
SECURE
STREAMS

SECURESTREAMS:

A Reactive Middleware Framework for Secure Data Stream Processing

Aurélien HAVET - aurelien.havet@unine.ch

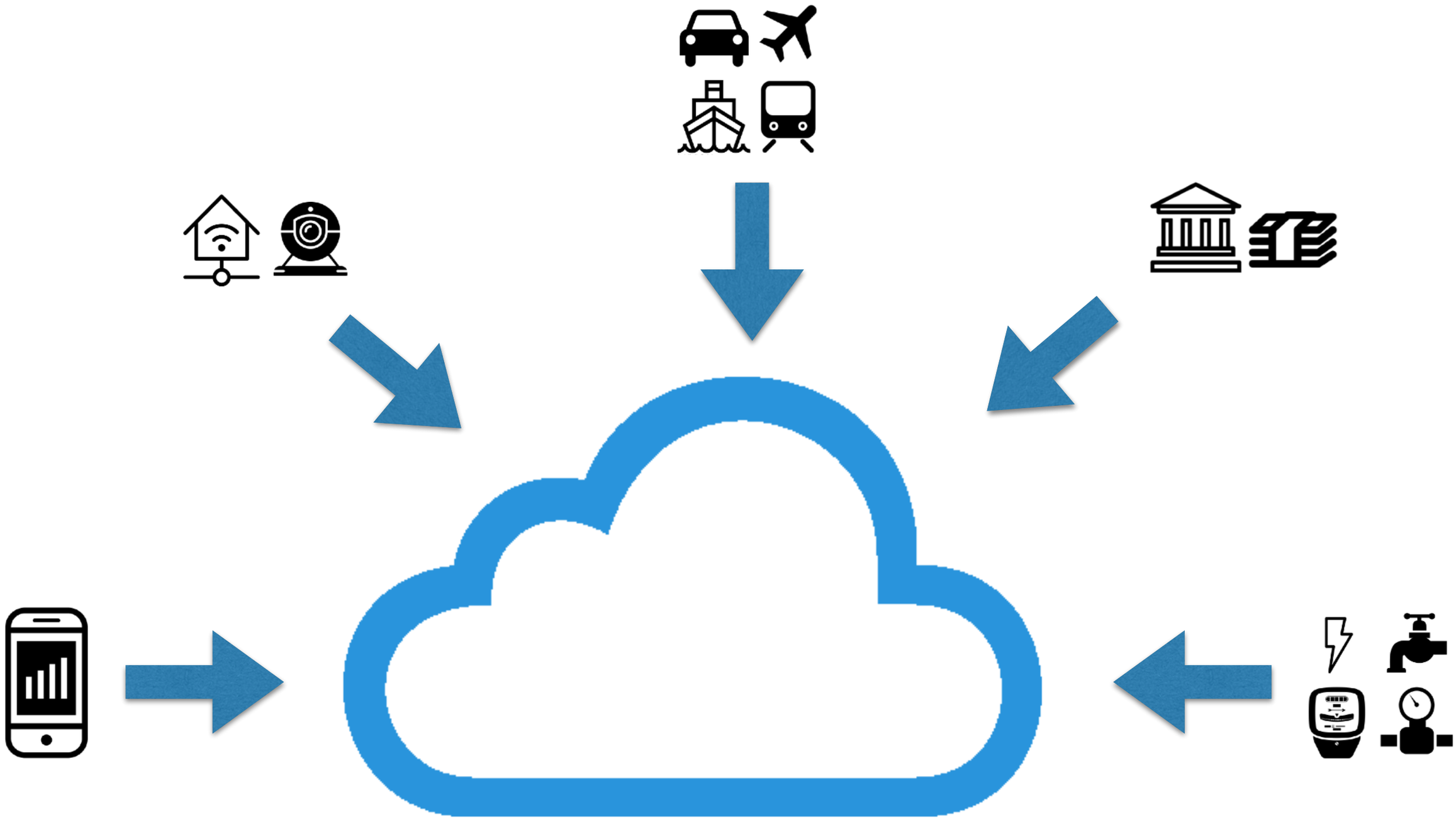
Rafael PIRES, Marcelo PASIN, Pascal FELBER, Valerio SCHIAVONI
(UniNE, Neuchâtel, CH)
Romain ROUVOY (INRIA, Lille, FR)

Roadmap

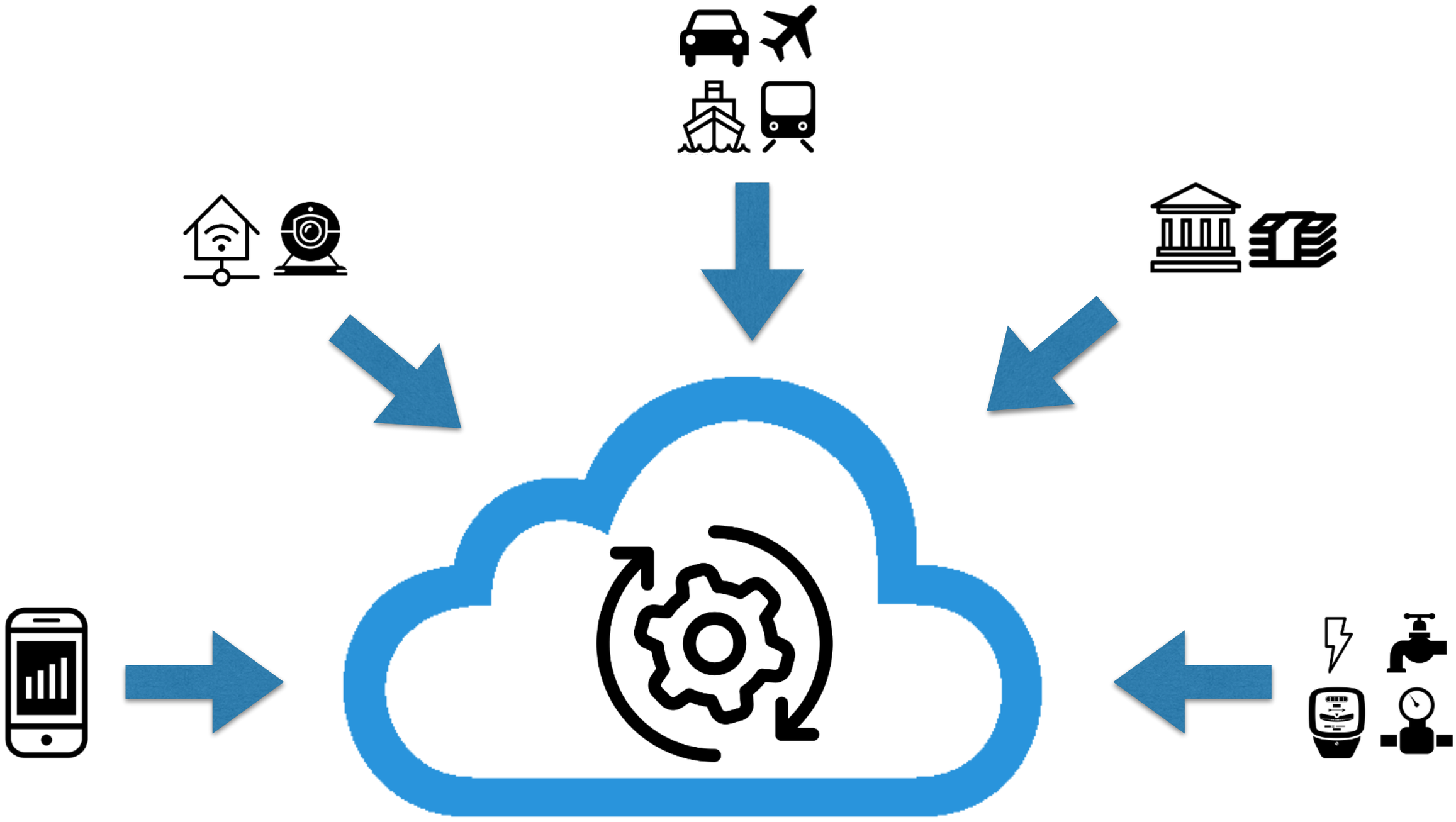
- Sensitive data stream processing context
- SGX overview
- SecureStreams: architecture and implementation
- Preliminary evaluation
- Conclusion and future work

