Drift

an imperative programming
environment for the cloud

#2 Implementation

Overview

- Drift Language
  - Concepts
  - Examples
- Drift Execution
  - Drift “FS”
  - Architecture
  - Error Model

Recap

- Want: ‘language of the system’
- Workflow languages one possible domain
  - black box tasks, …
- Bash: system-view coordination
  - black box tasks, immediate feedback REPL, …
  - Problem: FS → shared mutable state
- Functional ❤ Distributed
  - Cuneiform: functional, distributed, …

Recap

Bash	Functional
&	- (lazy)
|	function composition
>, >>	name binding
<	-
$	eval

Drift Language

Can we build an *imperative*, *stateful*, *interpreted* language for distributed (micro) service coordination ?

- Need ‘state’ to be stateful on
- Essence: data + services

“Do this on that thing over here.”

“Now do this on that and put it over there”

Drift Language

- How do we (humans) ‘interact’ with data?

  → need ‘names’ to identify and retrieve our data

  → need ‘names’ to give data *meaning*

- *Names* are at the center of programming!

  → only names and services

- Names best be hierarchical → Namespaces

  → Names, Namespaces and Services

Drift Language

```
.> Cat foo.txt
   Lorem ipsum dolor sit amet, consetetur
   sadipscing elitr, sed diam nonumy eirmod
.>

.> foo = Cat foo.txt
.>

.> ls
   foo

.> $foo
   Lorem ipsum dolor sit amet, consetetur
   sadipscing elitr, sed diam nonumy eirmod
   …
```

Drift Language

```
.> wordcount_h  = Wc hamlet.txt
.> wordcount_mb = Wc macbeth.txt
.> Max wordcount_h wordcount_mb
   wordcount_h
```

```
.> a = A data.csv | B | C
```

Pipe cut into multiple commands

```
.> a = A in1.data in2.data
.> b = B a | C
.> a = G homework.txt
```

Drift Language

```
.> wordcount_h  = Wc hamlet.txt
.> wordcount_mb = Wc macbeth.txt
.> Max wordcount_h wordcount_mb
   wordcount_h
```

```
.> a = A data.csv | B | C
```

Pipe cut into multiple commands

```
.> a = A in1.data in2.data
.> b = B a | C
.> a = G homework.txt
```

Value of ‘a’ is a *value over time* – State

Drift Language

```
.> wordcount_h  = Wc hamlet.txt
.> wordcount_mb = Wc macbeth.txt
.> Max wordcount_h wordcount_mb
   wordcount_h
```

```
.> a = A data.csv | B | C
```

Pipe cut into multiple commands

```
.> a = A in1.data in2.data
.> b = B a | C
.> a = G homework.txt
```

B ‘captures’ the value of a *at that time* – Closure

Drift Language

```
.> import my.tar
.> res/ = Untar* my.tar | *FormatCheck txt

.> ls
   my.tar
   res/

.> $res/
   c1.txt
   c2.txt

.> wordcount = Wc res/c1.txt
```

ls, cd, rm are language keywords

Drift Language

So far:

- no arithmetic
- no conditionals
- not turing-complete! (hopefully)
- tiny
- very abstract
- very few constructs

Drift Execution

Front End:

- own shell implementation
- parser generated by ANTLR 4 then customized

Back End:

- own worker implementation
- RabbitMQ and Kafka as data and signal queues
- Mesos + Marathon for fault-tolerance (Workers)

