Optimizing execution time in a distributed virtual processor

Dan Bogdanov
researcher / PhD student
Cybernetica / University of Tartu

Outline of the talk

1. The Sharemind virtual processor
2. The command processing pipeline
3. Chosen benchmarks
4. Optimization techniques

What is Sharemind?

Sharemind

Sharemind

- A privacy-preserving virtual machine that

Sharemind

- A privacy-preserving virtual machine that
  - performs secure multi-party computation

Sharemind

- A privacy-preserving virtual machine that
  - performs secure multi-party computation
  - can add, multiply and compare integers

Sharemind

- A privacy-preserving virtual machine that
  - performs secure multi-party computation
  - can add, multiply and compare integers
  - has a simple programming interface

Sharemind

- A privacy-preserving virtual machine that
  - performs secure multi-party computation
    - can add, multiply and compare integers
    - has a simple programming interface
- Sharemind can be used for processing private data without compromising it

Data collection

Input data

f	32	teacher	Tallinn	...

Data collection

First share of the data

j34	f2b7	271pgn	823tfbffsd2	...

Second share of the data

dha	lcc7	nf62ng0	72v2ovgxfv	...

Third share of the data

vjn	h27	fjsfkha2	chdn7d2nd	...

Data collection

First share of the data

j34	f2b7	271pgn	823tfbffsd2	...

No share reveals anything about the original data

Second share of the data

dha	lcc7	nf62ng0	72v2ovgxfv	...

Third share of the data

vjn	h27	fjsfkha2	chdn7d2nd	...

Processing the data

First share of the data

j34	f2b7	271pgn	823tfbffsd2	...

Second share of the data

dha	lcc7	nf62ng0	72v2ovgxfv	...

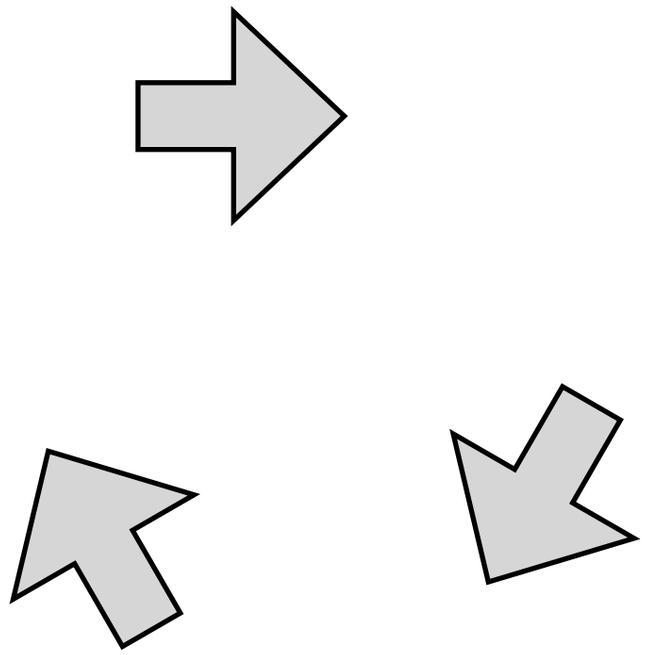
Third share of the data

vjn	h272	fjsfkha2	chdn7d2nd	...

Processing the data

First share of the data

j34	f2b7	271pgn	823tfbffsd2	...



Second share of the data

dha	lcc7	nf62ng0	72v2ovgxfv	...

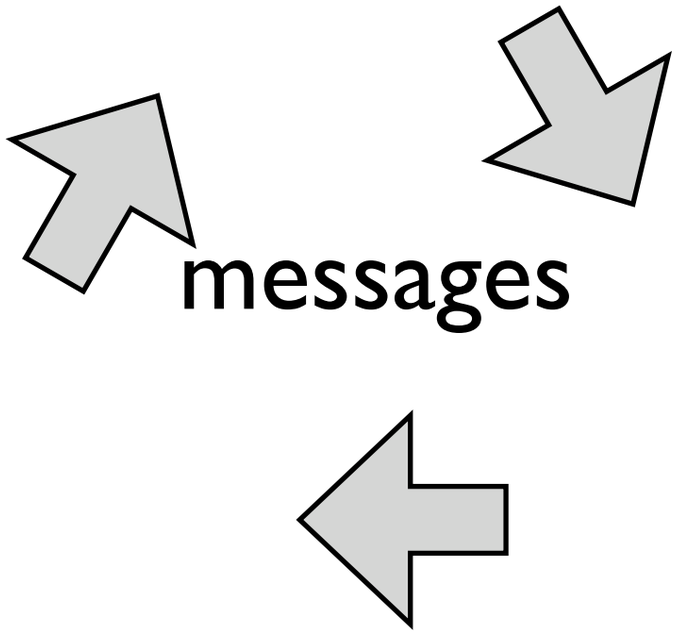
Third share of the data

vjn	h272	fjsfkha2	chdn7d2nd	...

Processing the data

First share of the data

j34	f2b7	271pgn	823tfbffsd2	...



Second share of the data

dha	lcc7	nf62ng0	72v2ovgxfv	...

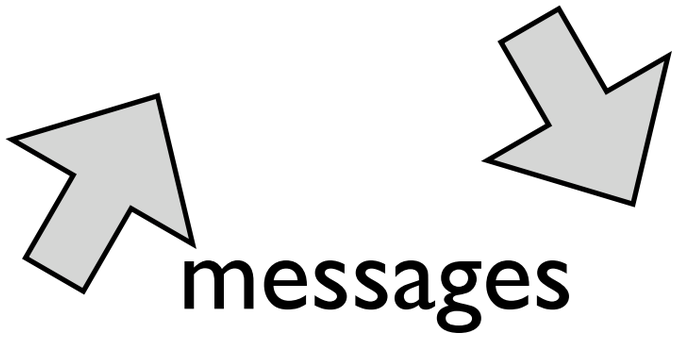
Third share of the data

vjn	h272	fjsfkha2	chdn7d2nd	...

Processing the data

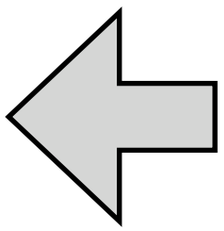
First share of the data

j34	f2b7	271pgn	823tfbffsd2	...



Second share of the data

dha	lcc7	nf62ng0	72v2ovgxfv	...



No message reveals anything about the data

Third share of the data

vjn	h272	fjsfkha2	chdn7d2nd	...

Processing the data

First share of the data

j34	f2b7	271pgn	823tfbffsd2	...

Second share of the data

dha	lcc7	nf62ng0	72v2ovgxfv	...

Third share of the data

vjn	h272	fjsfkha2	chdn7d2nd	...

Processing the data

First share of the data

j34	f2b7	271pgn	823tfbffsd2	...
gs32kq	bvx2as	huo2ei	...	

Second share of the data

dha	lcc7	nf62ng0	72v2ovgxfv	...
09fgh2	bnv6sd	2yfhod	...	

Third share of the data

vjn	h272	fjsfkha2	chdn7d2nd	...
439fxj	yvbwr2	l9sufg	...	

Reading the results

First share of the data

j34	f2b7	271pgn	823tfbffsd2	...
gs32kq	bvx2as	huo2ei	...	

Second share of the data

dha	lcc7	nf62ng0	72v2ovgxfv	...
09fgh2	bnv6sd	2yfhod	...	

Third share of the data

vjn	h272	fjsfkha2	chdn7d2nd	...
439fxj	yvbwr2	l9sufg	...	

Reading the results

First share of the data

gs32kq	bvx2as	huo2ei	...

Second share of the data

09fgh2	bnv6sd	2yfhod	...

Third share of the data

439fxj	yvbwr2	l9sufg	...

Reading the results

gs32kq	bvx2as	huo2ei	...

09fgh2	bnv6sd	2yfhod	...

439fxj	yvbwr2	l9sufg	...

Reading the results

Results

68% women	32% men	average income 6350 EEK	...

Security

Security

- The protocols are proven secure in the honest-but-curious model

Security

- The protocols are proven secure in the honest-but-curious model
- If a miner looks at its data it sees uniformly distributed values

Security

- The protocols are proven secure in the honest-but-curious model
- If a miner looks at its data it sees uniformly distributed values
- If two miners collude during a protocol, they can recover the secret values

Works in the real world

Works in the real world

- We have implemented it using C++

Works in the real world

- We have implemented it using C++
- The software consists of two parts

Works in the real world

- We have implemented it using C++
- The software consists of two parts

1) the data miner runtime server

Works in the real world

- We have implemented it using C++
- The software consists of two parts

1) the data miner runtime server

2) controller library for creating clients

Works in the real world

- We have implemented it using C++
- The software consists of two parts
  1) the data miner runtime server
  2) controller library for creating clients
- Uses the RakNet fast networking library

Works in the real world

- We have implemented it using C++
- The software consists of two parts
  1) the data miner runtime server
  2) controller library for creating clients
- Uses the RakNet fast networking library
- Runs (at least) on Linux, Mac OS X, Win XP

The processing pipeline

Execution model

Execution model

- Our processor is based on a stack machine

Execution model

- Our processor is based on a stack machine
  - The input is pushed on the stack

Execution model

- Our processor is based on a stack machine
  - The input is pushed on the stack
  - The operation is executed

Execution model

- Our processor is based on a stack machine
  - The input is pushed on the stack
  - The operation is executed
  - The results are read from the stack

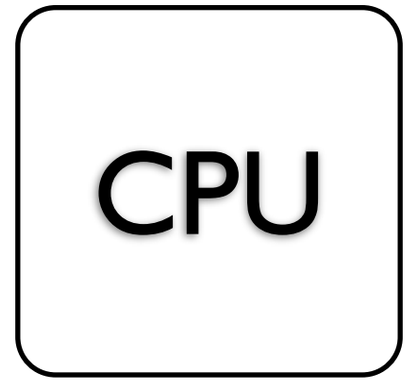
Execution model

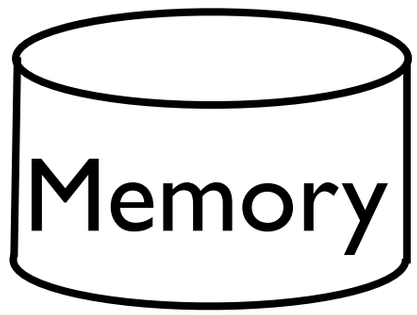
- Our processor is based on a stack machine
  - The input is pushed on the stack
  - The operation is executed
  - The results are read from the stack
- The CPU also has a heap and a database

Execution model

- Our processor is based on a stack machine
  - The input is pushed on the stack
  - The operation is executed
  - The results are read from the stack
- The CPU also has a heap and a database
  - Which basically make it Turing-complete