Sensitive data stream processing context

Sensitive data stream processing context



Sensitive data stream processing context



SGX hardware

Skylake products: https://ark.intel.com/products/codename/37572/Skylake

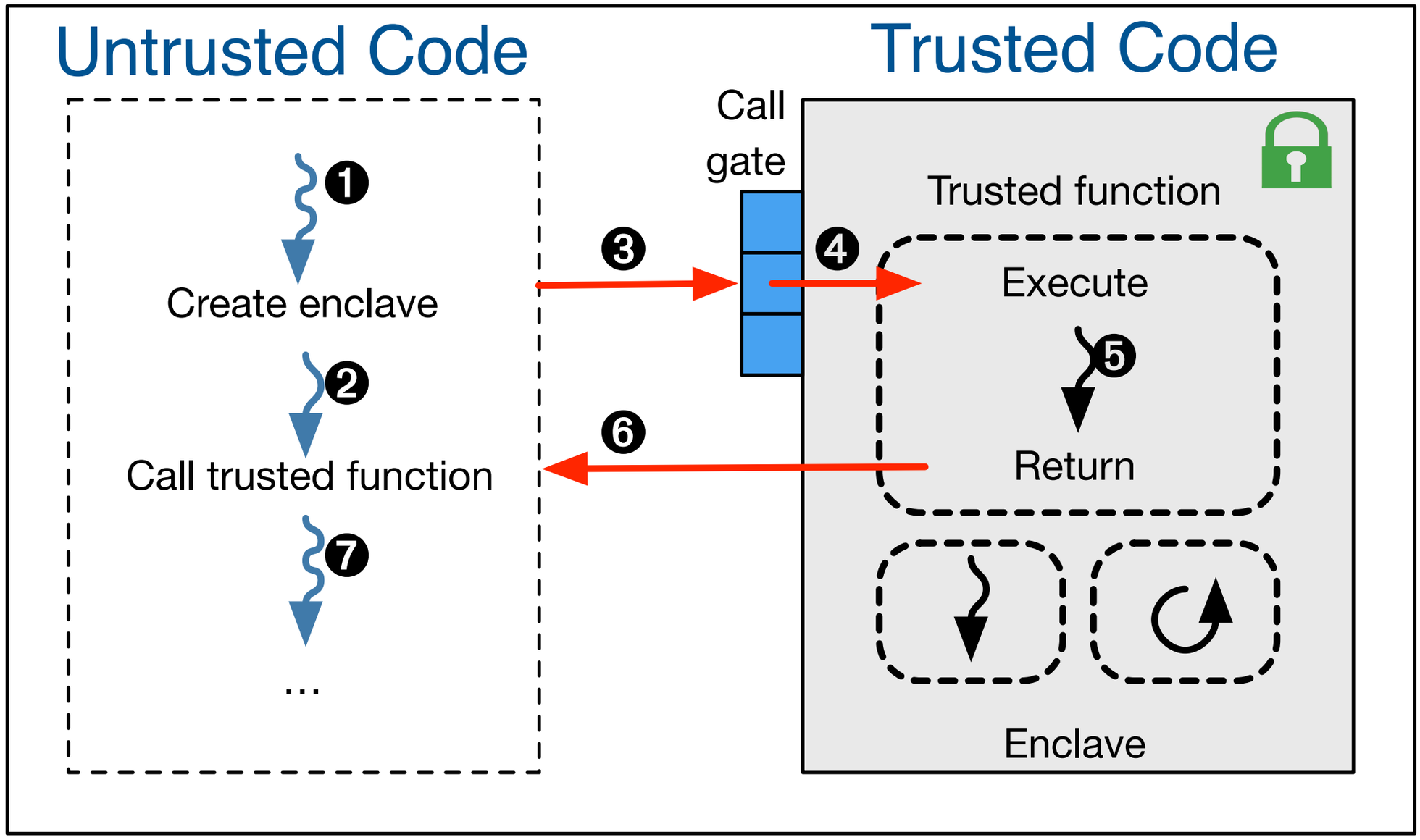
SGX hardware

Google Cloud

amazon web services™

Skylake products: https://ark.intel.com/products/codename/37572/Skylake

Secure processing with SGX



SGX hardware limitations

Enclave Page Cache:

- 128 MB
- used by enclave
- costly overflows

SGX hardware limitations

Enclave Page Cache:

- 128 MB
- used by enclave
- costly overflows

SGX hardware limitations

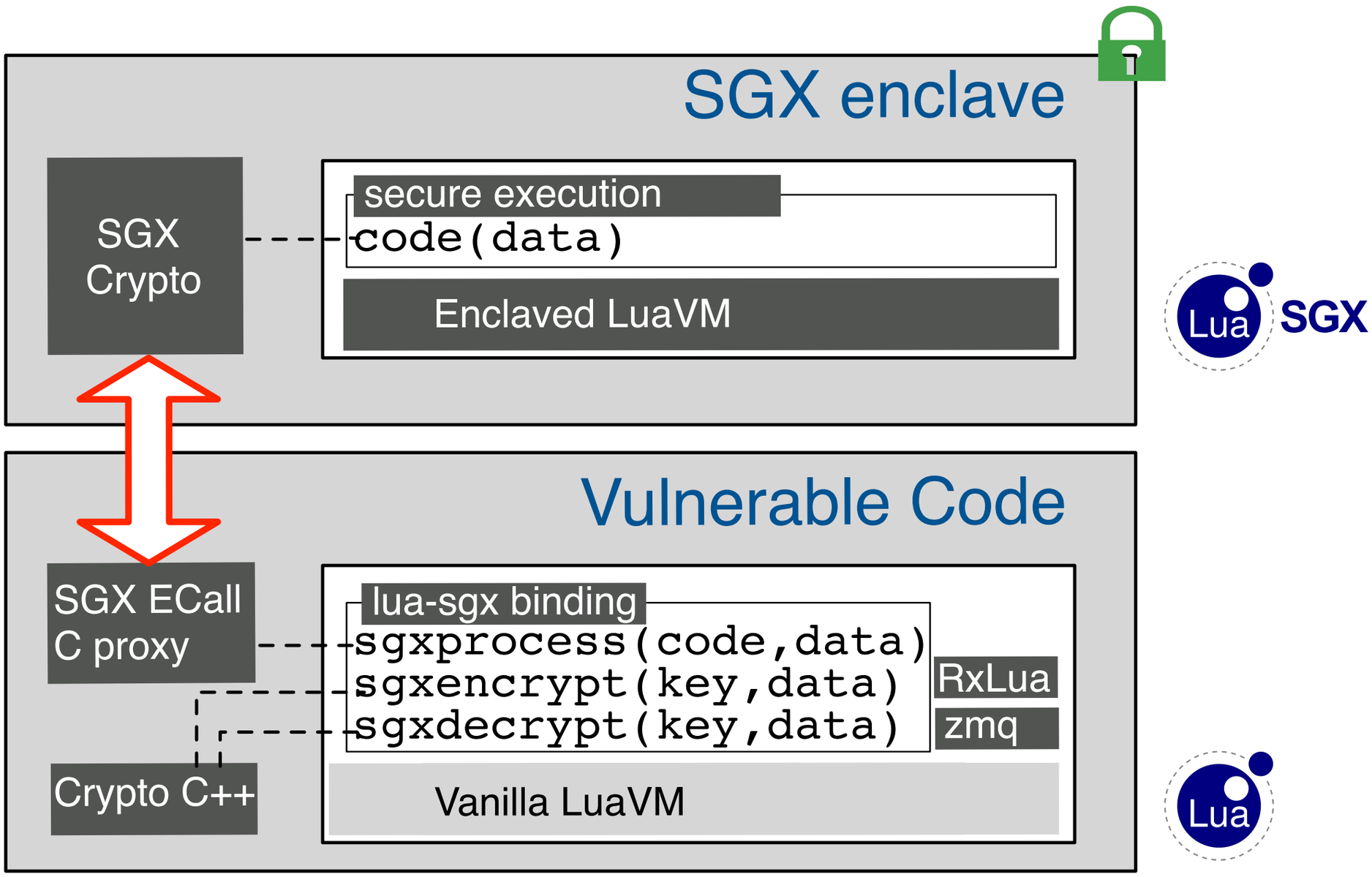
Enclave Page Cache:

- 128 MB
- used by enclave
- costly overflows

Lua: a lightweight multiplatform runtime

www.lua.org

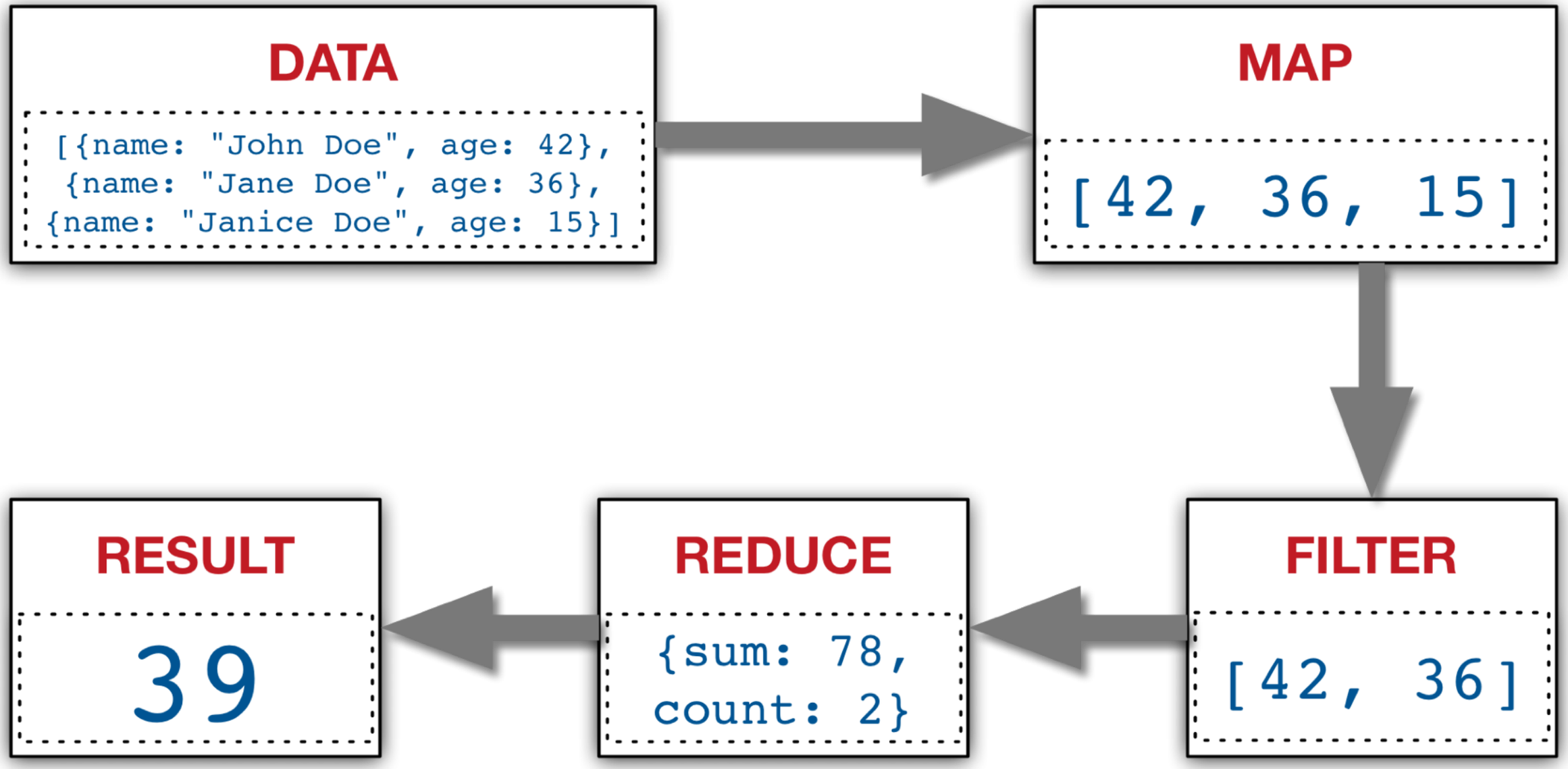
Integration between Lua and SGX



SecureStreams overview

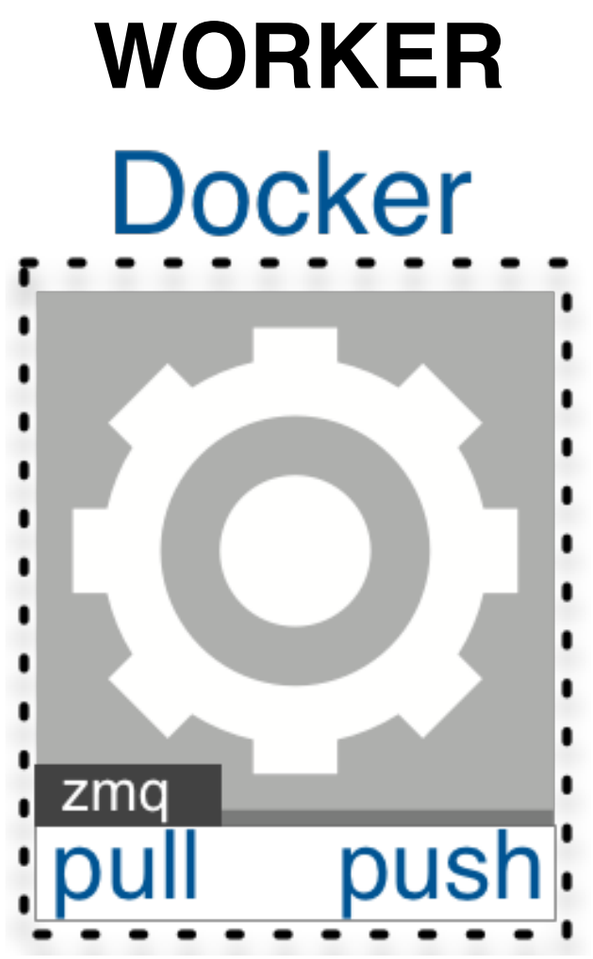
- Distributed stream processing
- Dataflow programming paradigm
- Deployment infrastructure abstraction
- Encrypted communication
- Secure processing in trusted hardware enclaves