Drift Execution

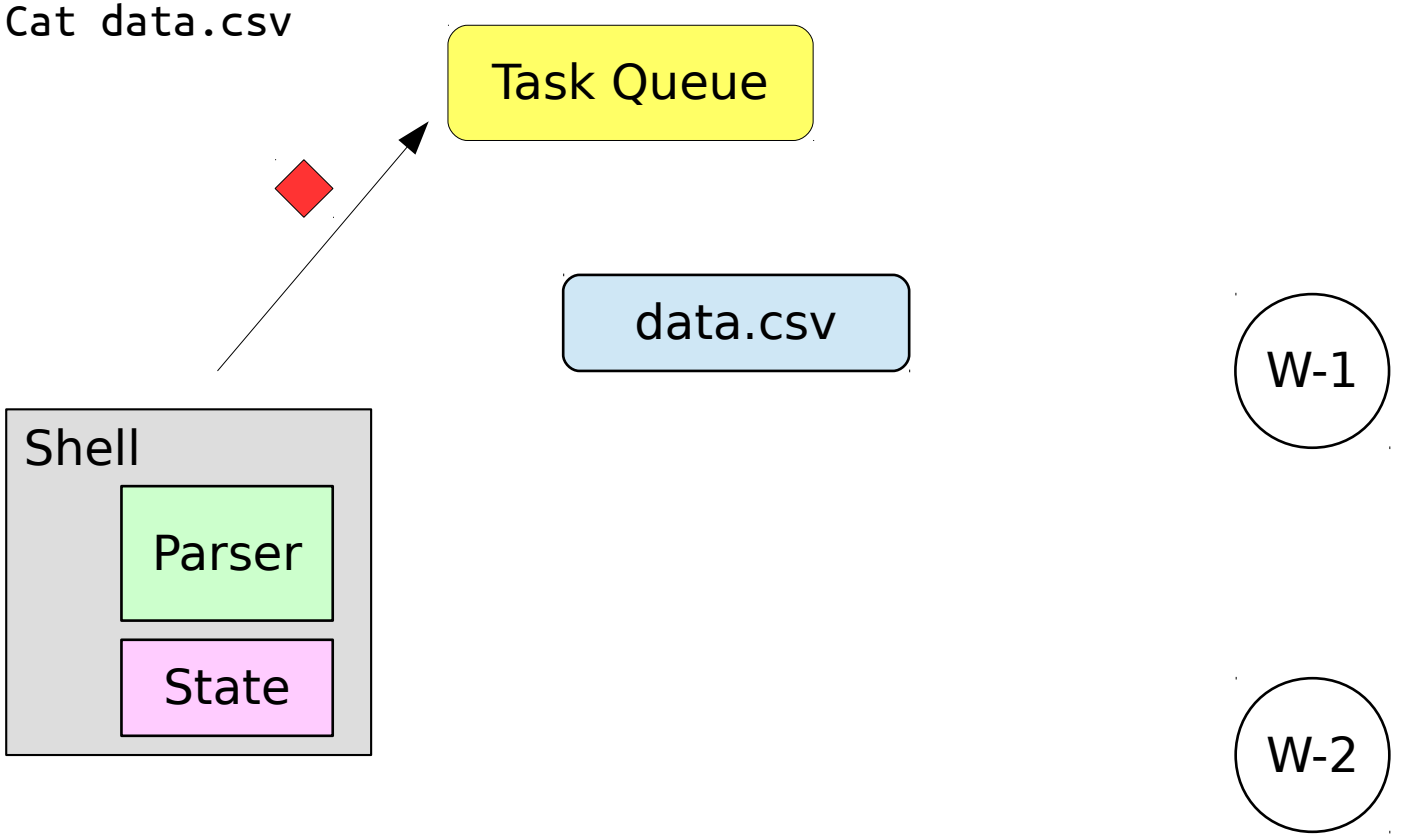
System Properties

- Tasks are deterministic
- only get scheduled / executed exactly once

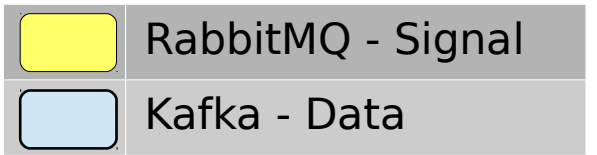
Basic Operation

Drift Execution