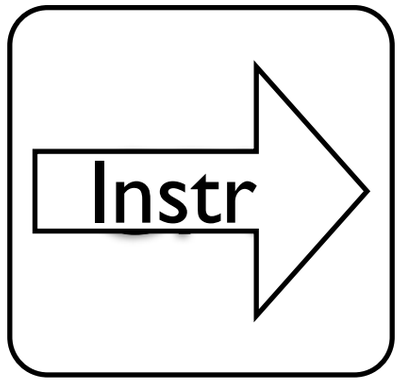
A generic computer

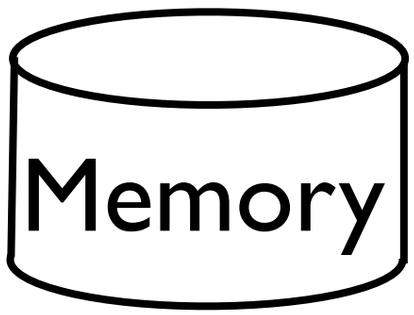




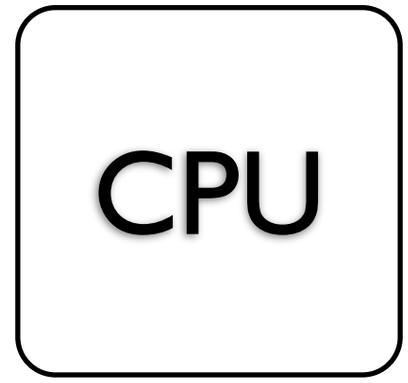
A generic computer

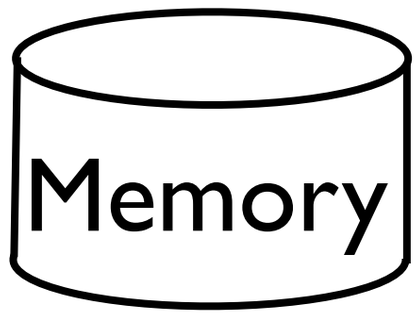
A generic computer



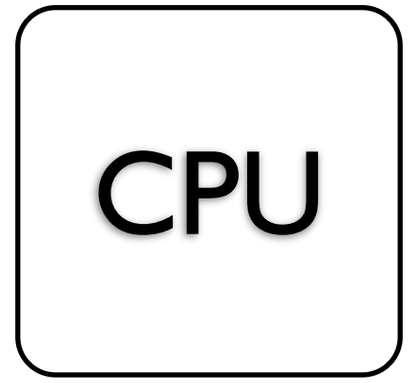


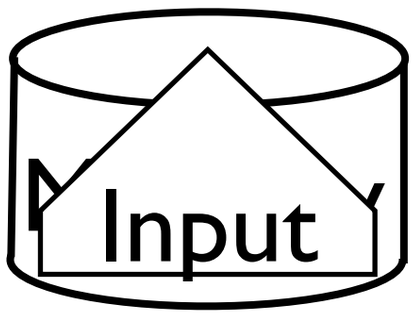
A generic computer



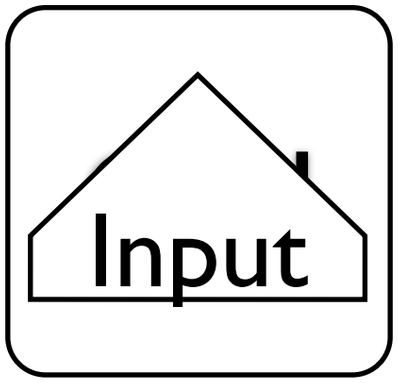


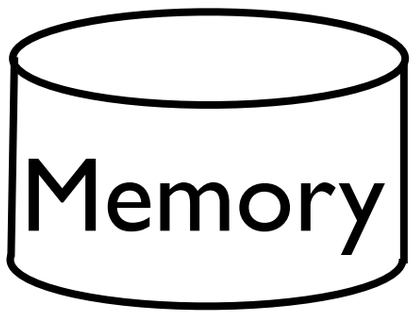
A generic computer



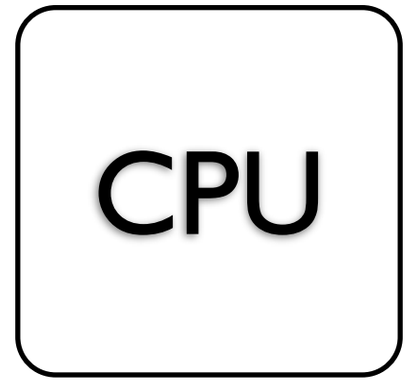


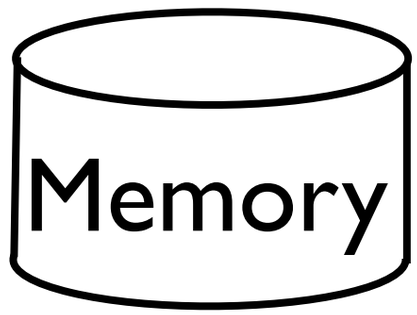
A generic computer



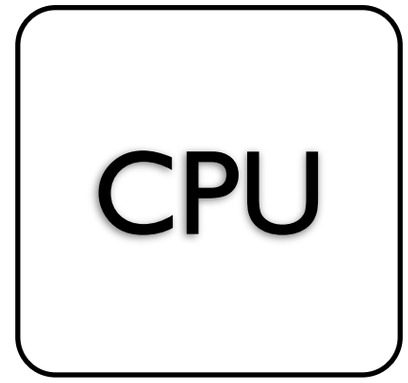


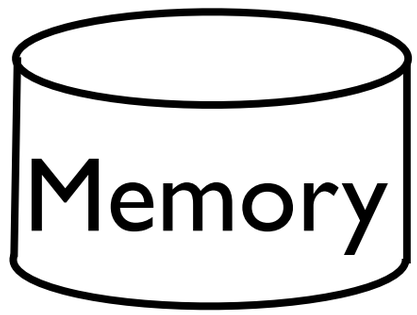
A generic computer



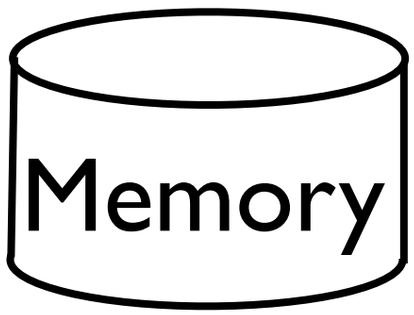


A generic computer

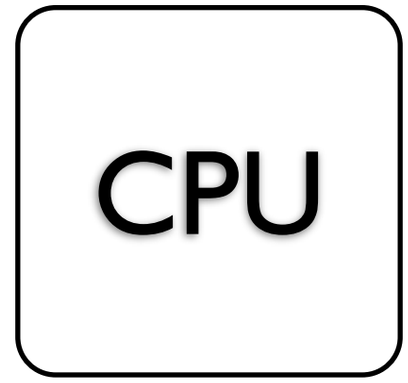


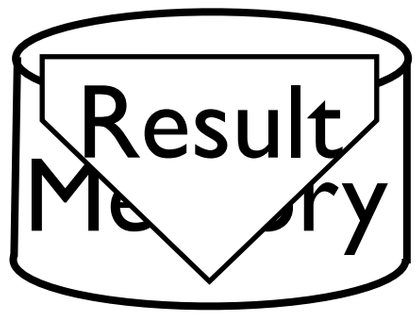


A generic computer

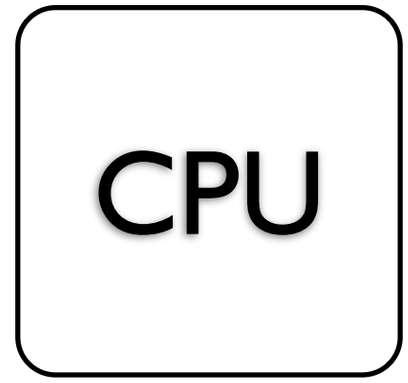


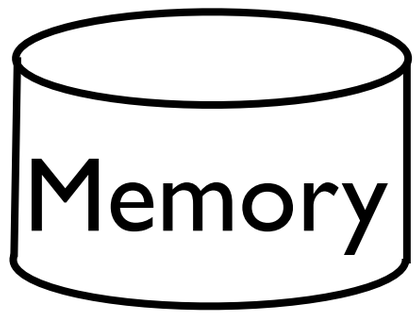
A generic computer