SecureStreams overview



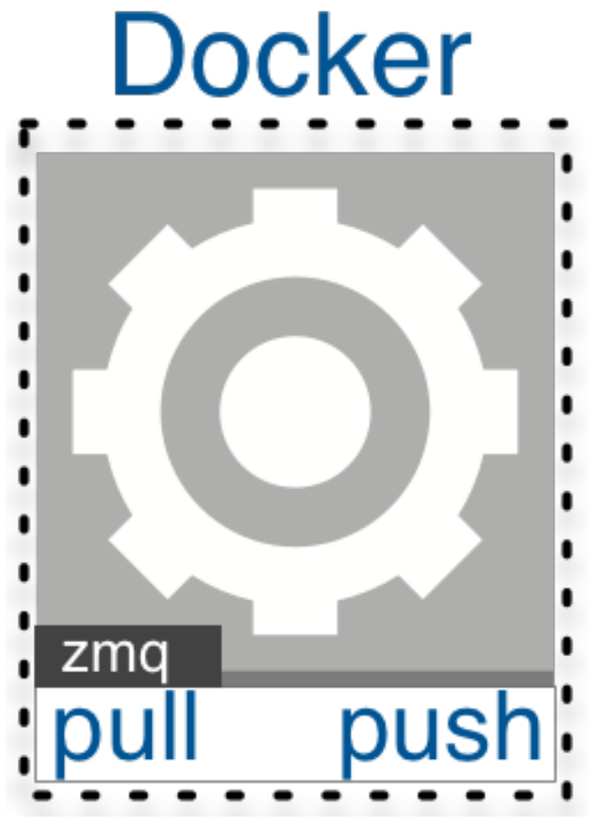
Architecture

Architecture

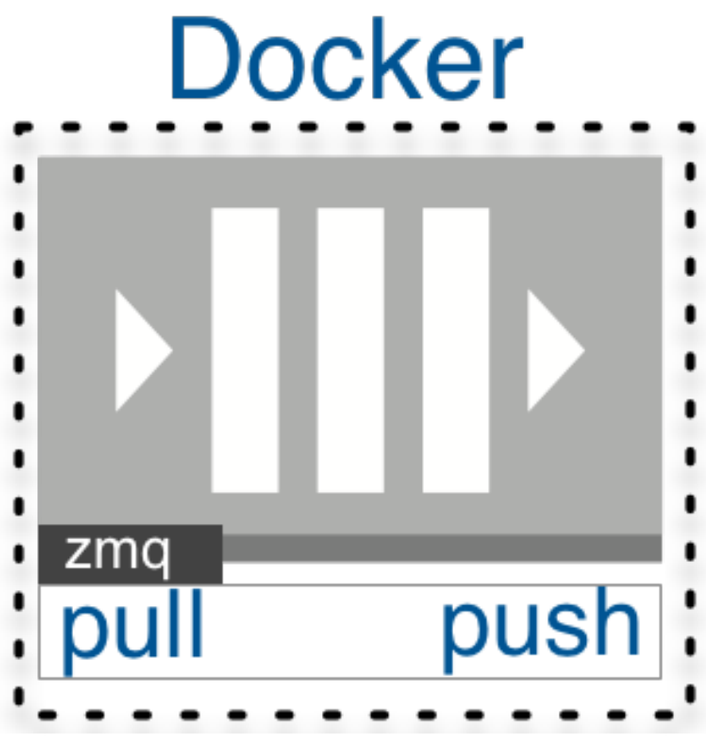


Architecture

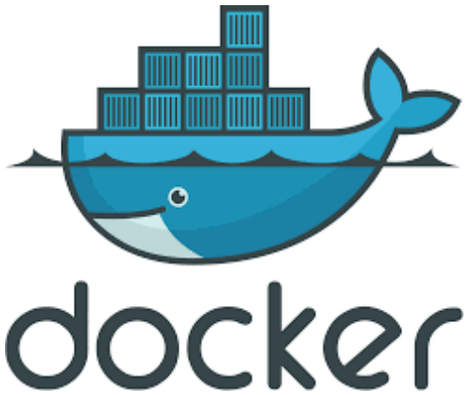
WORKER

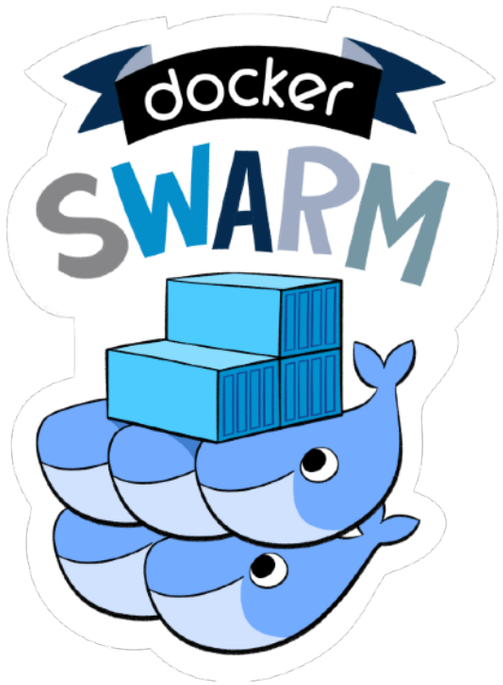


ROUTER

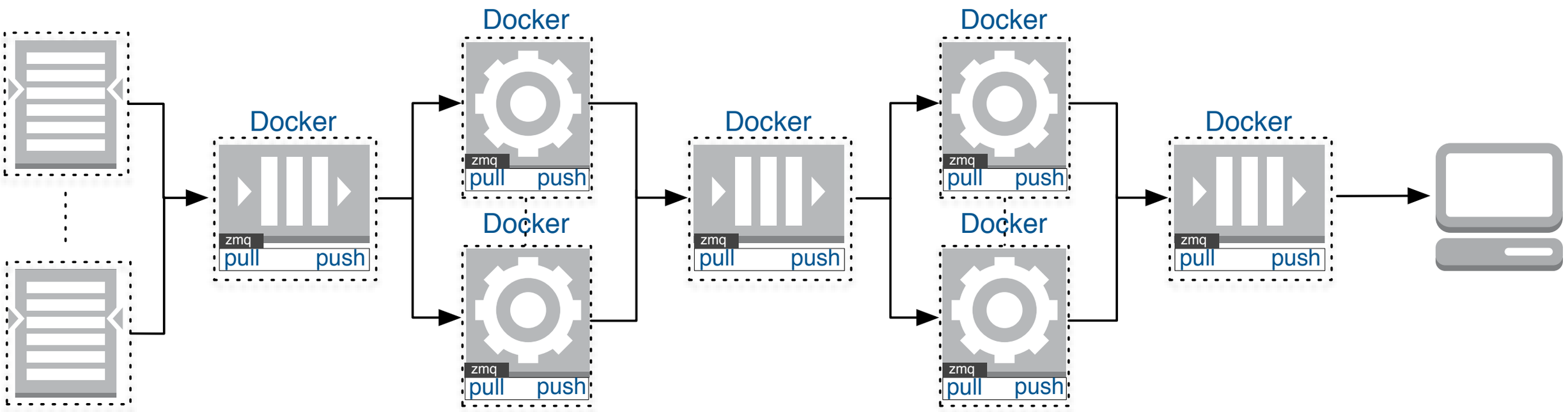


Architecture





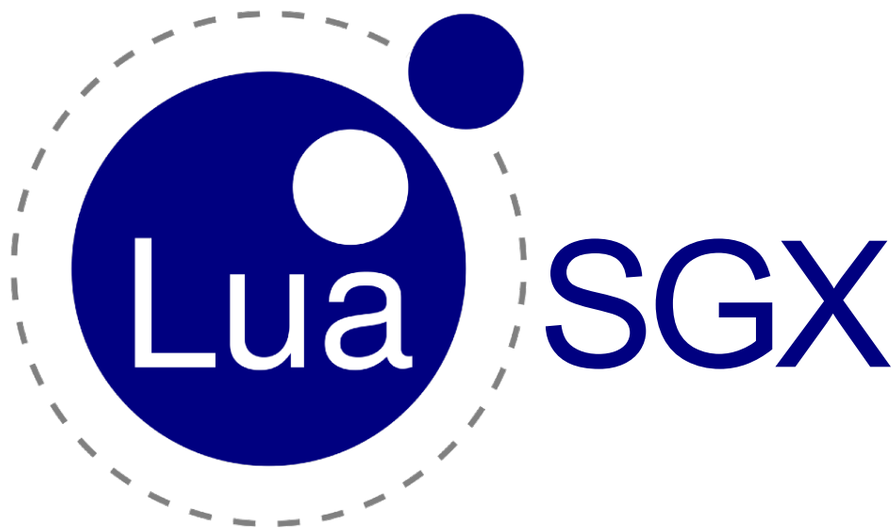
Architecture



Implementation

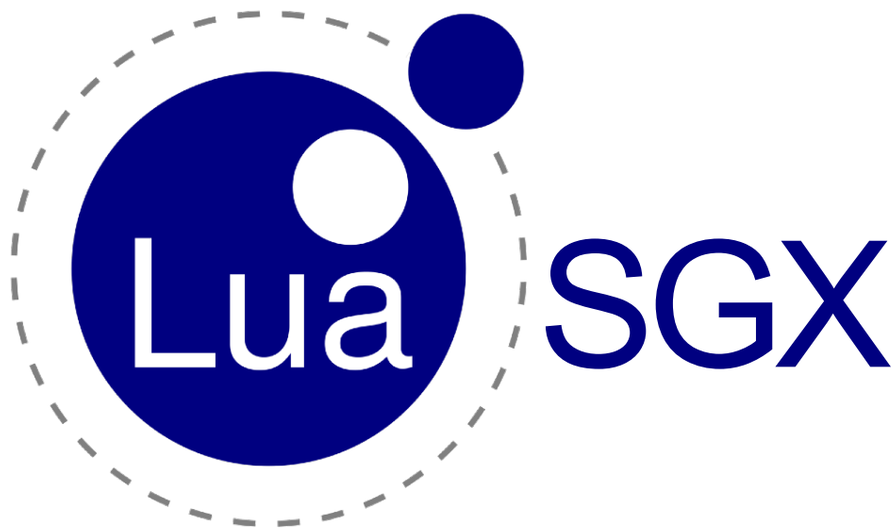
Implementation

Implementation



RxLua: https://github.com/bjornbytes/RxLua

Implementation



RxLua: https://github.com/bjornbytes/RxLua

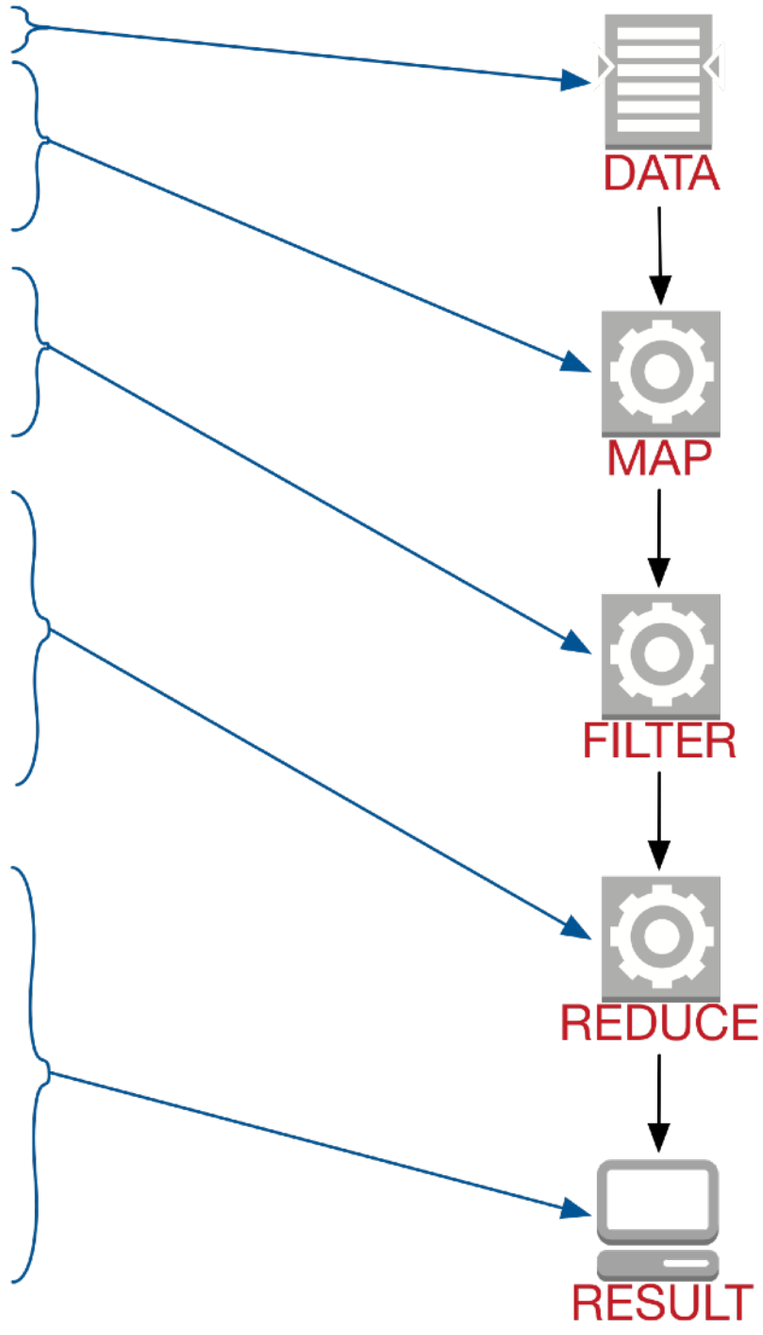
lzmq: https://github.com/zeromq/lzmq

Implementation

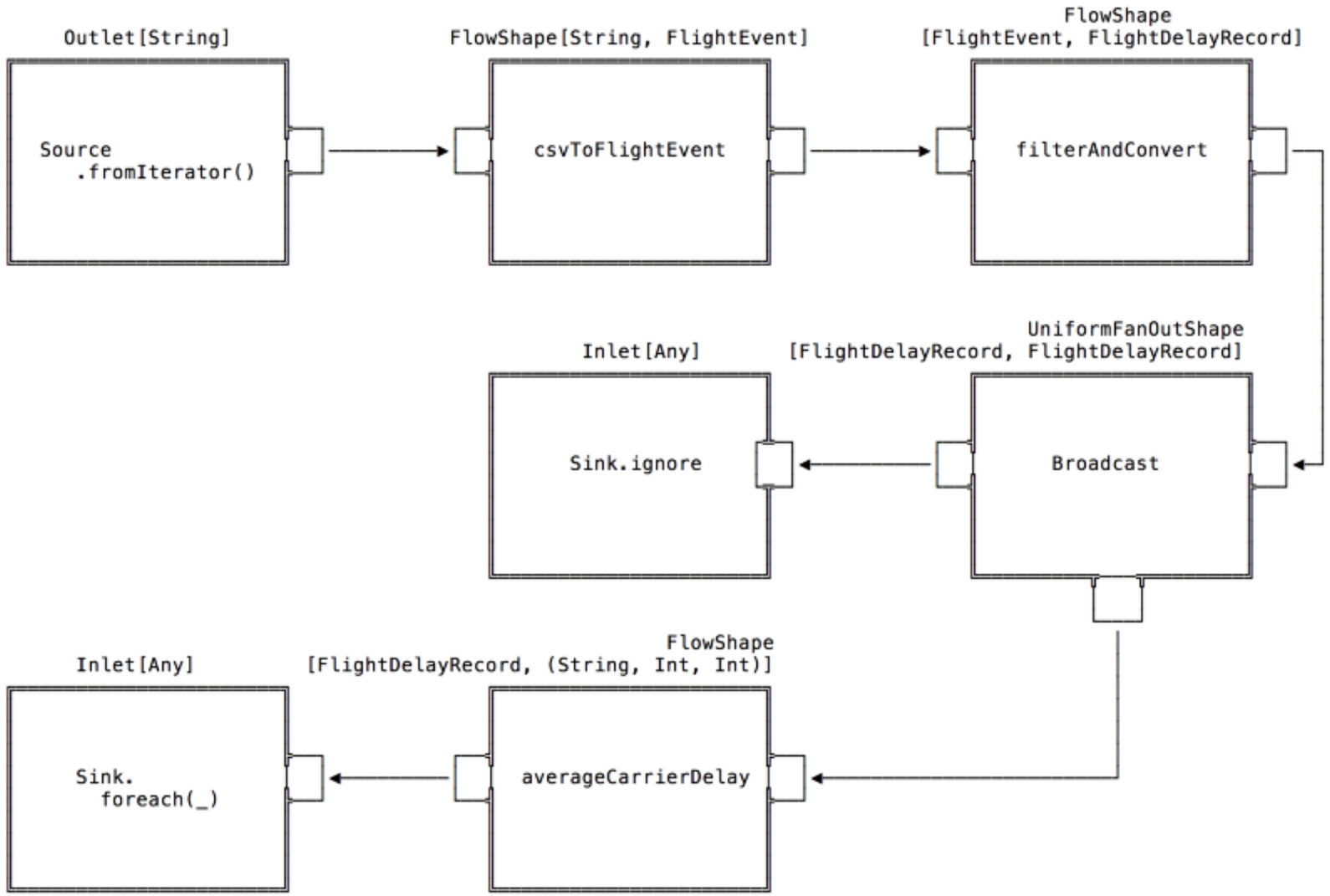
```
Rx.Observable.fromTable(people)
 :map(
   function(person)
     return person.age
   end
 )
 :filter(
   function(age)
     return age > 18
   end
 )
 :reduce(
   function(accumulator, age)
     accumulator[count] = (accumulator.count
       or 0) + 1
     accumulator[sum] = (accumulator.sum
       or 0) + age
