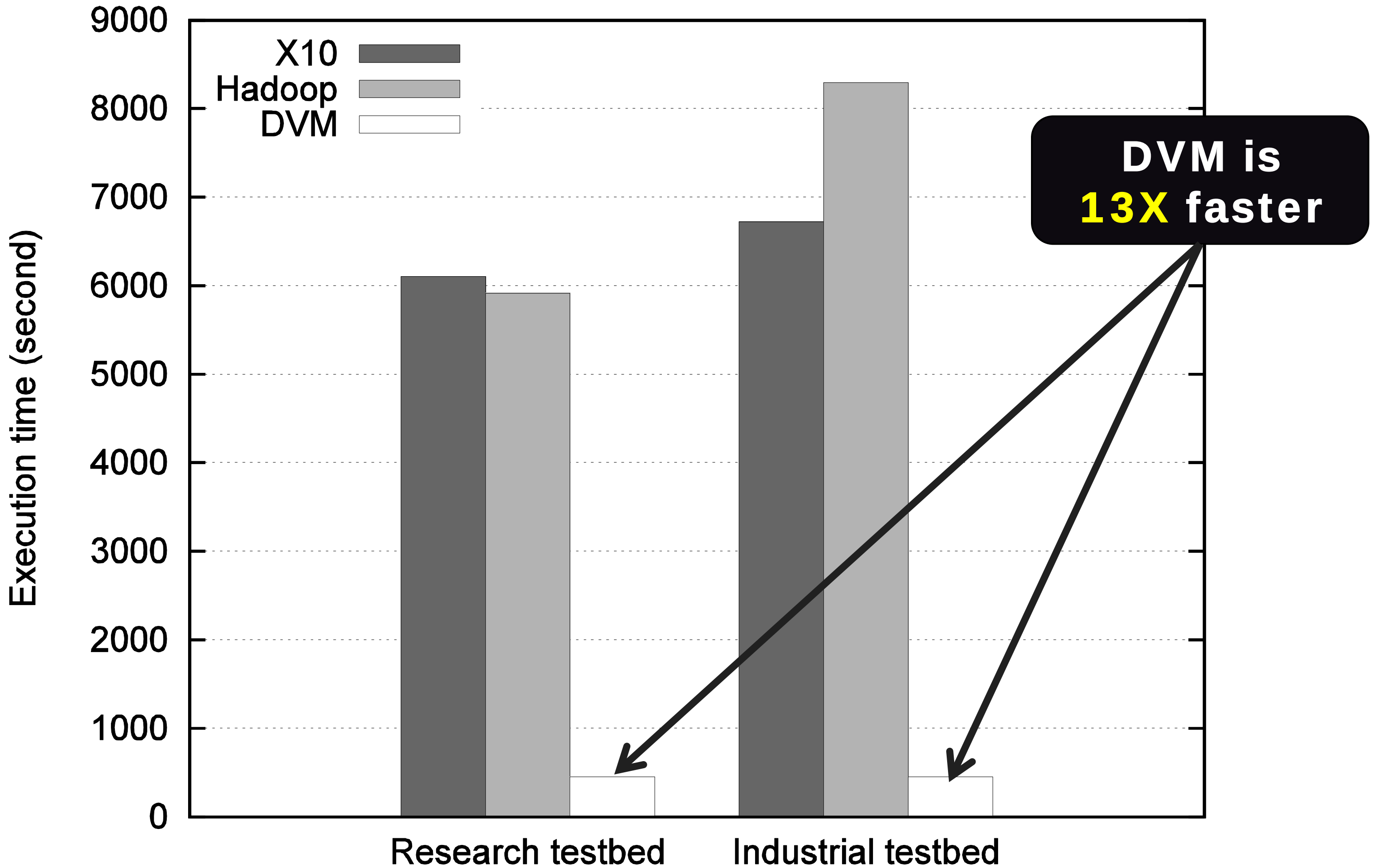
Performance comparison – *k*-means on 16 nodes



Execution time of *k*-means on 16 working nodes

General, scalable, efficient, portable, easy-to-program

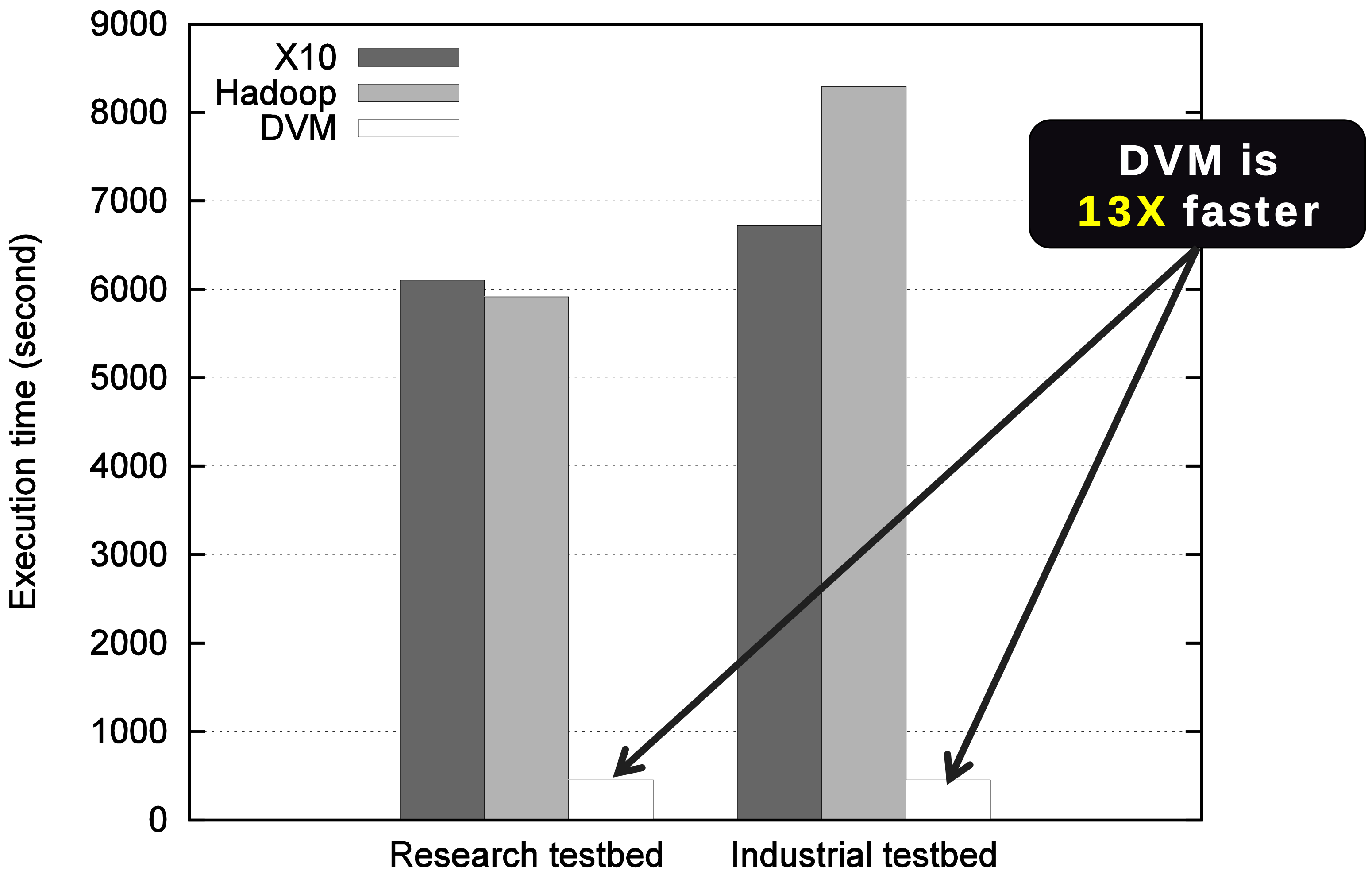
Performance comparison – *k*-means on 16 nodes