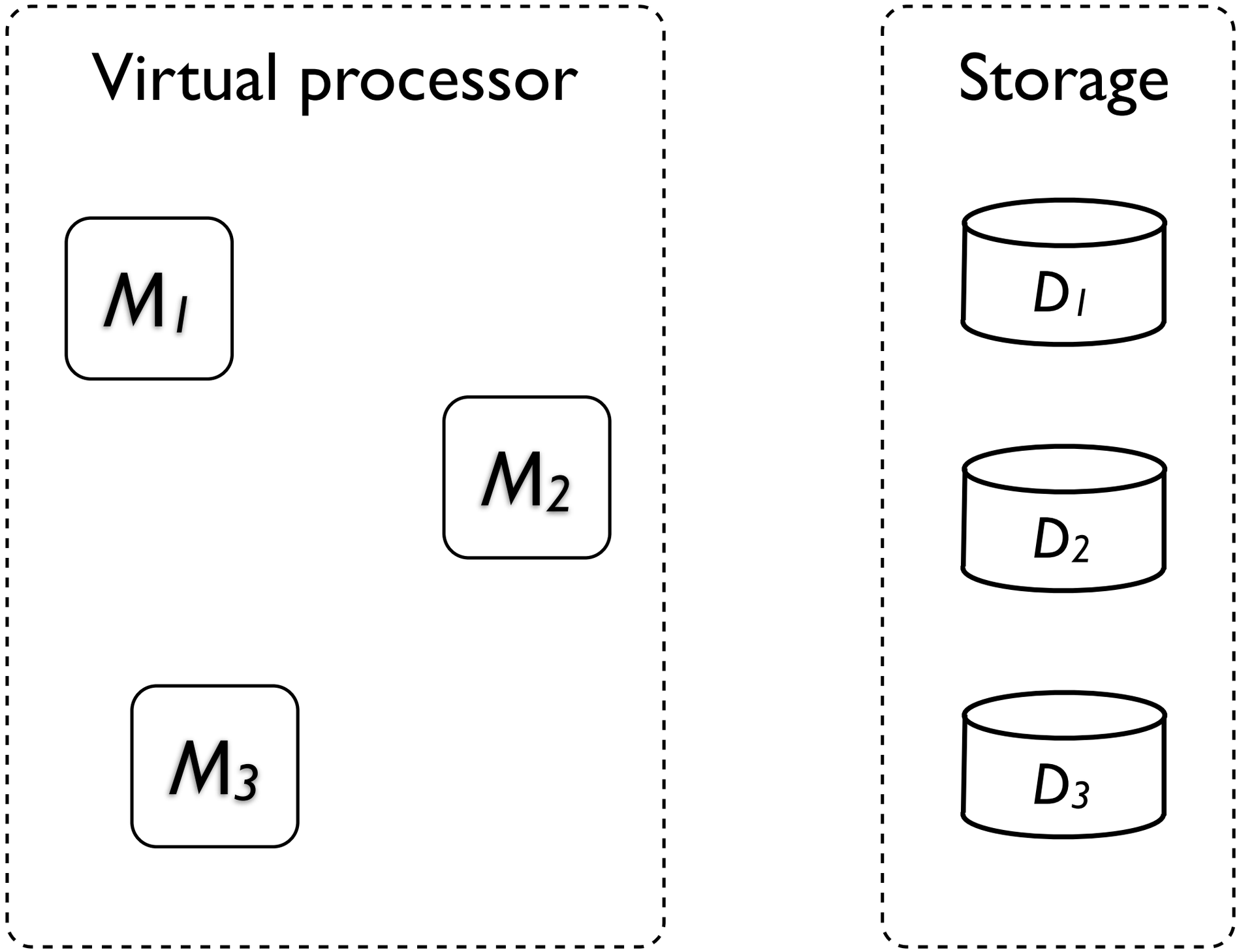


A generic computer

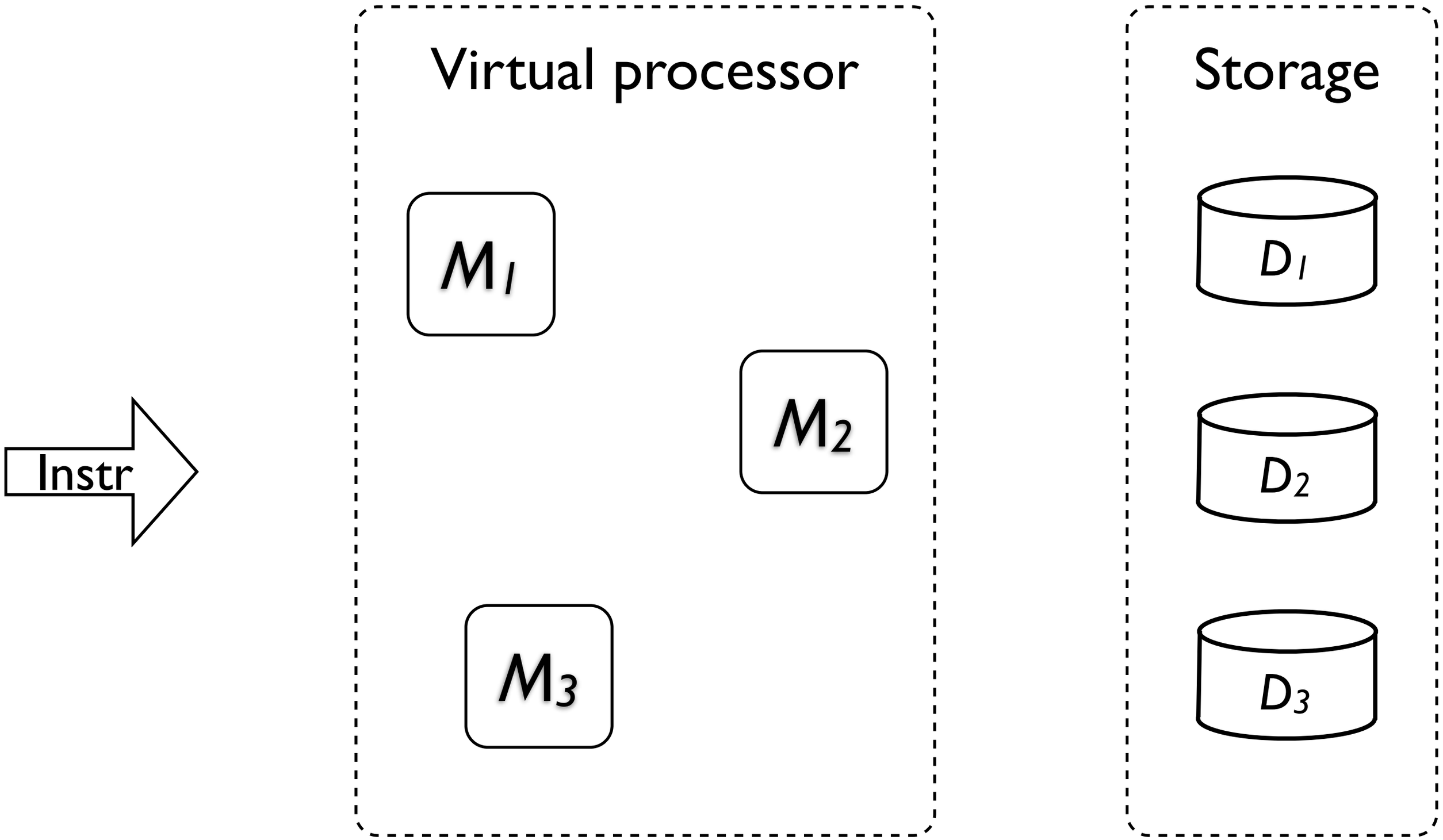




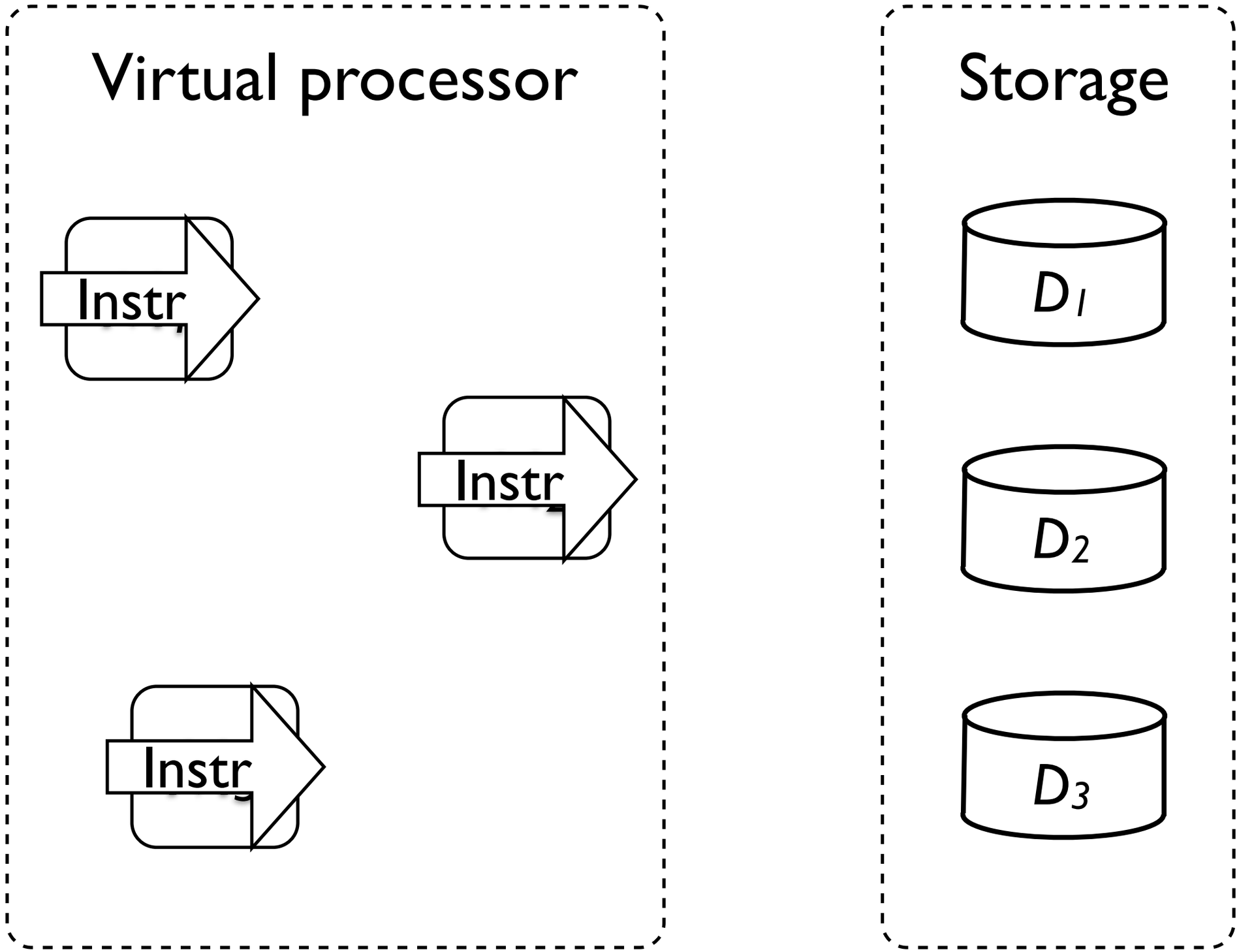
The three little miners



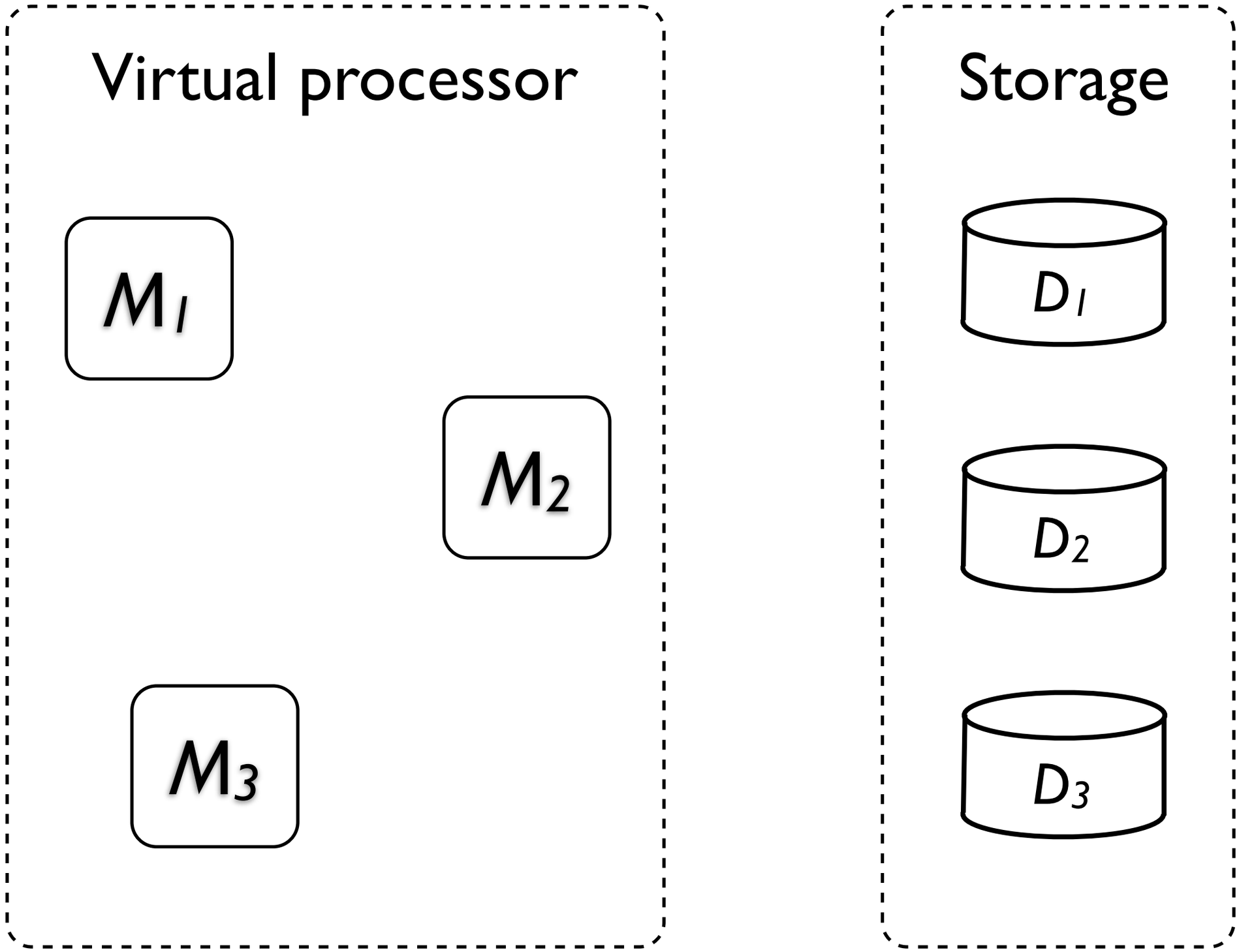
The three little miners



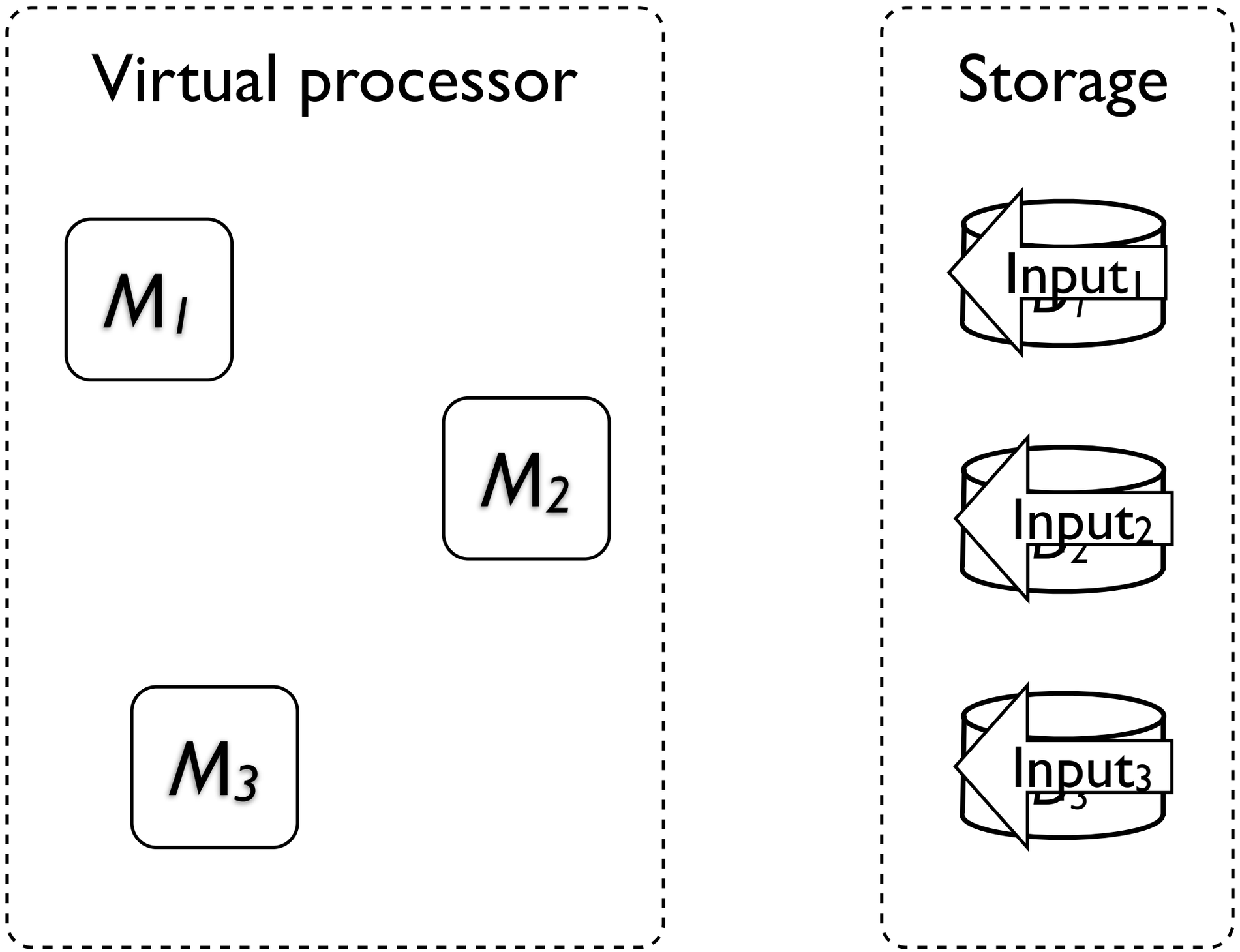
The three little miners



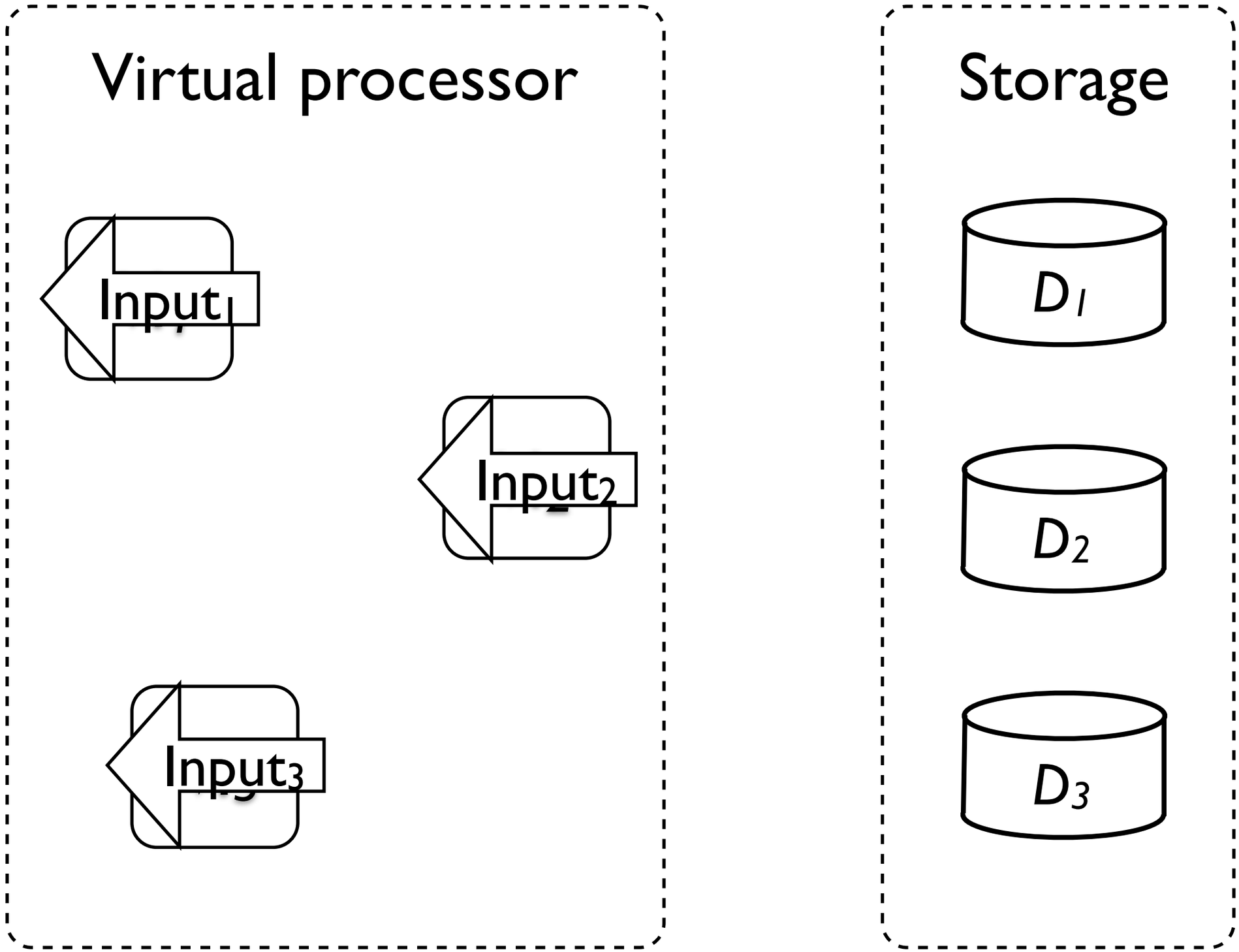
The three little miners



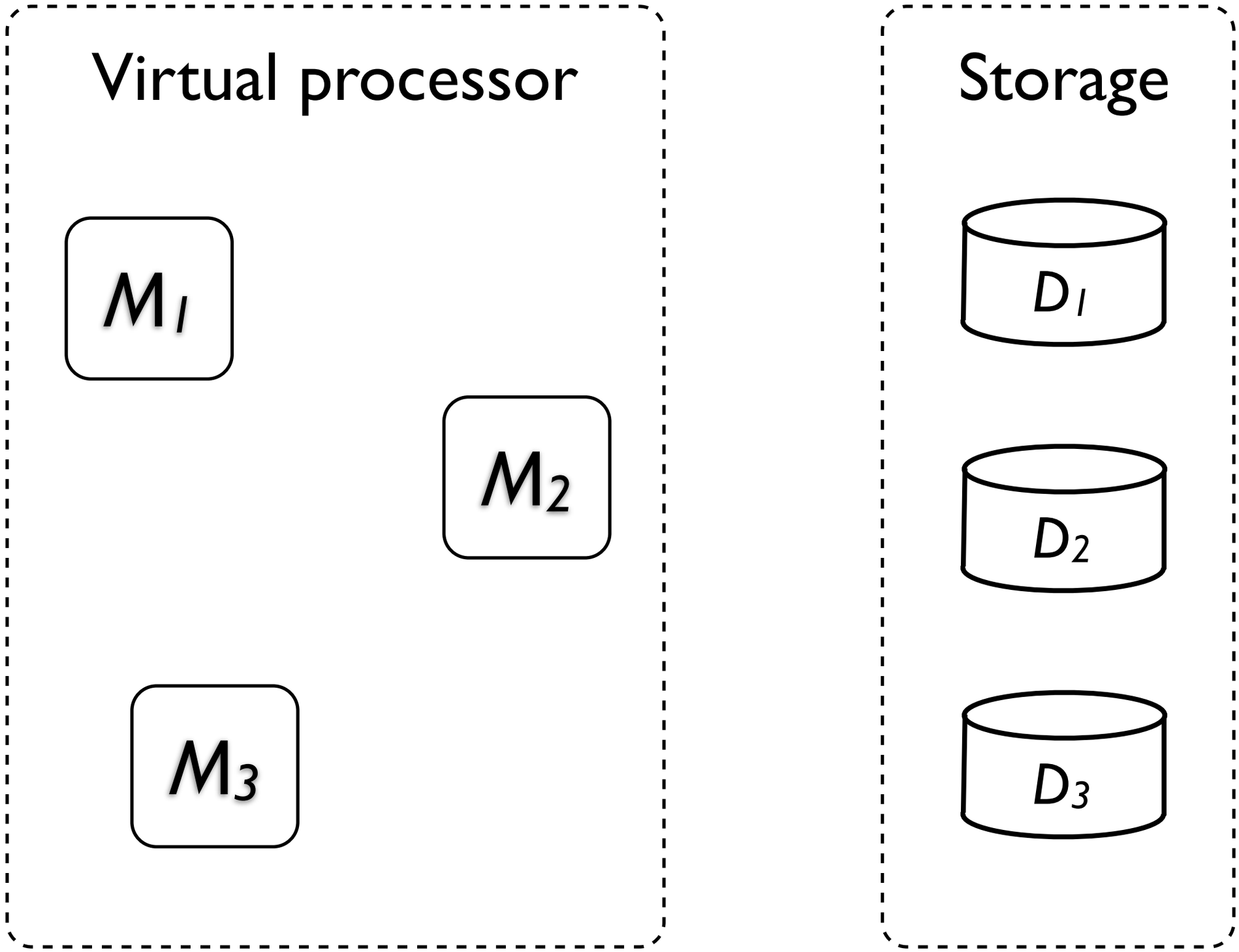
The three little miners



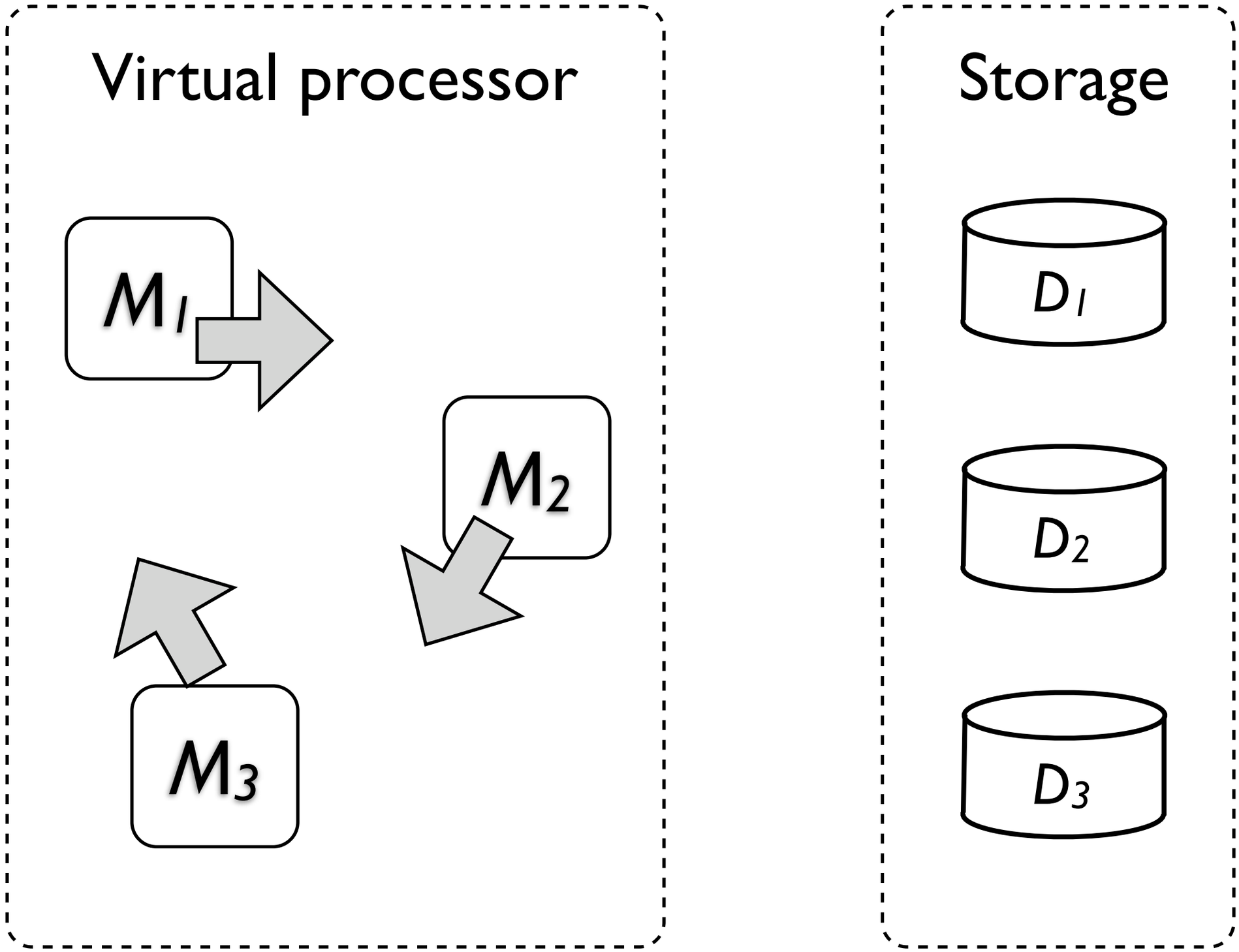
The three little miners



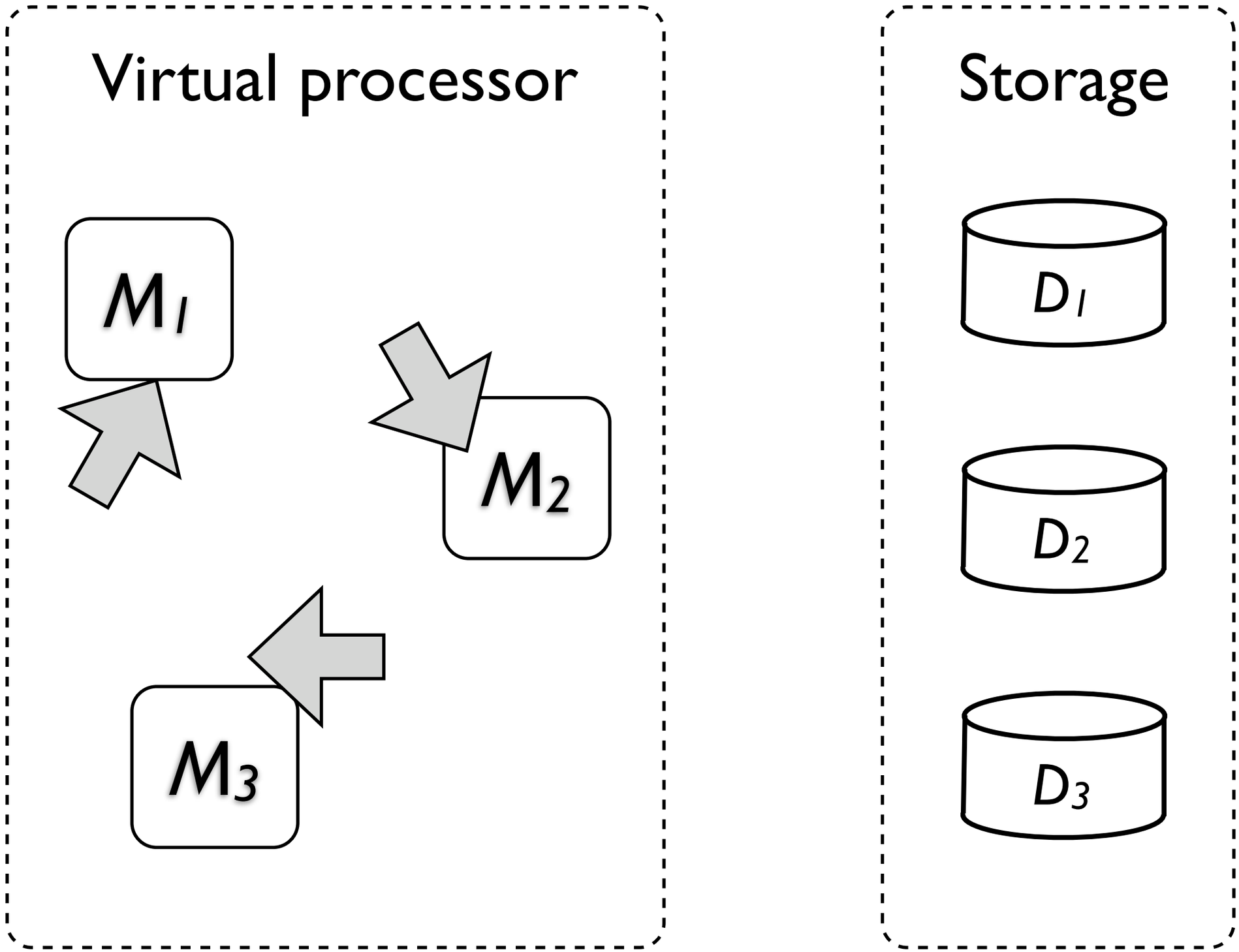
The three little miners



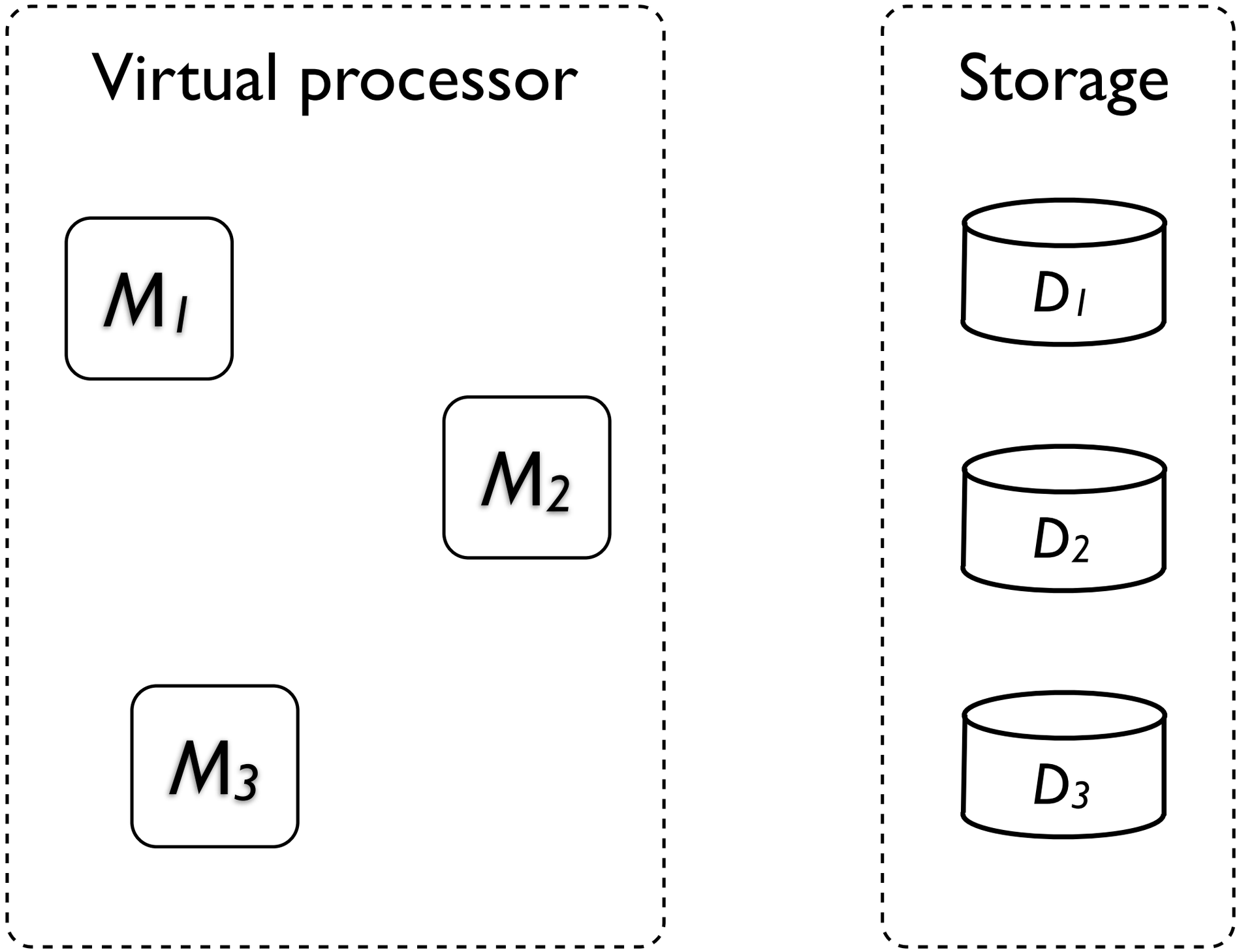
The three little miners



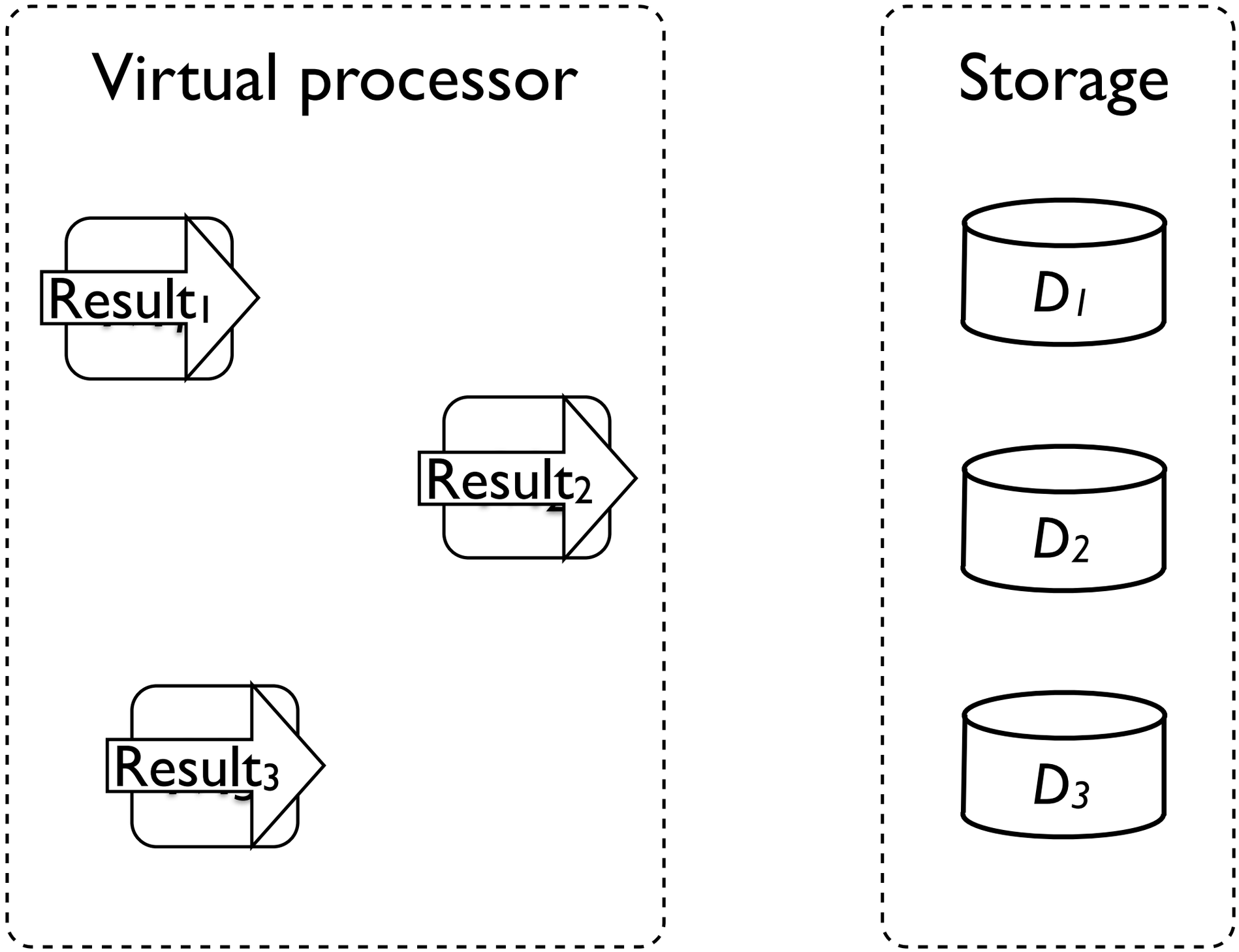
The three little miners



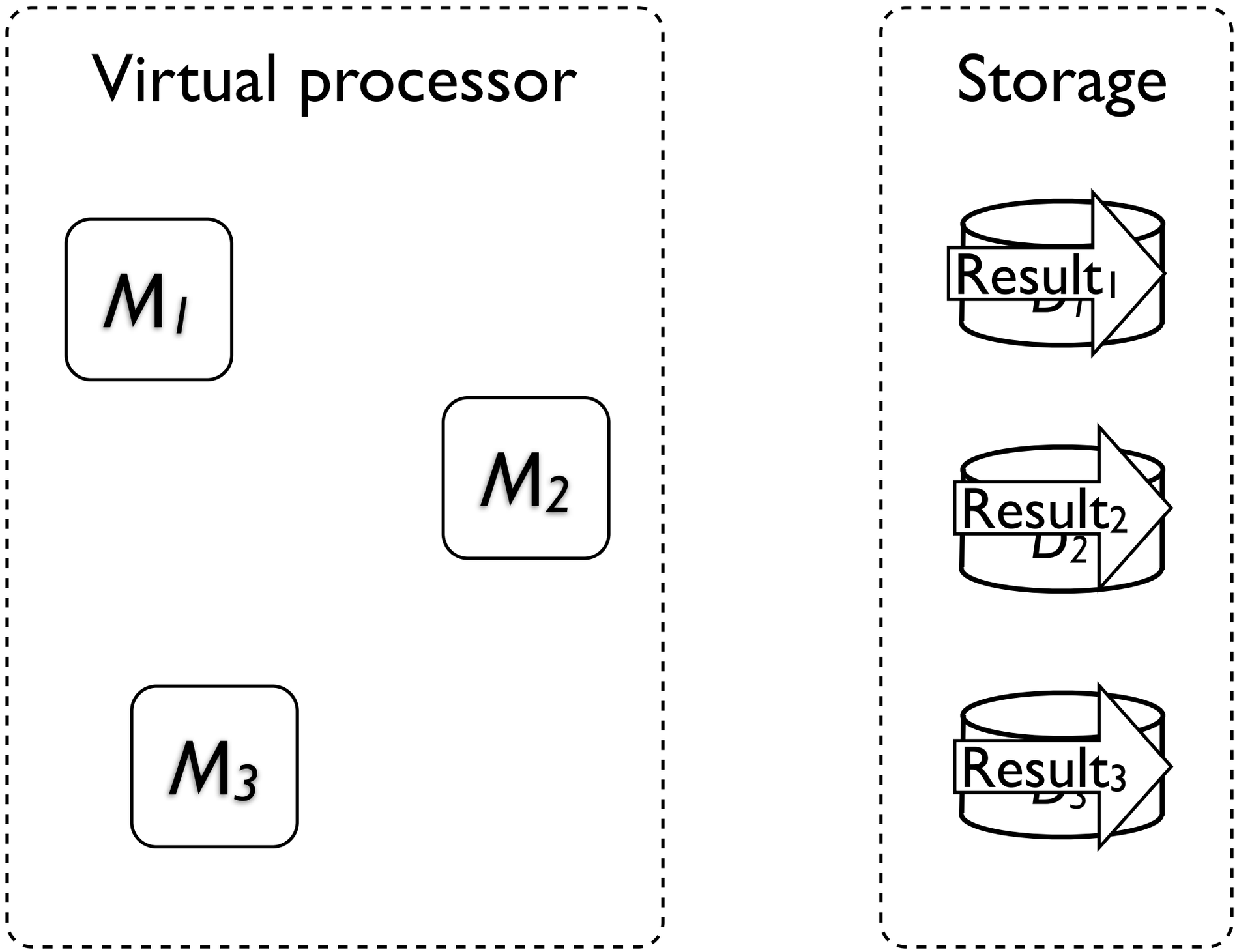
The three little miners



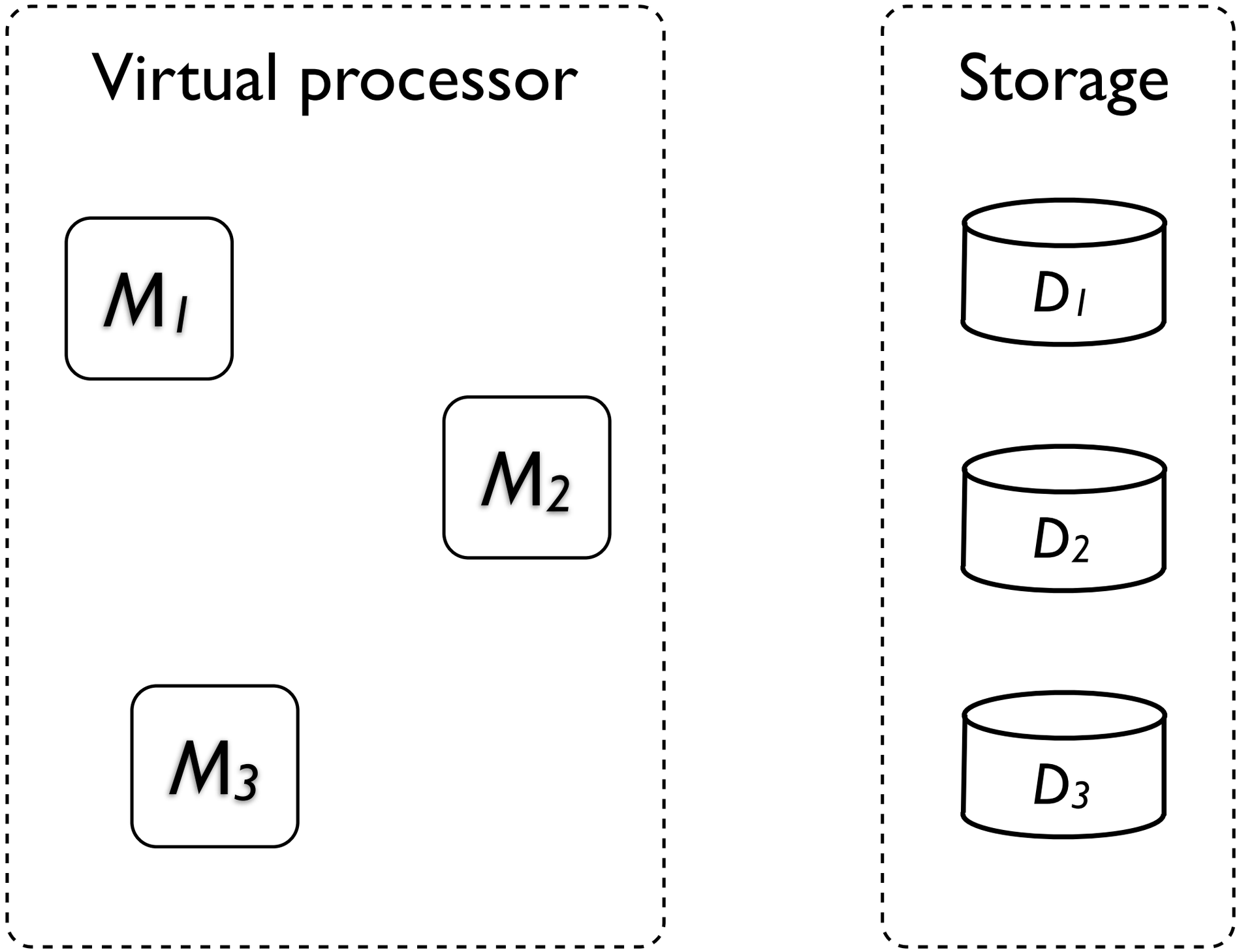
The three little miners



The three little miners



The three little miners



Operation scheduling