     return accumulator
   end, {}
)
:subscribe(
   function(datas)
     print("Adult people average:",
       datas.sum / datas.count)
   end,
   function(err)
     print(err)
   end,
   function()
     print("Process complete!")
   end
)
```

Implementation

```
Rx.Observable.fromTable(people)
 :map(
   function(person)
     return person.age
   end
 )
 :filter(
   function(age)
     return age > 18
   end
 )
 :reduce(
   function(accumulator, age)
     accumulator[count] = (accumulator.count
       or 0) + 1
     accumulator[sum] = (accumulator.sum
       or 0) + age
     return accumulator
   end, {}
)
:subscribe(
   function(datas)
     print("Adult people average:",
       datas.sum / datas.count)
   end,
   function(err)
     print(err)
   end,
   function()
     print("Process complete!")
   end
)
```

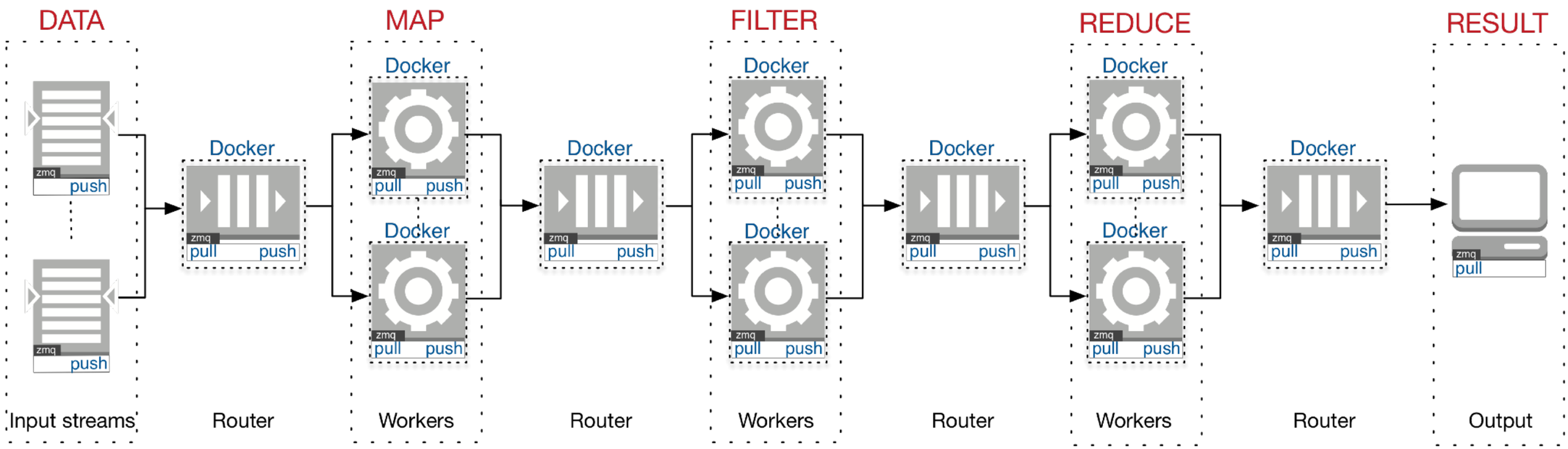


Preliminary evaluation



Source: https://blog.redelastic.com/diving-into-akka-streams-2770b3aeabb0

Preliminary evaluation



Preliminary evaluation

Dataset:

- American Bureau of Transportation Statistic
- http://stat-computing.org/dataexpo/2009/the-data.html

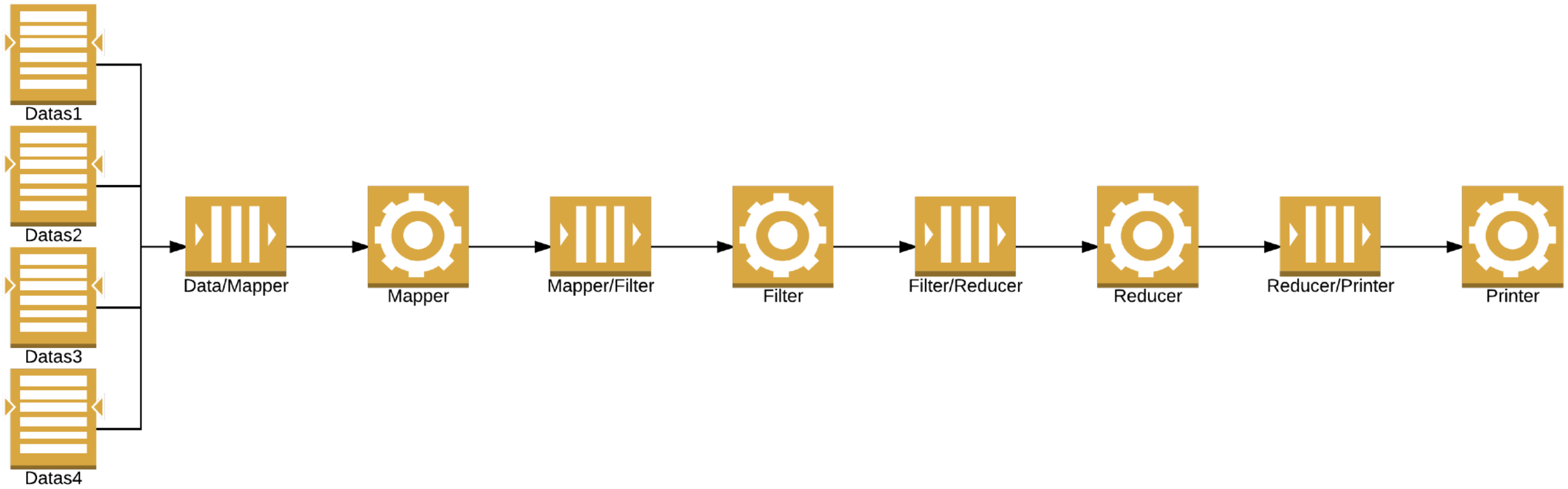
- 4 last years available: from 2005 to 2008
- 28 M of entries
- 2.73 GB of data

Preliminary evaluation

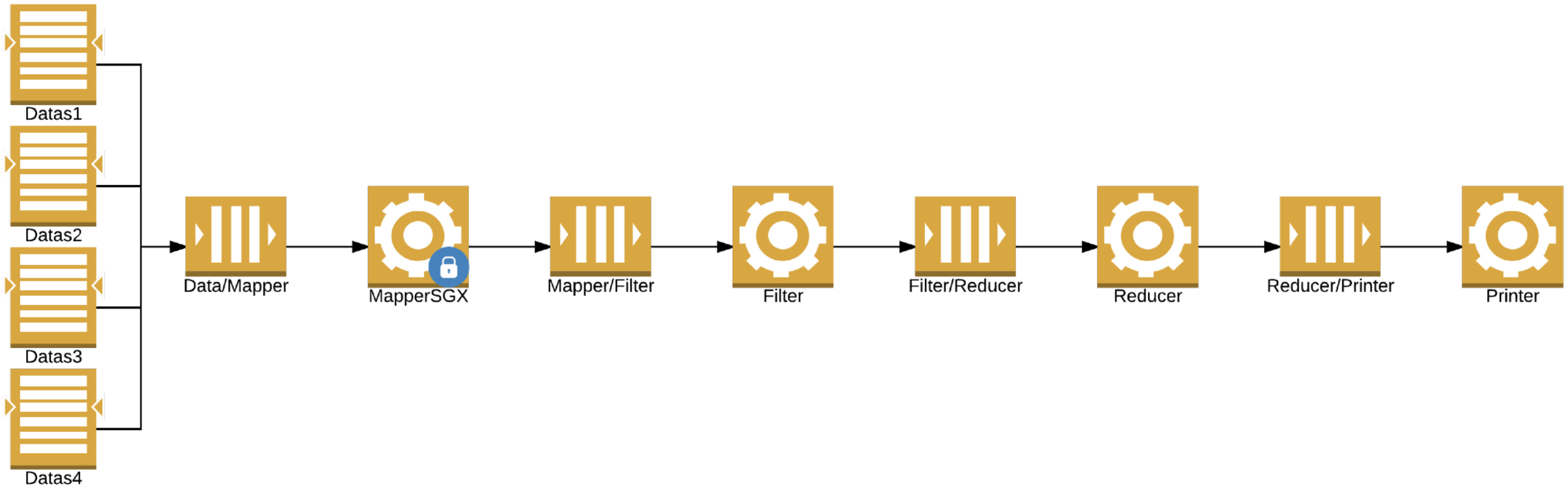
Evaluation settings:

- Cluster of 2 SGX machines 8CPU/8GB w/ Ubuntu 14.04.1 LTS
- Switched 1 Gbps network
- Docker (v. 1.13.0) & Docker Swarm (v. 1.2.5)