Execution time of *k*-means on 16 working nodes

General, scalable, efficient, portable, easy-to-program

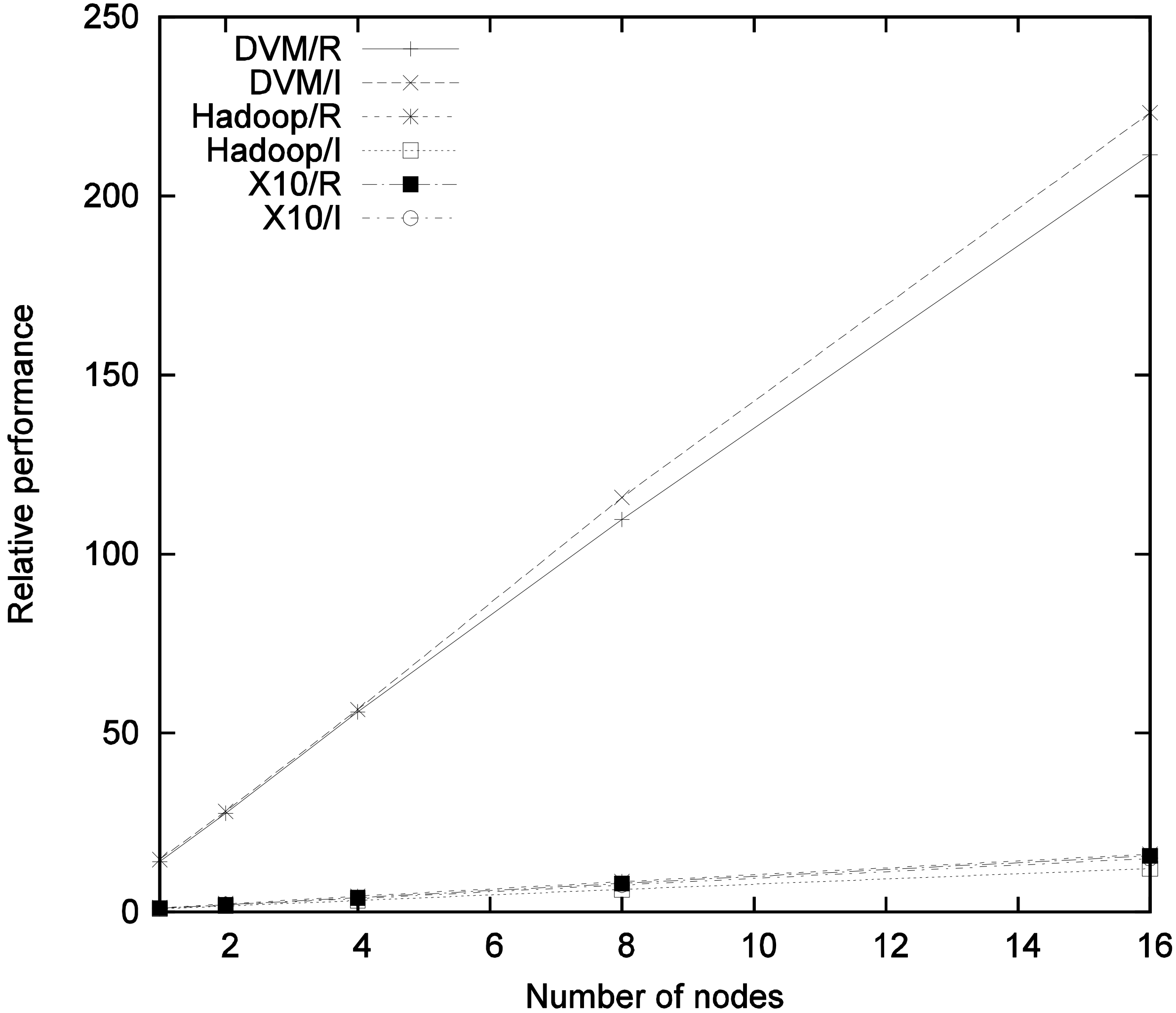
Performance comparison – *k*-means on 16 nodes



Execution time of *k*-means on 16 working nodes

General, scalable ✓, efficient ✓, portable, easy-to-program

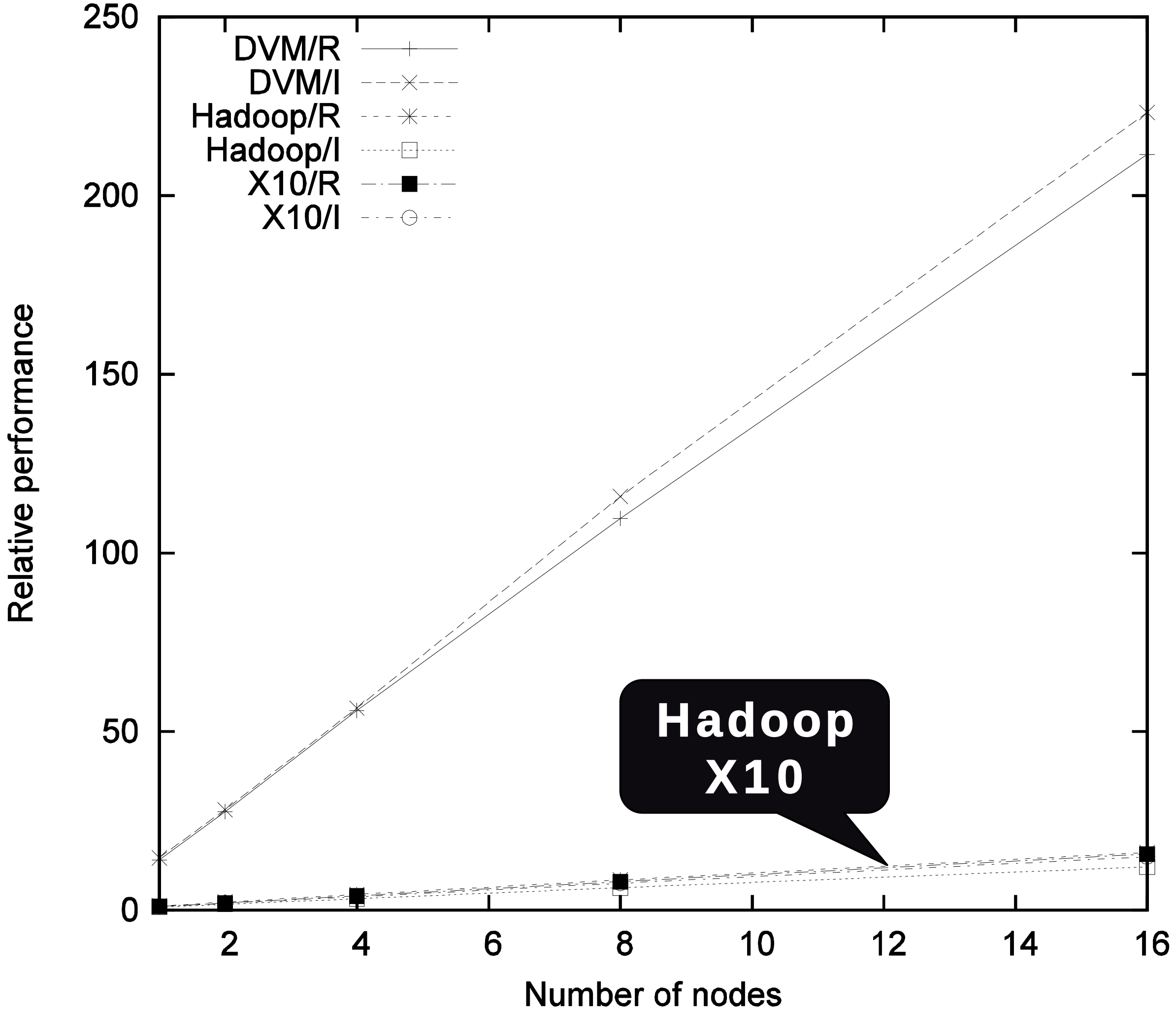
Performance comparison – relative performance of *k*-means