Operation scheduling

- Operation execution is divided into rounds

Operation scheduling

- Operation execution is divided into rounds
  - A round ends with message sending

Operation scheduling

- Operation execution is divided into rounds
  - A round ends with message sending
- Single-round operations are called *local*

Operation scheduling

- Operation execution is divided into rounds
  - A round ends with message sending
- Single-round operations are called *local*
- Multi-round operations can be scheduled with a granularity of one round

Operation scheduling

- Operation execution is divided into rounds
  - A round ends with message sending
- Single-round operations are called *local*
- Multi-round operations can be scheduled with a granularity of one round
  - This allows sub-operations (calls)

Chosen benchmarks

Benchmark operations

Benchmark operations

- We currently benchmark atomic operations and simple compositions

Benchmark operations

- We currently benchmark atomic operations and simple compositions
- Our current candidates are:

Benchmark operations

- We currently benchmark atomic operations and simple compositions
- Our current candidates are:
  - scalar product

Benchmark operations

- We currently benchmark atomic operations and simple compositions
- Our current candidates are:
  - scalar product
  - vectorized comparison

Benchmark methods

Benchmark methods

- We measure execution time in milliseconds

Benchmark methods

- We measure execution time in milliseconds
- Option 1: Measure at the client

Benchmark methods

- We measure execution time in milliseconds
- Option 1: Measure at the client
  - What a client sees. Includes overhead.

Benchmark methods

- We measure execution time in milliseconds
- Option 1: Measure at the client
  - What a client sees. Includes overhead.
- Option 2: Measure at the miners