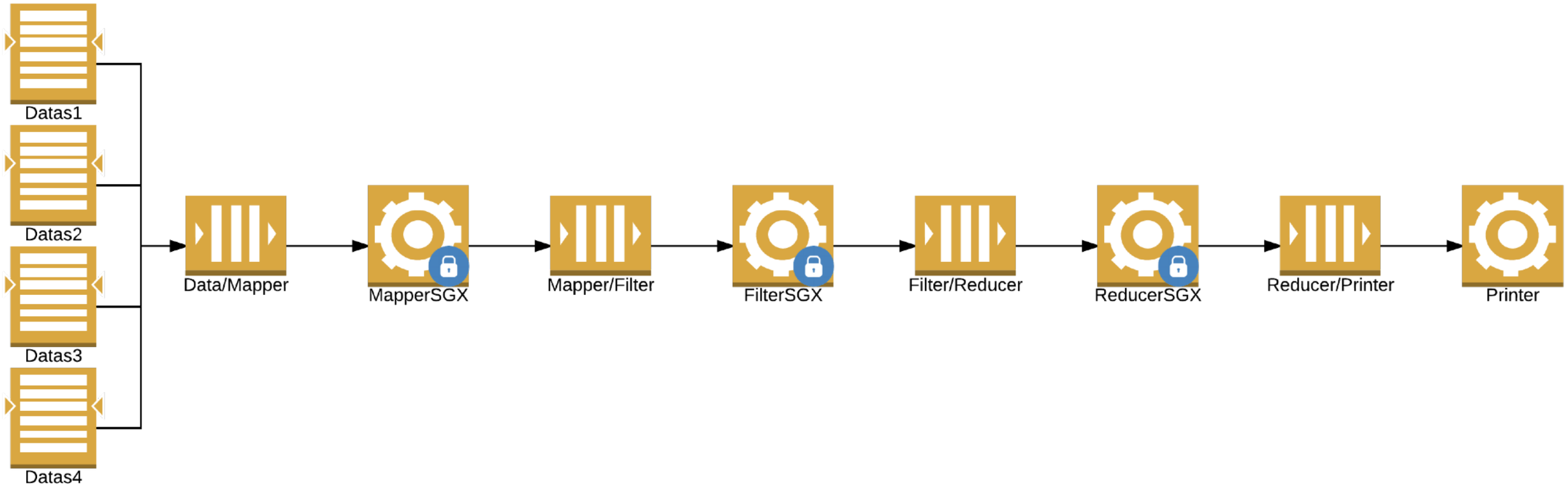
Preliminary evaluation



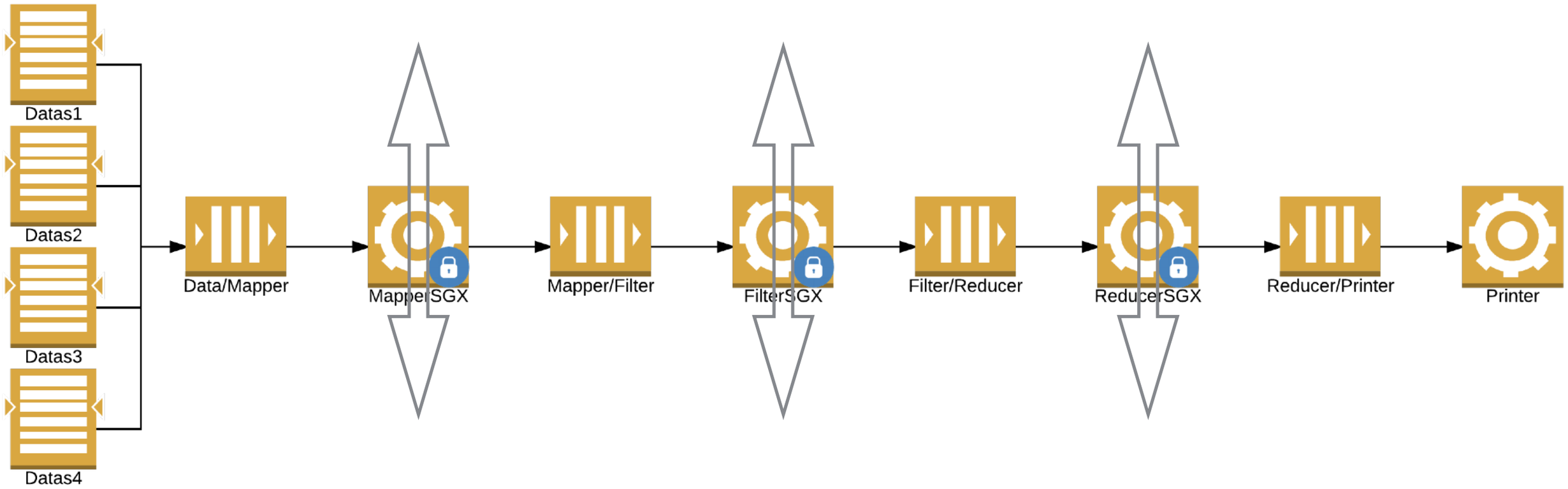
Preliminary evaluation



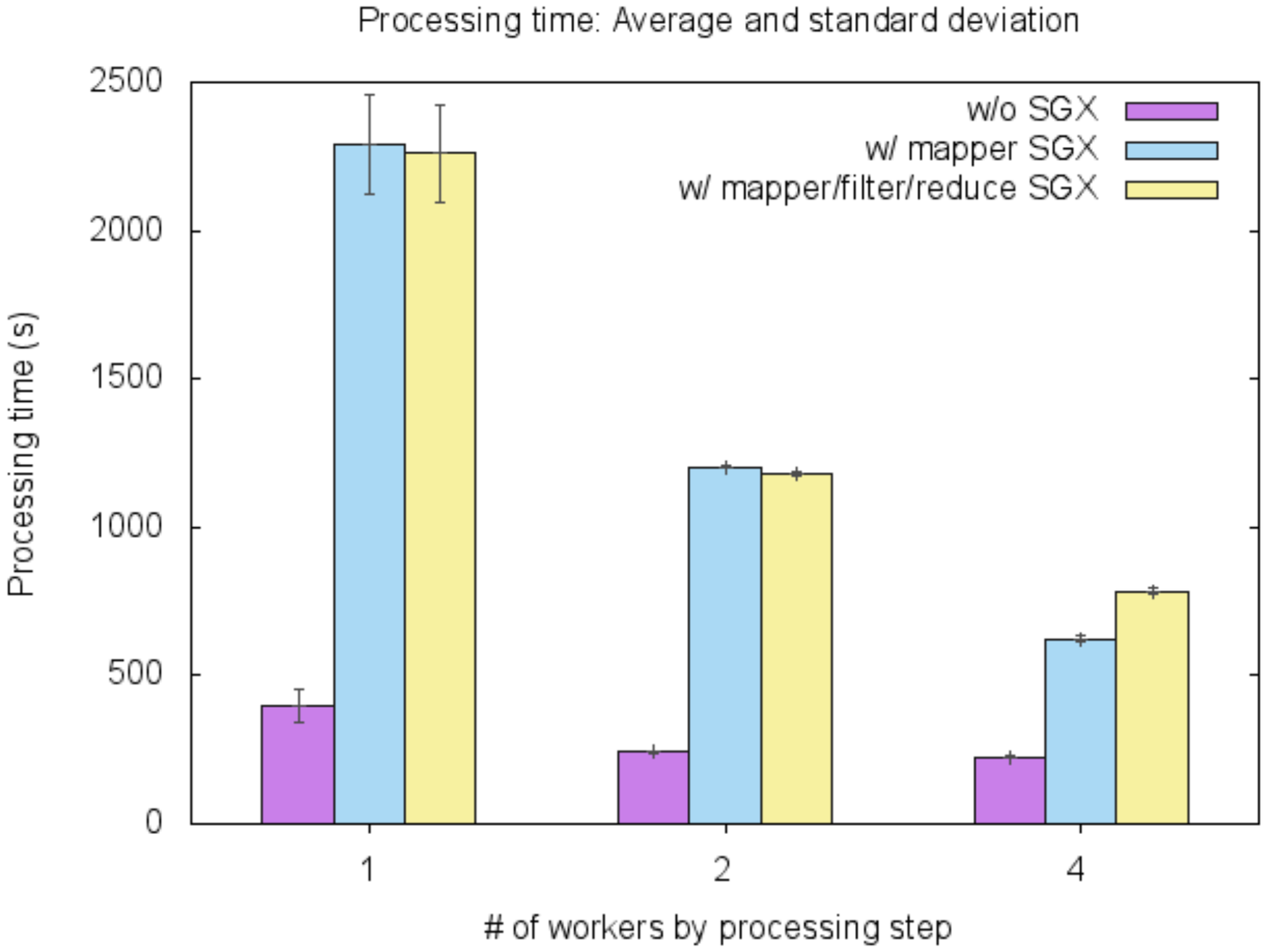
Preliminary evaluation



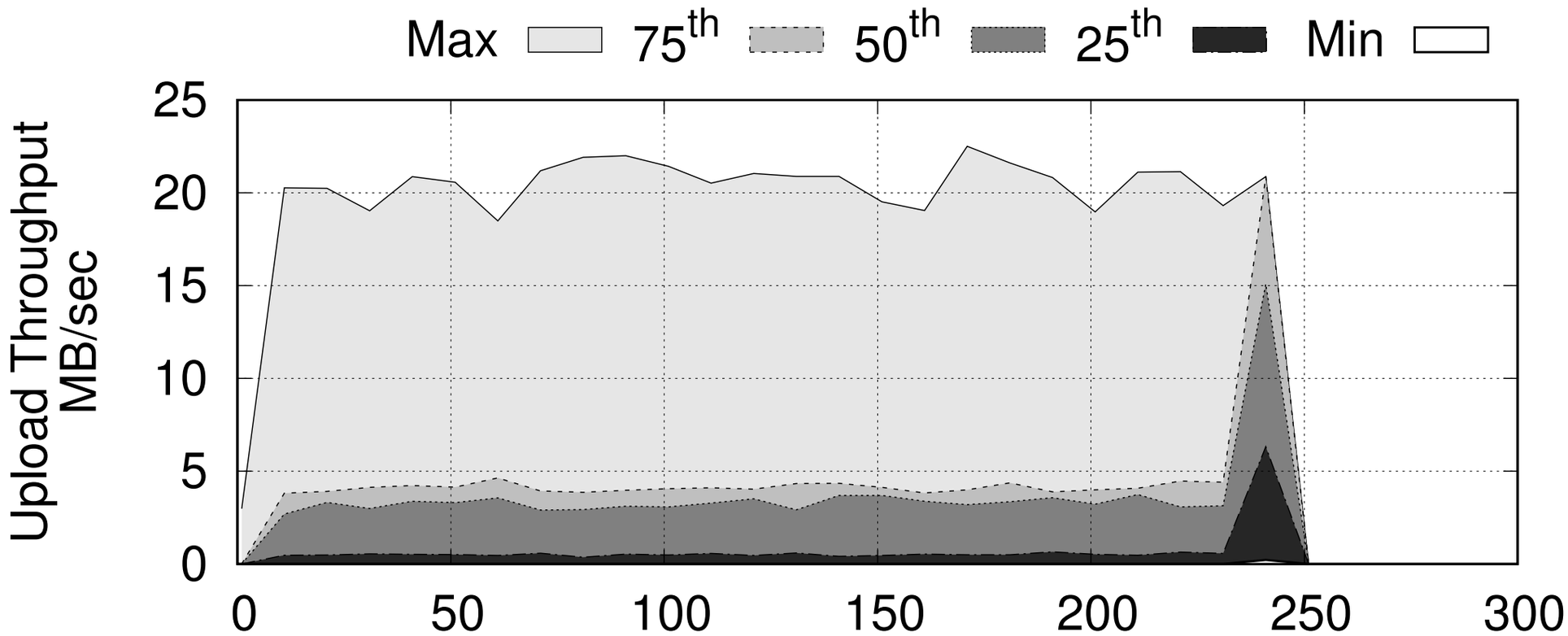
Preliminary evaluation



Preliminary evaluation

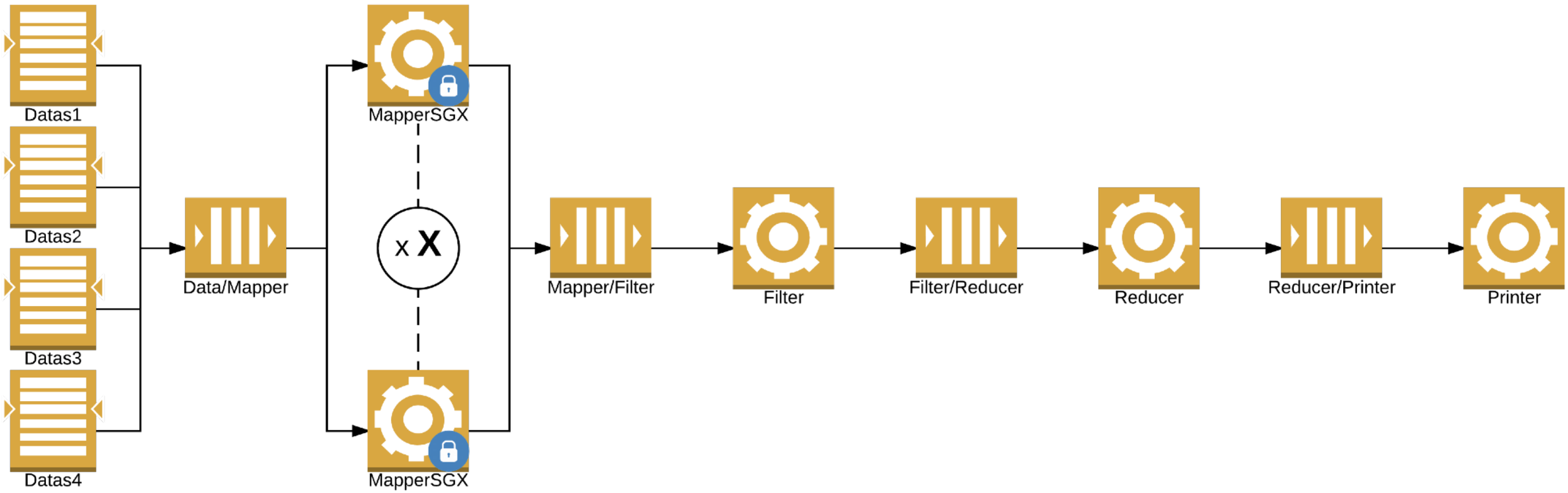


Preliminary evaluation

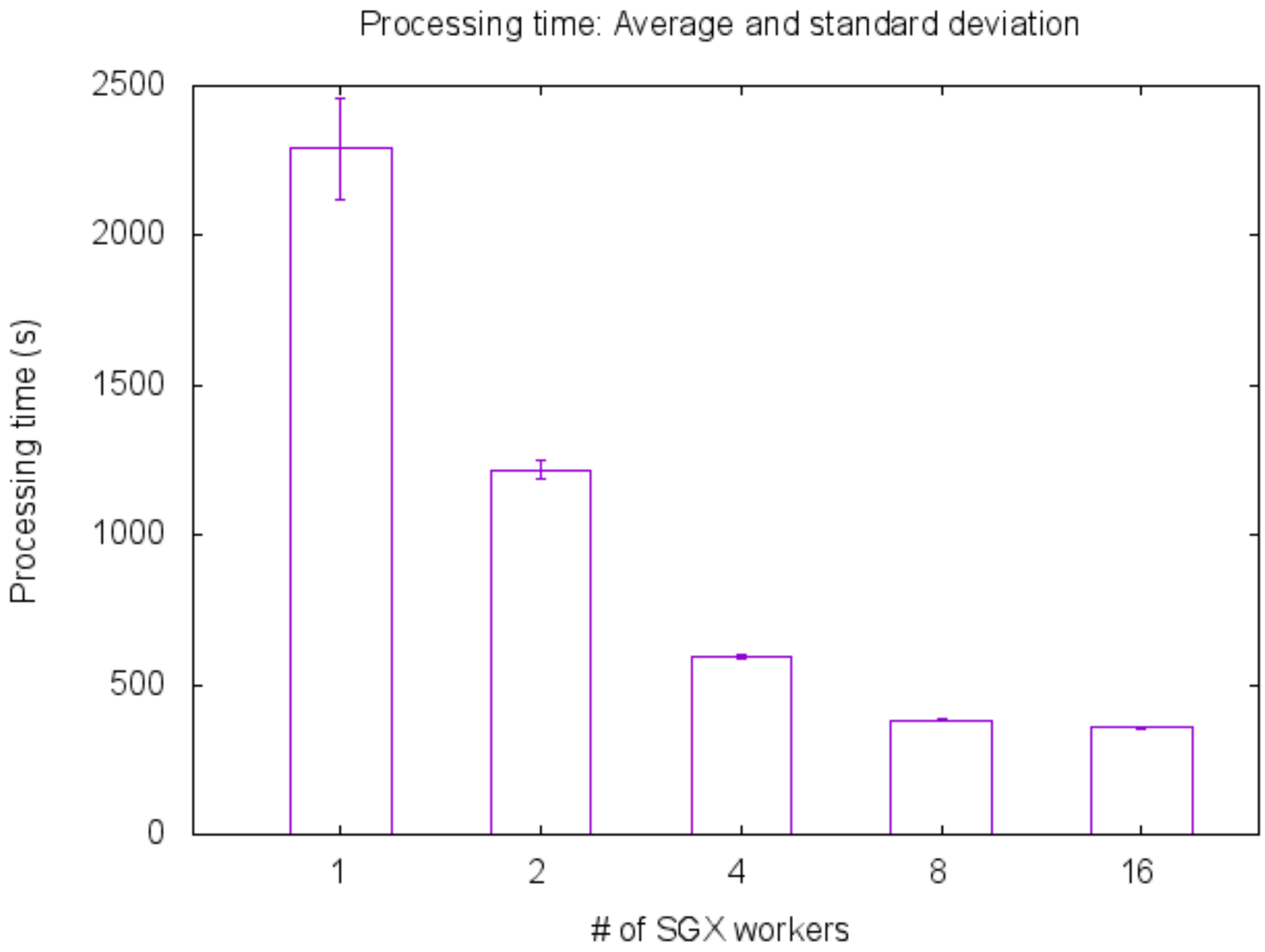


Throughput

Preliminary evaluation



Preliminary evaluation



Scalability

Conclusion & Future work

- ✓ Distributed stream processing
- ✓ Secure communication
- ✓ Secure processing in TEE

Conclusion & Future work

- ✓ Distributed stream processing
- ✓ Secure communication
- ✓ Secure processing in TEE

- ➡ Container deployment in full automation
- ➡ Dynamic scale at runtime
- ➡ Smart placement of containers
- ➡ Third-party Lua components embedding

Thanks for your attention