Relative performance of *k*-means as the number of working nodes grows

General, scalable, efficient, portable, easy-to-program

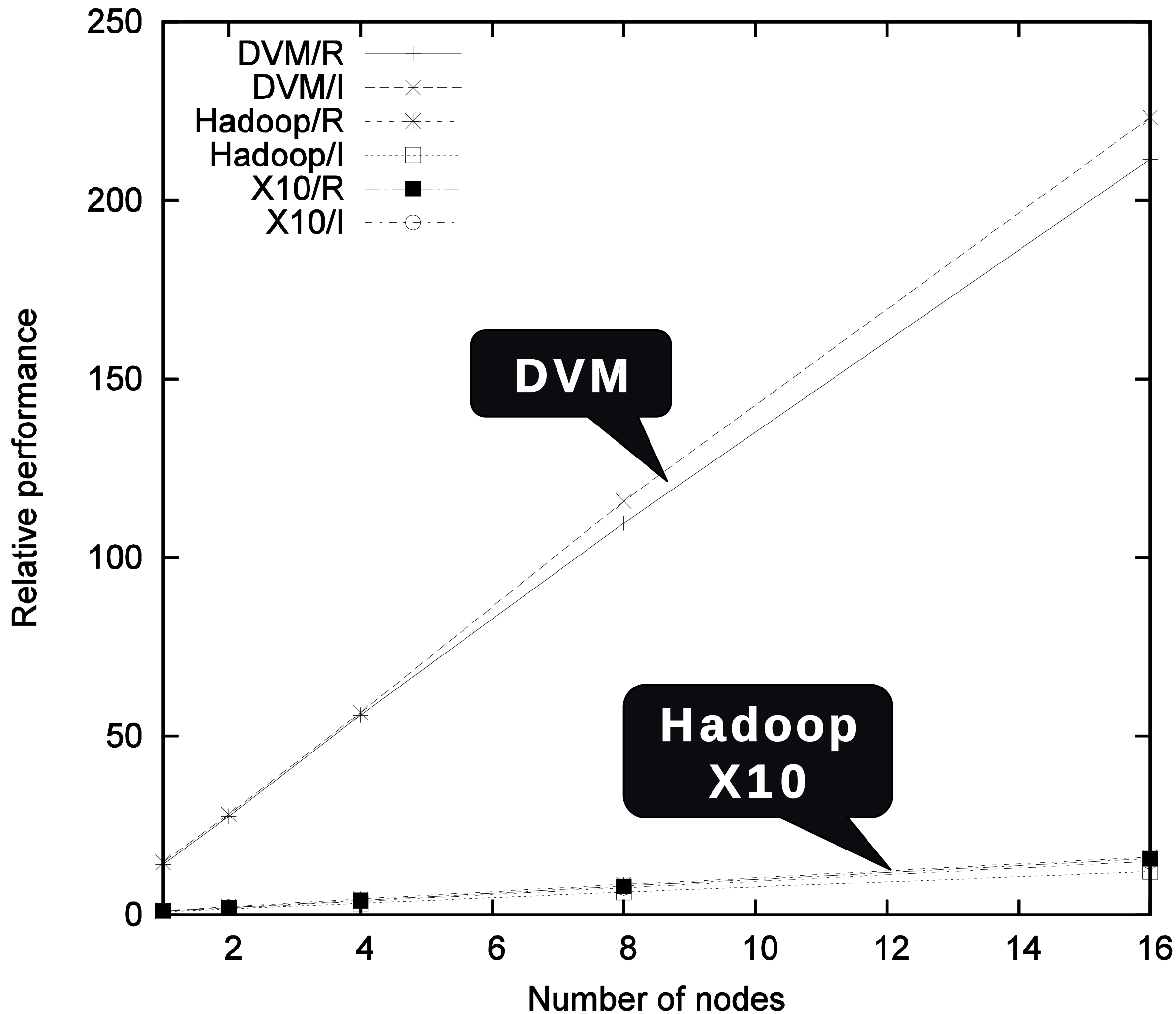
Performance comparison – relative performance of *k*-means



Relative performance of *k*-means as the number of working nodes grows

General, scalable, efficient, portable, easy-to-program

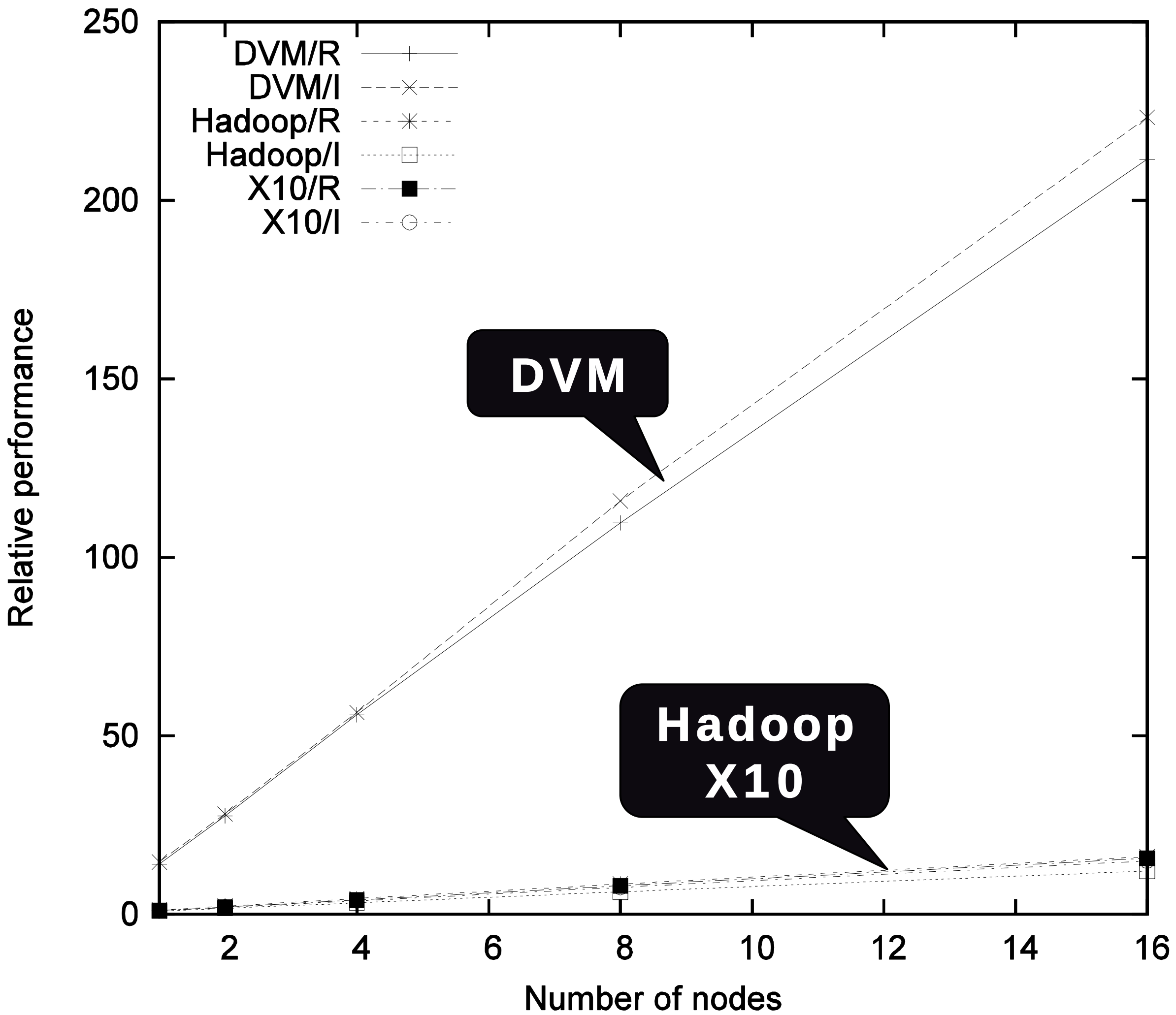
Performance comparison – relative performance of *k*-means