Benchmark methods

- We measure execution time in milliseconds
- Option 1: Measure at the client
  - What a client sees. Includes overhead.
- Option 2: Measure at the miners
  - Exact computation time. No overhead.

Benchmark methods

- We measure execution time in milliseconds
- Option 1: Measure at the client
  - What a client sees. Includes overhead.
- Option 2: Measure at the miners
  - Exact computation time. No overhead.
- We have been using both methods

Experimental setup

Experimental setup

- Repeat each experiment a number of times

Experimental setup

- Repeat each experiment a number of times
- Throw away first 10% of results after the test program starts up.

Experimental setup

- Repeat each experiment a number of times
- Throw away first 10% of results after the test program starts up.
- This eliminates the effects of the network layer flow control algorithms.

Experimental setup

- Repeat each experiment a number of times
- Throw away first 10% of results after the test program starts up.
  - This eliminates the effects of the network layer flow control algorithms.
  - A real-life system is “always on” and doesn’t need flow calibration so often.

Optimization techniques

Where does time go?

Where does time go?

- We can group work done in a distributed system in three categories

Where does time go?

- We can group work done in a distributed system in three categories

1. Actual computation at the miners

Where does time go?

- We can group work done in a distributed system in three categories

1. Actual computation at the miners
2. Sending data to other miners

Where does time go?

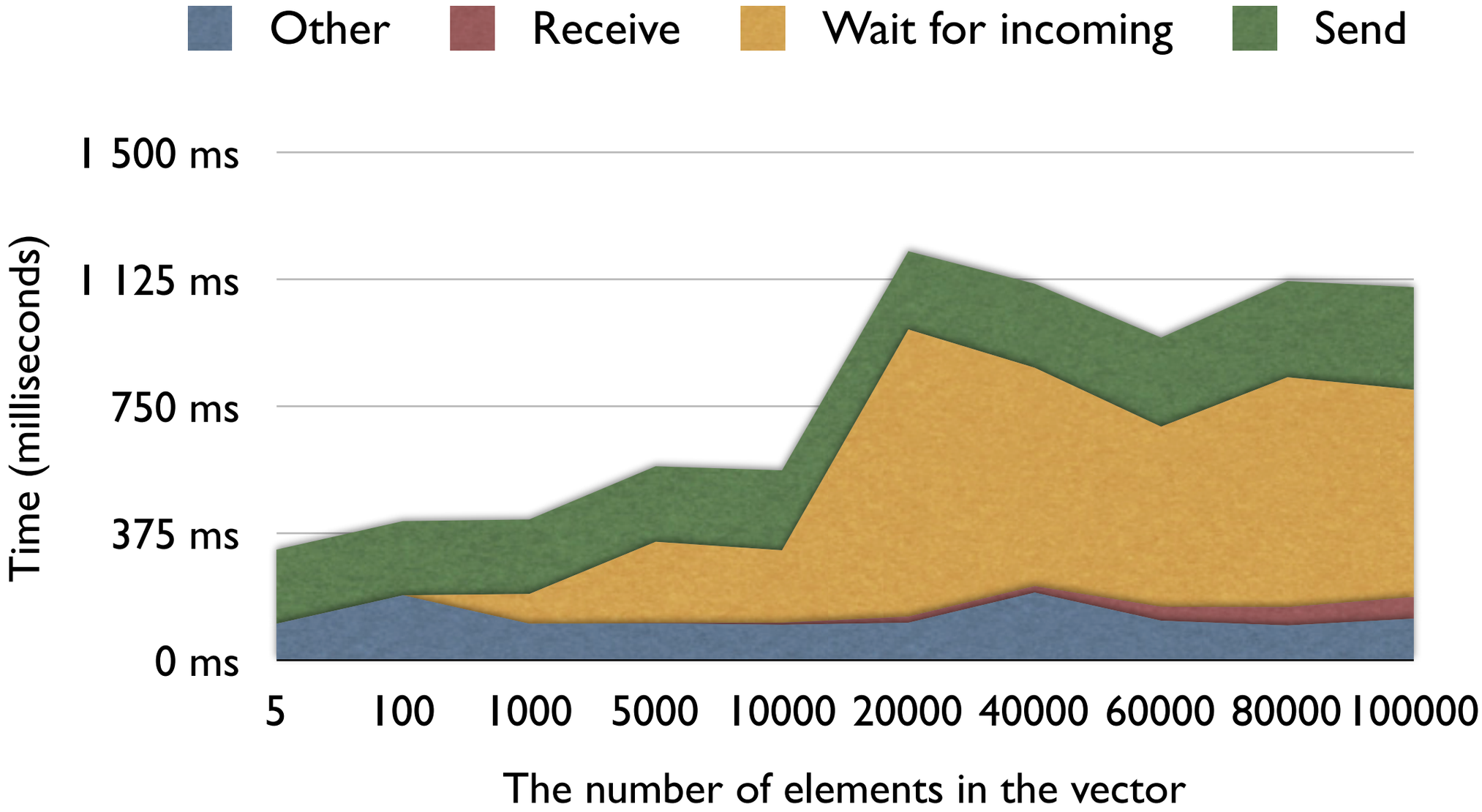
- We can group work done in a distributed system in three categories

1. Actual computation at the miners
2. Sending data to other miners
3. Receiving data from other miners

Where does time go?