Relative performance of *k*-means as the number of working nodes grows

General, scalable, efficient, portable, easy-to-program

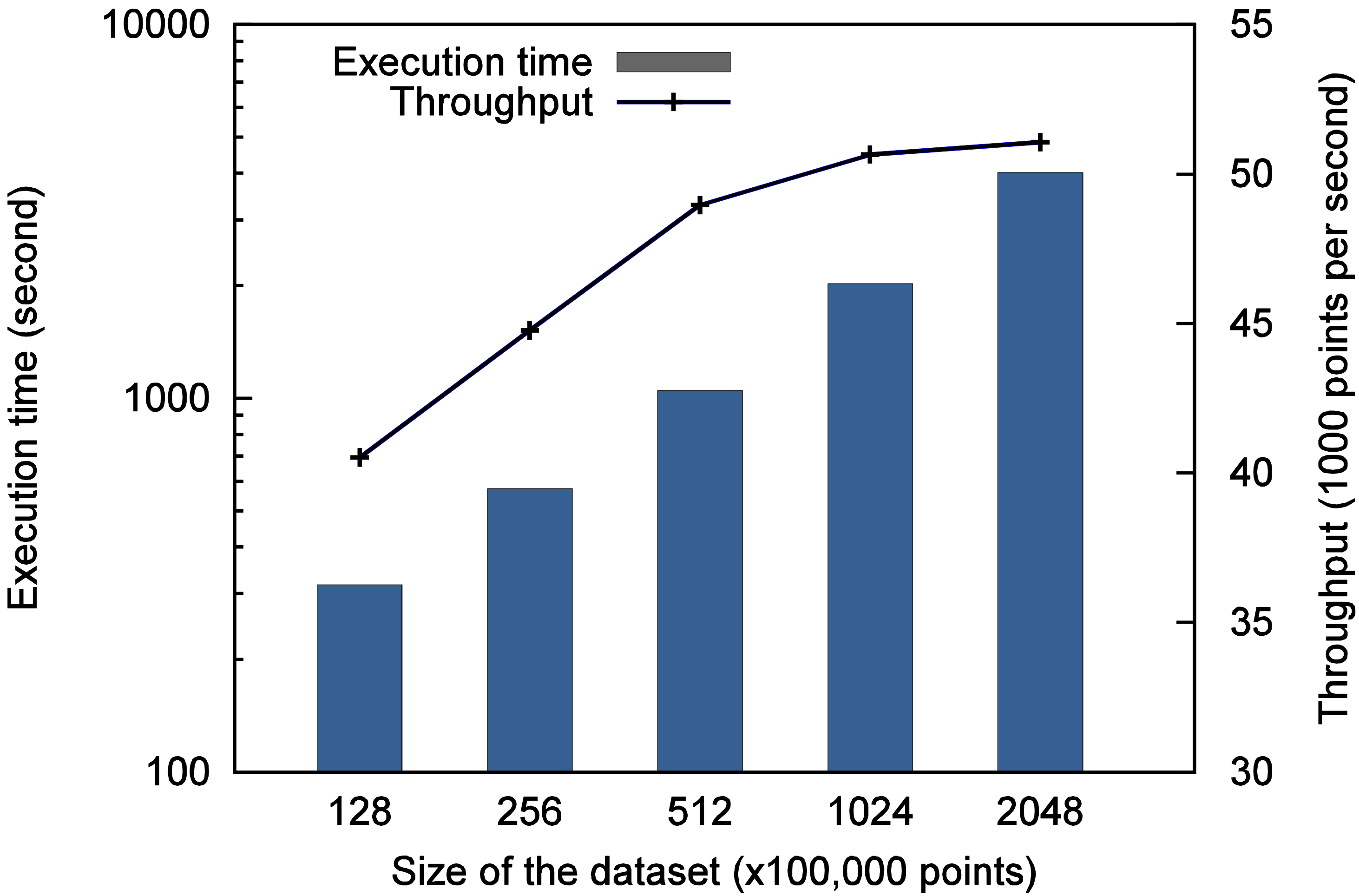
Performance comparison – relative performance of *k*-means