- We can group work done in a distributed system in three categories
    1. Actual computation at the miners
    2. Sending data to other miners
    3. Receiving data from other miners
- Find out the truth by running experiments

Profiling scalar product



Direct implications

Direct implications

- Time required for sending network packets grows by little

Direct implications

- Time required for sending network packets grows by little
- Time for receiving and interpreting network packets grows by little

Direct implications

- Time required for sending network packets grows by little
- Time for receiving and interpreting network packets grows by little
- The computation time grows by little

Direct implications

- Time required for sending network packets grows by little
- Time for receiving and interpreting network packets grows by little
- The computation time grows by little
- Waiting for data takes most of the time

Why are we waiting?

Why are we waiting?

- Waiting happens between rounds

Why are we waiting?

- Waiting happens between rounds
- The miner has computed something

Why are we waiting?

- Waiting happens between rounds
- The miner has computed something
- The results have been sent out

Why are we waiting?

- Waiting happens between rounds
- The miner has computed something
- The results have been sent out
- To continue, the miner must have the computation results from other miners

What controls the delay?

What controls the delay?

- The delay is influenced by the following:

What controls the delay?

- The delay is influenced by the following:
  - network speed

What controls the delay?

- The delay is influenced by the following:
  - network speed
  - size of the packets

What controls the delay?

- The delay is influenced by the following:
  - network speed
  - size of the packets
  - number of packets

What controls the delay?

- The delay is influenced by the following:
  - network speed
  - size of the packets
  - number of packets
  - organization of the protocol

Network latency

Network latency

- Test machines are on a 1 Gbps network

Network latency

- Test machines are on a 1 Gbps network
- Hard to optimize much further :)

Network latency

- Test machines are on a 1 Gbps network
- Hard to optimize much further :)
- Reality is bound to be a lot worse anyway

Packet size and count

Packet size and count

- Big packets are hard to transmit

Packet size and count

- Big packets are hard to transmit
- Many packets create too much overhead

Packet size and count

- Big packets are hard to transmit
- Many packets create too much overhead
- Obviously, a balance is required

Packet size and count

- Big packets are hard to transmit
- Many packets create too much overhead
- Obviously, a balance is required
- Actually, two clever ideas can be derived:

Packet size and count

- Big packets are hard to transmit
- Many packets create too much overhead
- Obviously, a balance is required
- Actually, two clever ideas can be derived:
  - vectorization

Packet size and count

- Big packets are hard to transmit
- Many packets create too much overhead
- Obviously, a balance is required
- Actually, two clever ideas can be derived:
  - vectorization
  - batched processing

Vectorized operations

Vectorized operations

- A standard solution in CPU design

Vectorized operations

- A standard solution in CPU design
- Instead of doing a single operation at once, do many similar operations simultaneously

Vectorized operations

- A standard solution in CPU design
- Instead of doing a single operation at once, do many similar operations simultaneously
  - multiply 100 000 values at the same time

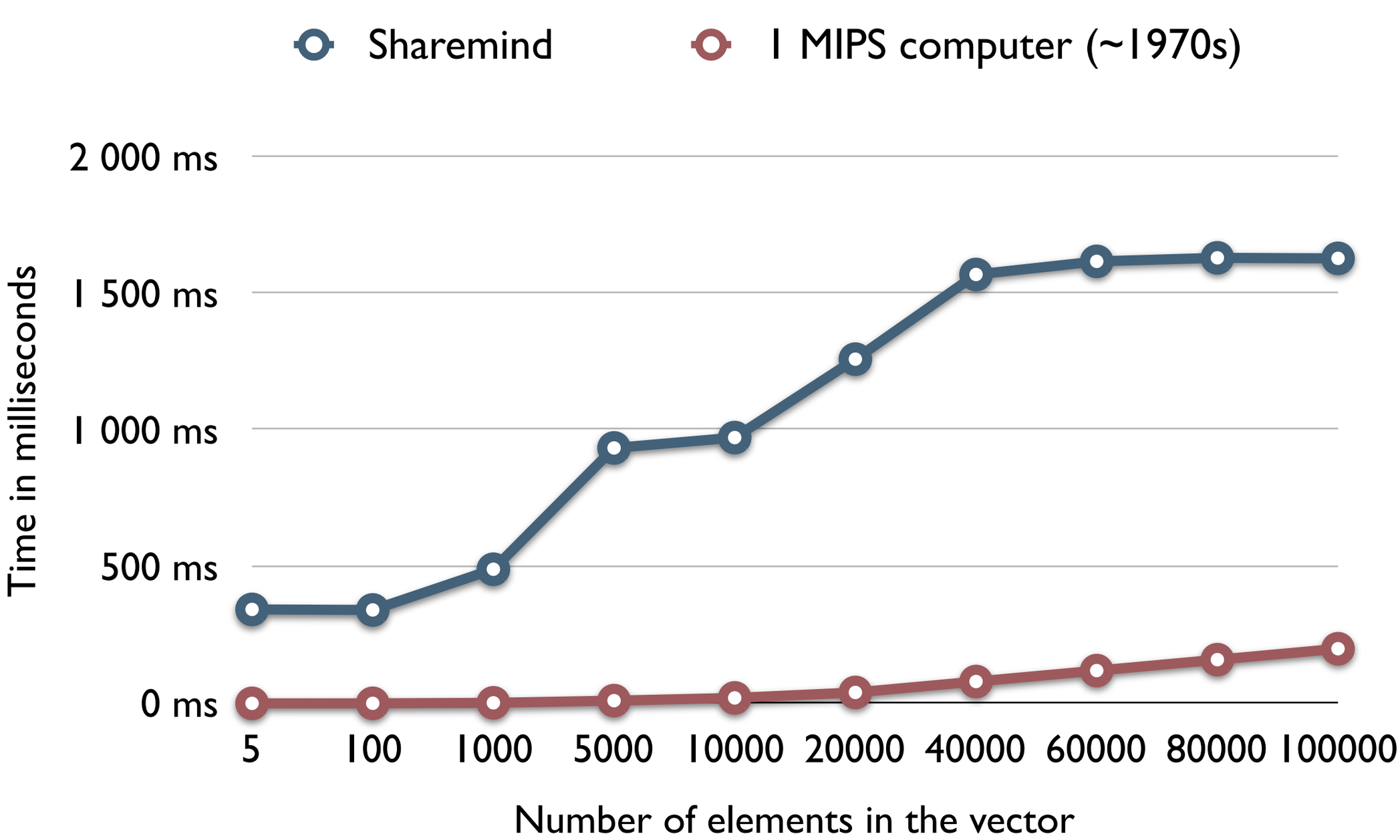
Vectorized operations

- A standard solution in CPU design
- Instead of doing a single operation at once, do many similar operations simultaneously
  - multiply 100 000 values at the same time
- This optimizes packet count, but makes the packets bigger

Vectorized operations

- A standard solution in CPU design
- Instead of doing a single operation at once, do many similar operations simultaneously
  - multiply 100 000 values at the same time
- This optimizes packet count, but makes the packets bigger
- The performance increase is substantial

Vectorization is good



Batched processing

Batched processing

- Also a standard solution in this area

Batched processing

- Also a standard solution in this area
- Instead of processing a lot of data at once, process a piece of it and send it to the next miner so it could start work earlier

Batched processing

- Also a standard solution in this area
- Instead of processing a lot of data at once, process a piece of it and send it to the next miner so it could start work earlier
- Decreases packet size, but increases count

Batched processing

- Also a standard solution in this area
- Instead of processing a lot of data at once, process a piece of it and send it to the next miner so it could start work earlier
- Decreases packet size, but increases count
- Balances the effects of vectorization

Batched processing

- Also a standard solution in this area
- Instead of processing a lot of data at once, process a piece of it and send it to the next miner so it could start work earlier
- Decreases packet size, but increases count
- Balances the effects of vectorization
- Improves performance by about 10%

Protocol organization

Protocol organization

- Each round increases the waiting time

Protocol organization

- Each round increases the waiting time
  - Try to decrease the number of rounds?

Protocol organization

- Each round increases the waiting time
  - Try to decrease the number of rounds?
- The temporal arrangement also matters

Protocol organization

- Each round increases the waiting time
  - Try to decrease the number of rounds?
- The temporal arrangement also matters
  - Try to arrange the work so that miners have to wait as little as possible

Number of rounds

Number of rounds

- Example: we replaced a logarithmic-time comparison protocol with a linear-time one

Number of rounds

- Example: we replaced a logarithmic-time comparison protocol with a linear-time one
- The new protocol was 3-10 times faster

Number of rounds

- Example: we replaced a logarithmic-time comparison protocol with a linear-time one
- The new protocol was 3-10 times faster
- It was also a lot more complex

Number of rounds

- Example: we replaced a logarithmic-time comparison protocol with a linear-time one
- The new protocol was 3-10 times faster
- It was also a lot more complex
- Decreasing the number of rounds in a protocol is harder

Work instead of waiting

Work instead of waiting

- We could build a better scheduler, which would do useful work instead of waiting

Work instead of waiting

- We could build a better scheduler, which would do useful work instead of waiting
- It would run a protocol round only when all the needed inputs have arrived

Work instead of waiting

- We could build a better scheduler, which would do useful work instead of waiting
- It would run a protocol round only when all the needed inputs have arrived
- It could run computation for other queries while waiting for data

Work instead of waiting

- We could build a better scheduler, which would do useful work instead of waiting
- It would run a protocol round only when all the needed inputs have arrived
- It could run computation for other queries while waiting for data
- We need more operations to run in parallel

Parallel execution

Parallel execution

- A program could run instructions in parallel

Parallel execution

- A program could run instructions in parallel
- Will give concurrent protocol instances...

Parallel execution

- A program could run instructions in parallel
- Will give concurrent protocol instances...
- ...but the stack machine fails right there

Parallel execution

- A program could run instructions in parallel
- Will give concurrent protocol instances...
- ...but the stack machine fails right there
- A synchronous machine will be needed

Parallel execution

- A program could run instructions in parallel
- Will give concurrent protocol instances...
- ...but the stack machine fails right there
- A synchronous machine will be needed
- We can also run queries for many clients at the same time (like a database server)

Conclusions

Conclusions

- Waiting for computation inputs between rounds is the biggest performance killer

Conclusions

- Waiting for computation inputs between rounds is the biggest performance killer
- Vectorization + batch processing provide a method for balancing network traffic

Conclusions

- Waiting for computation inputs between rounds is the biggest performance killer
- Vectorization + batch processing provide a method for balancing network traffic
- Reducing the number of protocol rounds will give the biggest improvements

Conclusions

- Waiting for computation inputs between rounds is the biggest performance killer
- Vectorization + batch processing provide a method for balancing network traffic
- Reducing the number of protocol rounds will give the biggest improvements
- Parallel execution will not improve speed, but it will improve throughput

That is all.