Relative performance of *k*-means as the number of working nodes grows

General, scalable, efficient, portable, easy-to-program

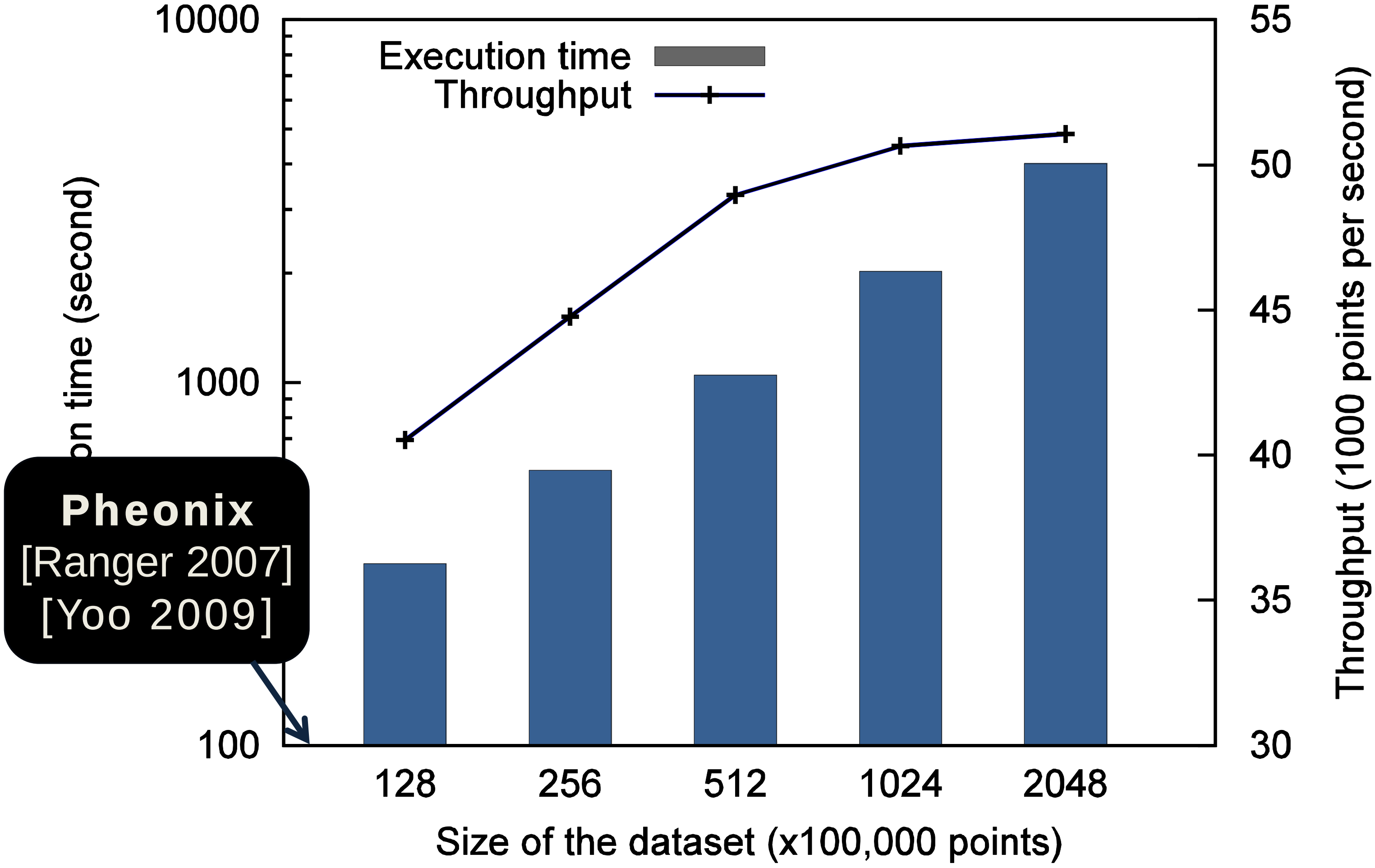
Scalability with data size



Execution time and throughput of *k*-means as the size of dataset grows

General, scalable, efficient, portable, easy-to-program

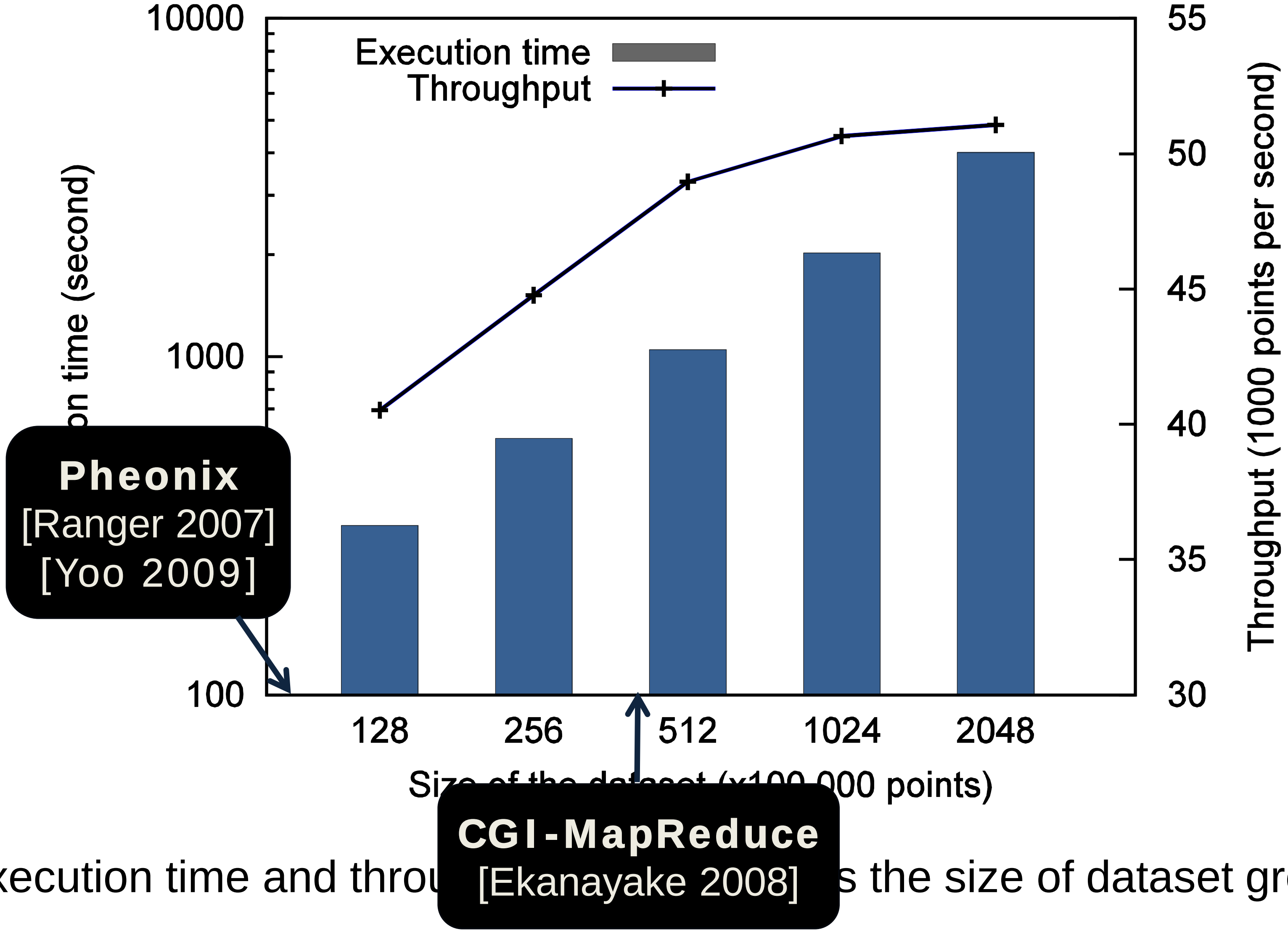
Scalability with data size



Execution time and throughput of *k*-means as the size of dataset grows

General, scalable, efficient, portable, easy-to-program

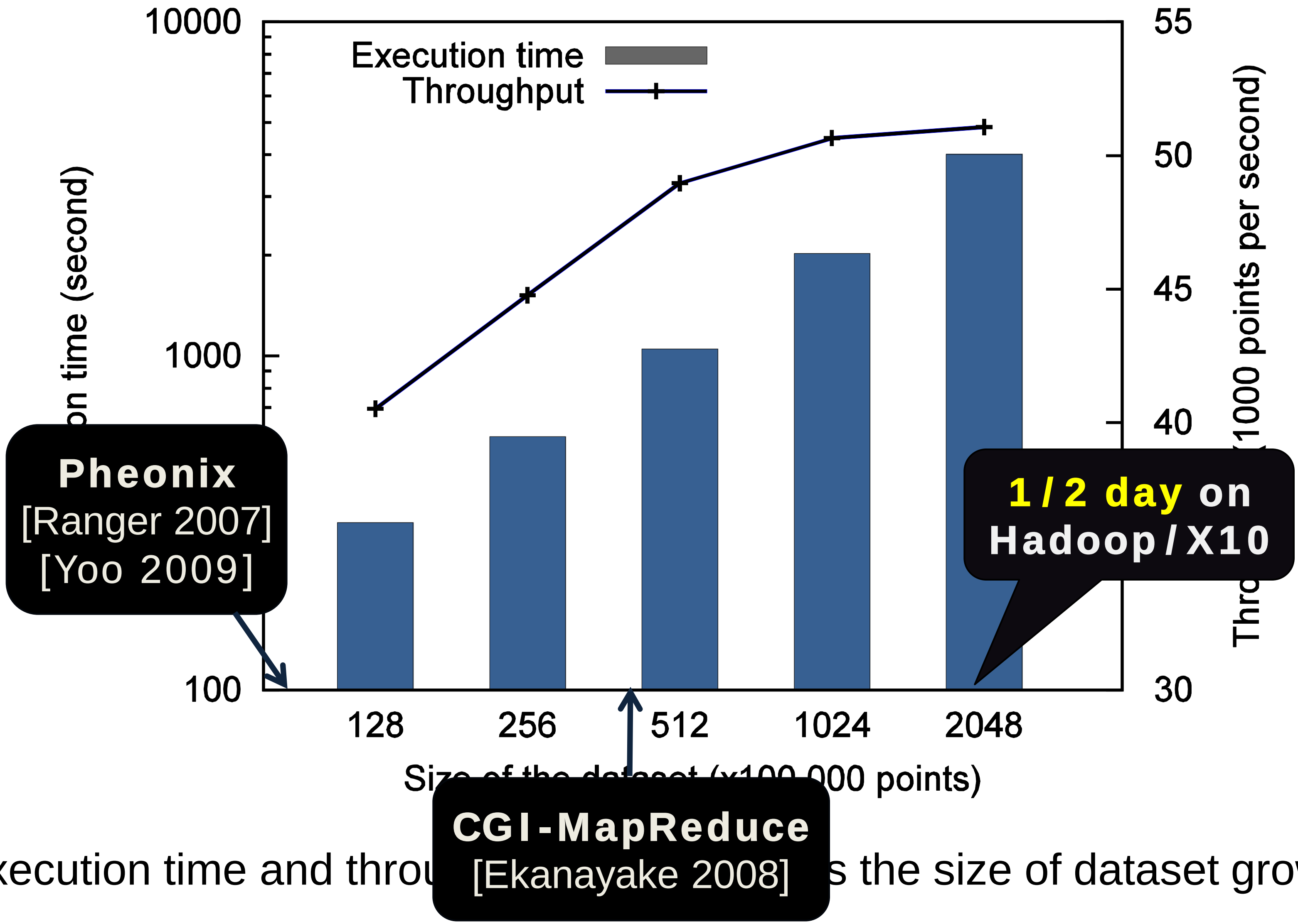
Scalability with data size



Execution time and throu... s the size of dataset grows

General, scalable, efficient, portable, easy-to-program

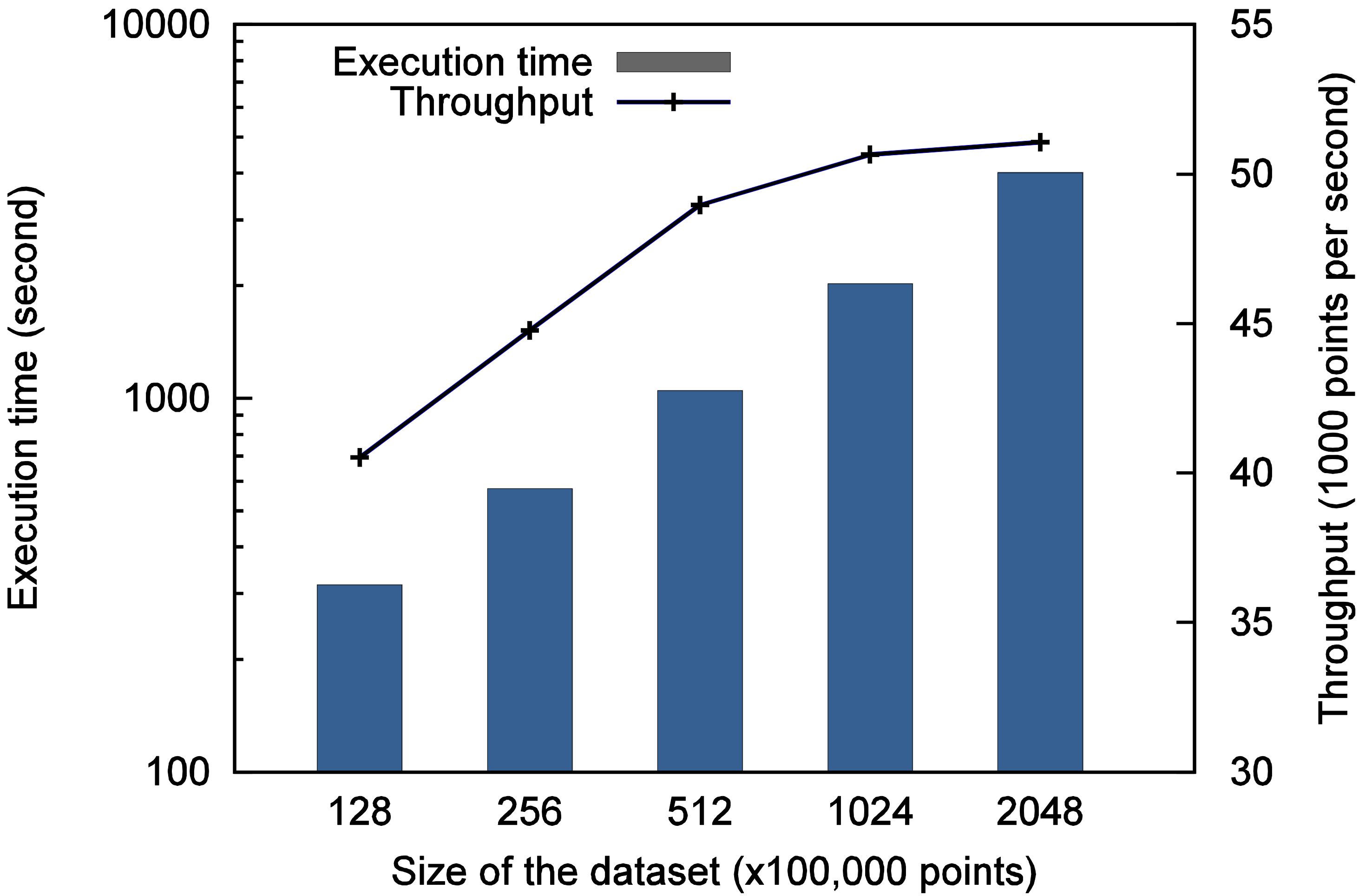
Scalability with data size



Execution time and throu... s the size of dataset grows

General, scalable, efficient, portable, easy-to-program

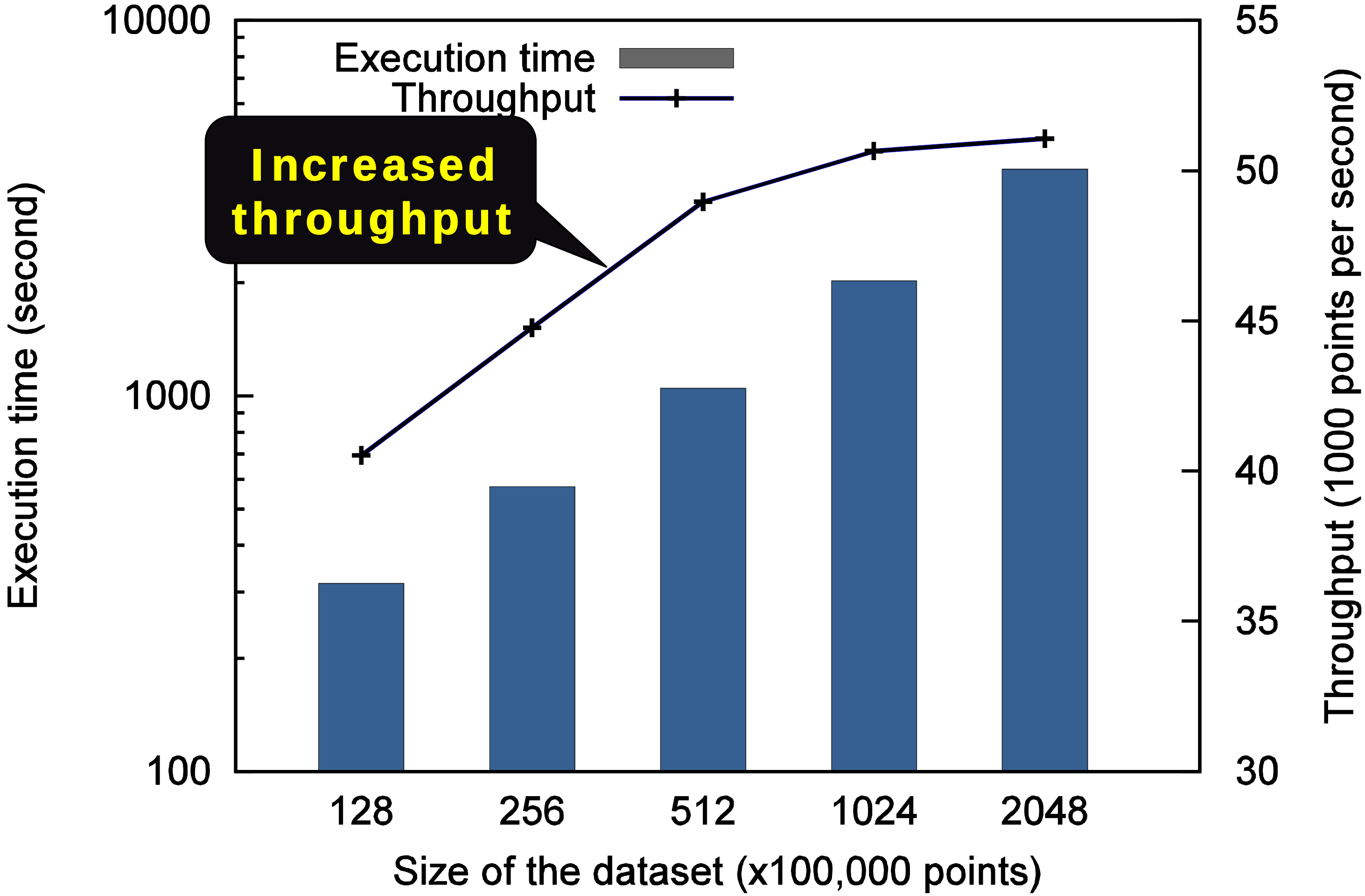
Scalability with data size



Execution time and throughput of *k*-means as the size of dataset grows

General, scalable, efficient, portable, easy-to-program

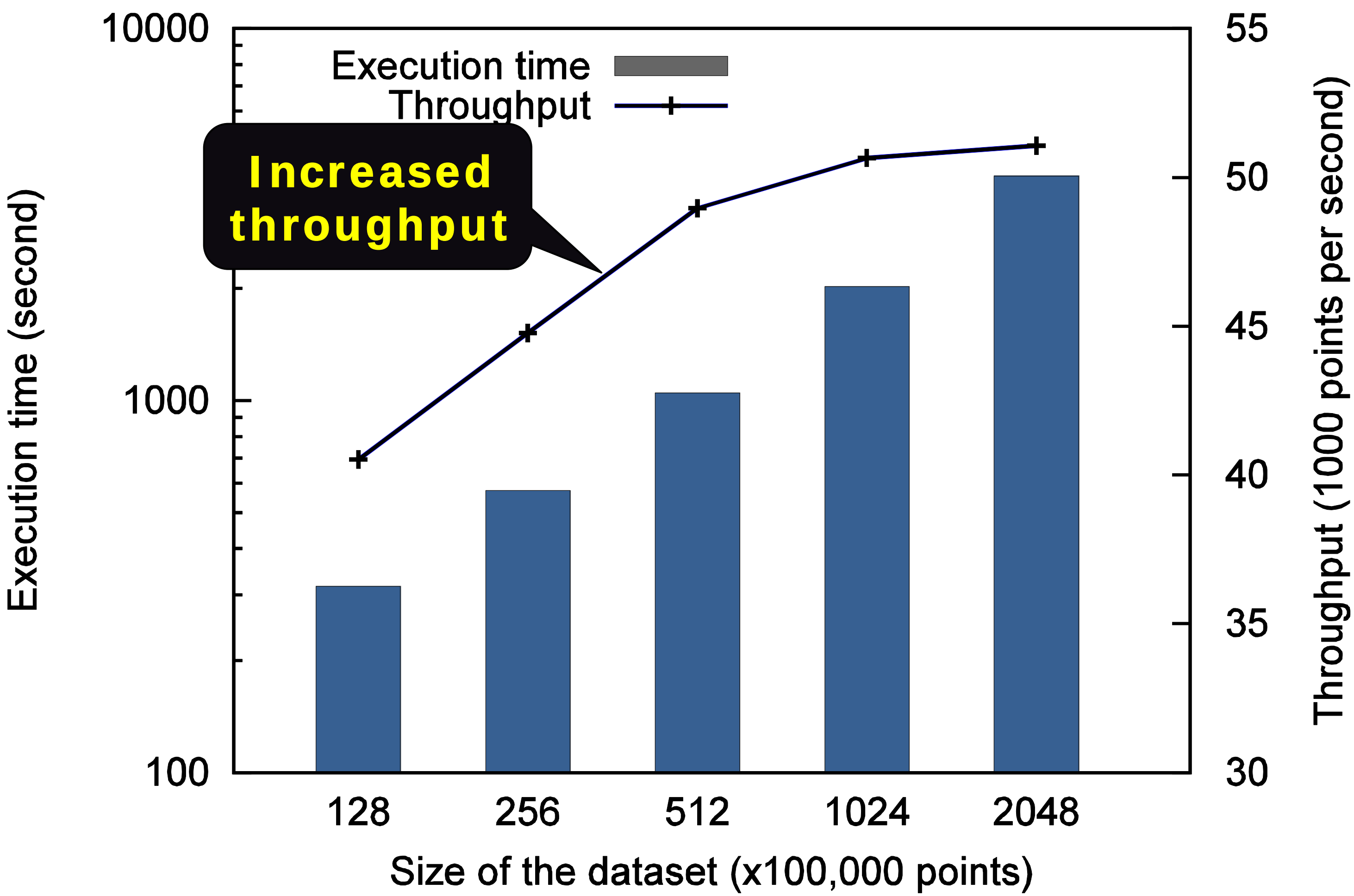
Scalability with data size



Execution time and throughput of *k*-means as the size of dataset grows

General, scalable, efficient, portable, easy-to-program

Scalability with data size



Execution time and throughput of *k*-means as the size of dataset grows

General, scalable, efficient, portable, easy-to-program

Conclusion and future work

- DVM is an approach to unifying computation in a datacenter
  - Illusion of a "big machine" – "The datacenter as a computer"
  - DISA as the programming interface and abstraction of DVM
  - One order of magnitude faster than Hadoop and X10
  - Scales to many compute nodes
- Future work
  - Compiler for programmers, DVM across datacenters, etc.

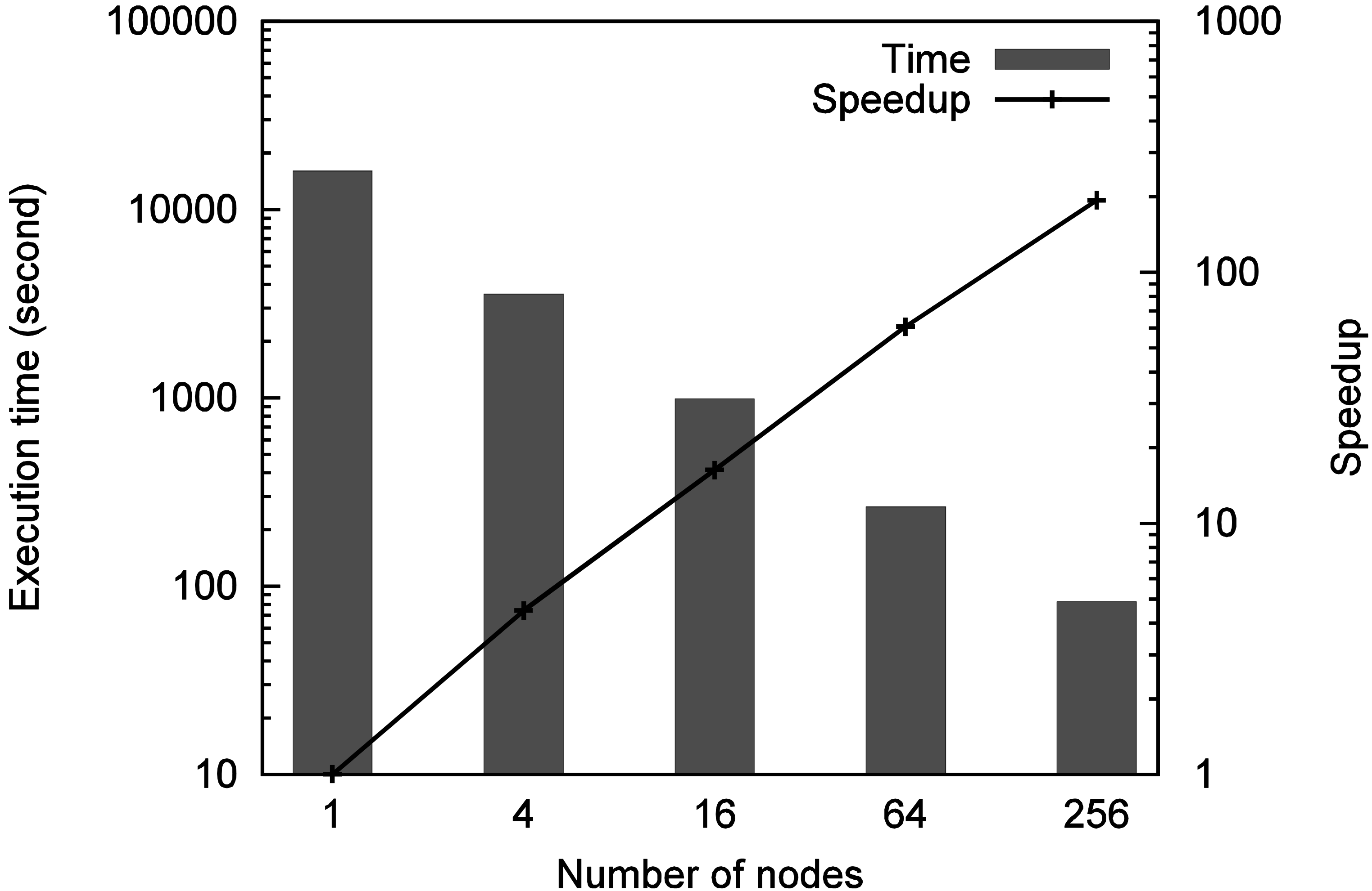
Thank you!

Reference

- **[Dean 2004]** J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *the 6th Conference on Symposium on Operating Systems Design & Implementation*, volume 6, pages 137–150, 2004.
- **[Barroso 2009]** L. Barroso and U. Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis Lectures on Computer Architecture*, 4(1):1–108, 2009.
- **[Ranger 2007]** C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis. Evaluating MapReduce for multi-core and multiprocessor systems. In *Proc. of the 2007 IEEE 13th Intl Symposium on High Performance Computer Architecture*, pages 13–24, 2007.
- **[Yoo 2009]** Richard M. Yoo, Anthony Romano, and Christos Kozyrakis. Phoenix Rebirth: Scalable MapReduce on a Large-Scale Shared-Memory System", *In Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC),* pp. 198-207, 2009.
- **[Ekanayake 2008]** J. Ekanayake, S. Pallickara, and G. Fox. MapReduce for data intensive scientific analysis. In *Fourth IEEE International Conference on eScience*, pages 277–284, 2008.

Backup slides

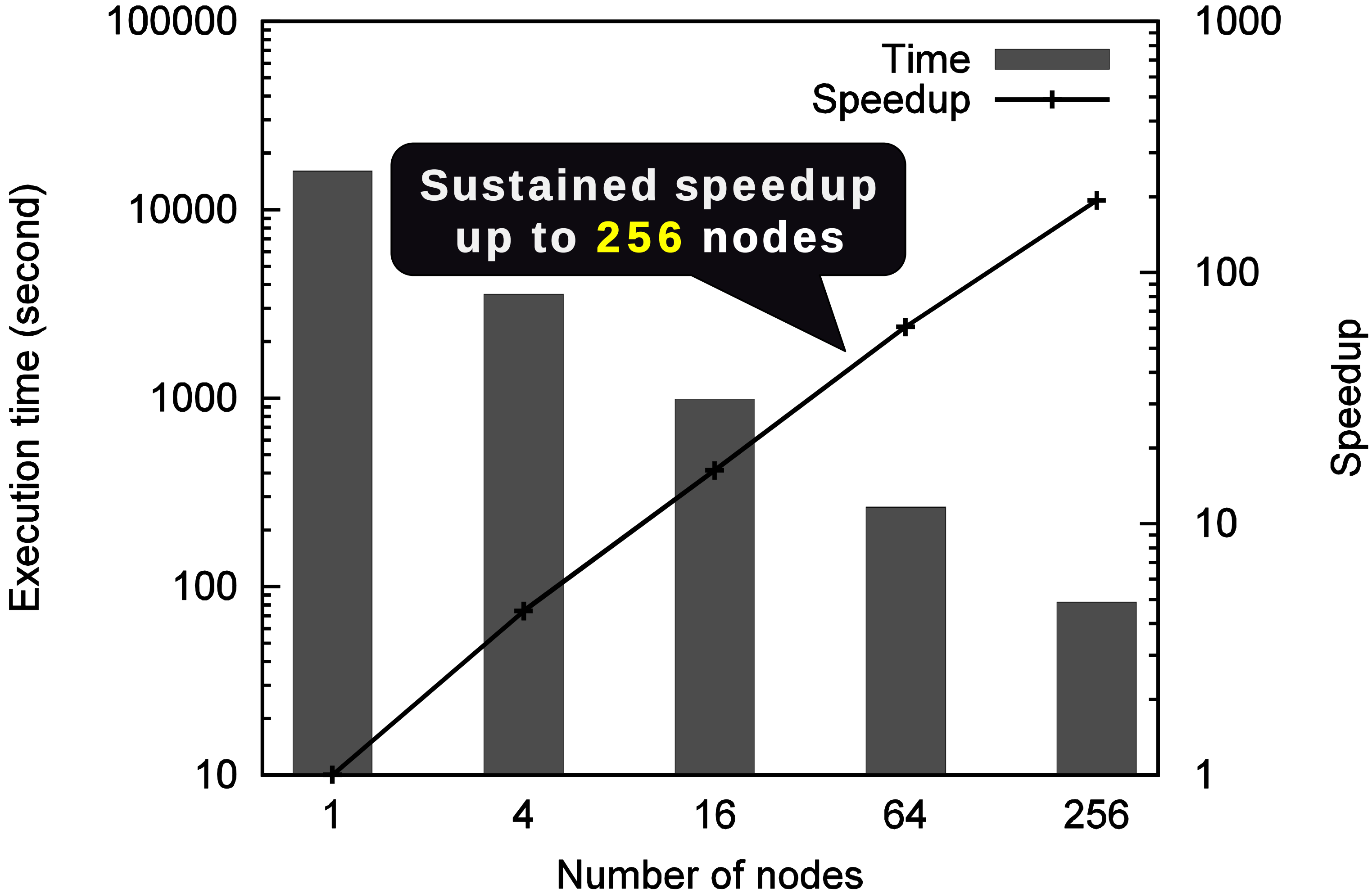
Scalability with number of nodes



Speedup and execution time of prime-checker
as the number of working nodes grows

General, scalable, efficient, portable, easy-to-program

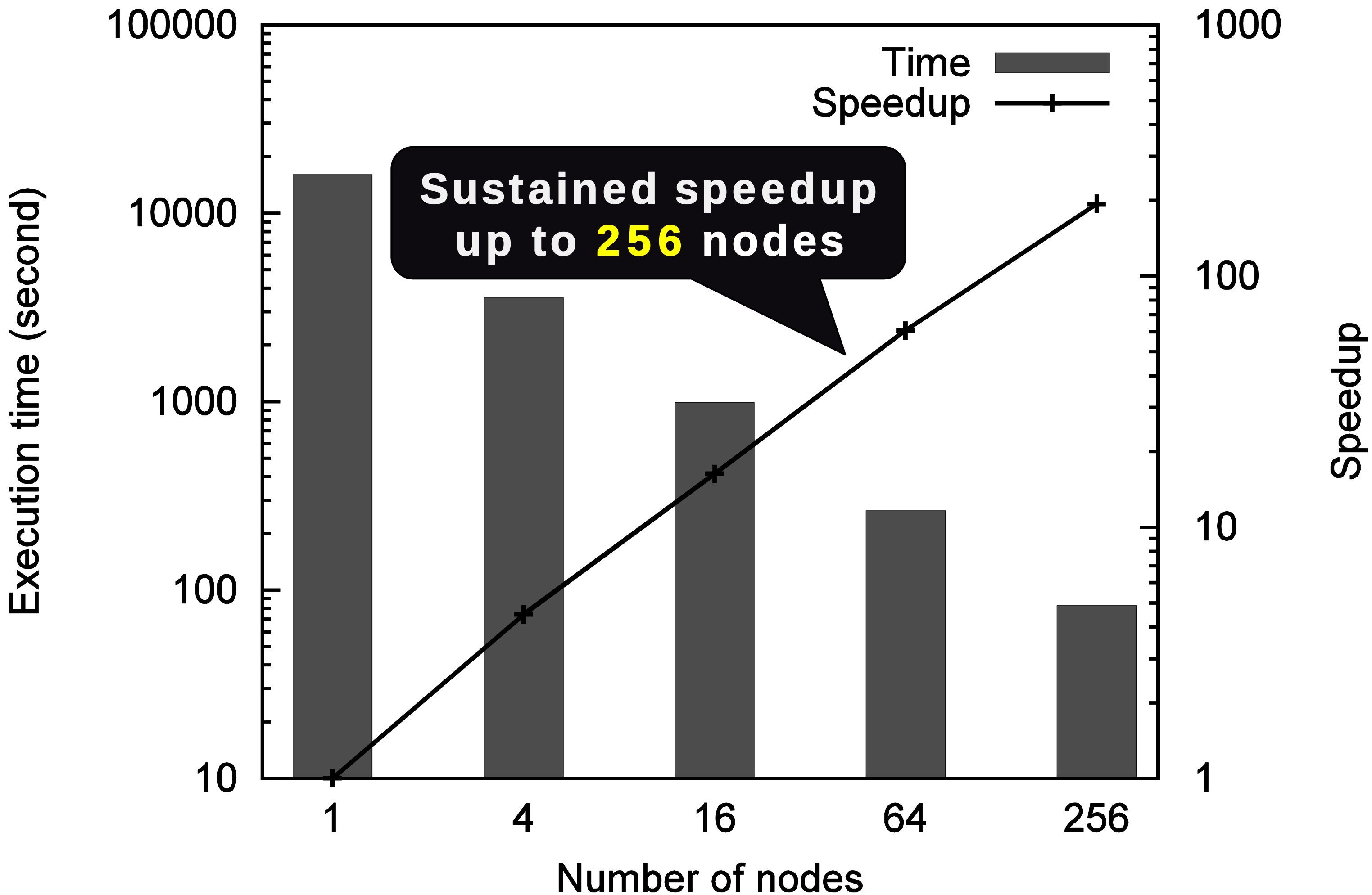
Scalability with number of nodes



Speedup and execution time of prime-checker as the number of working nodes grows

General, scalable, efficient, portable, easy-to-program

Scalability with number of nodes



Speedup and execution time of prime-checker as the number of working nodes grows

General, scalable, efficient, portable, easy-to-program