```
.> foo = Cat data.csv
```

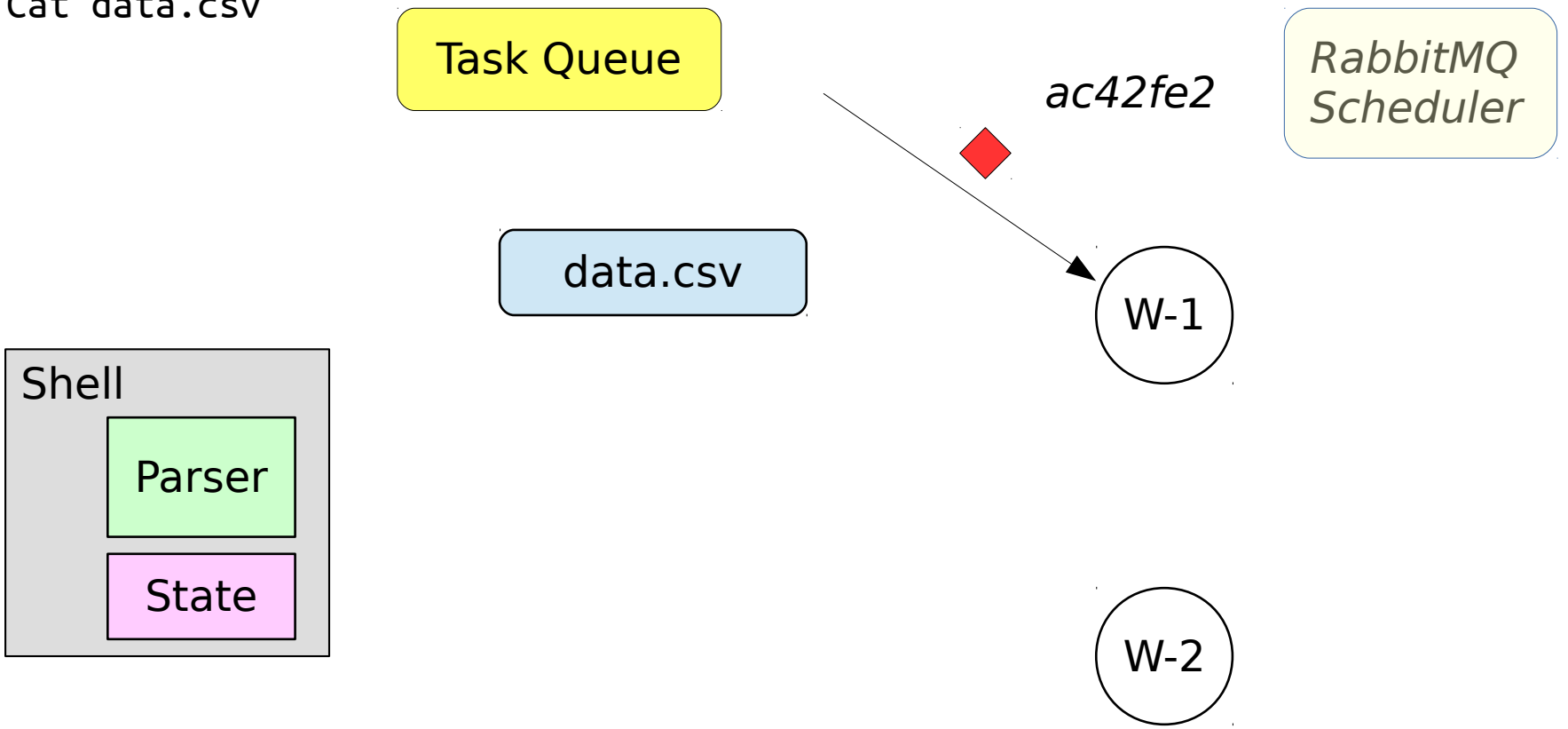


Name	Queue
data.csv	data.csv
foo	*ac42fe2*

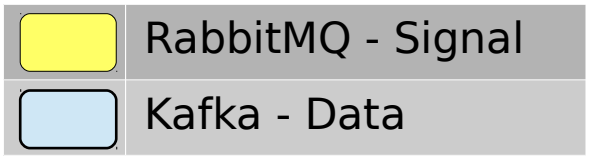


Drift Execution

```
.> foo = Cat data.csv
```

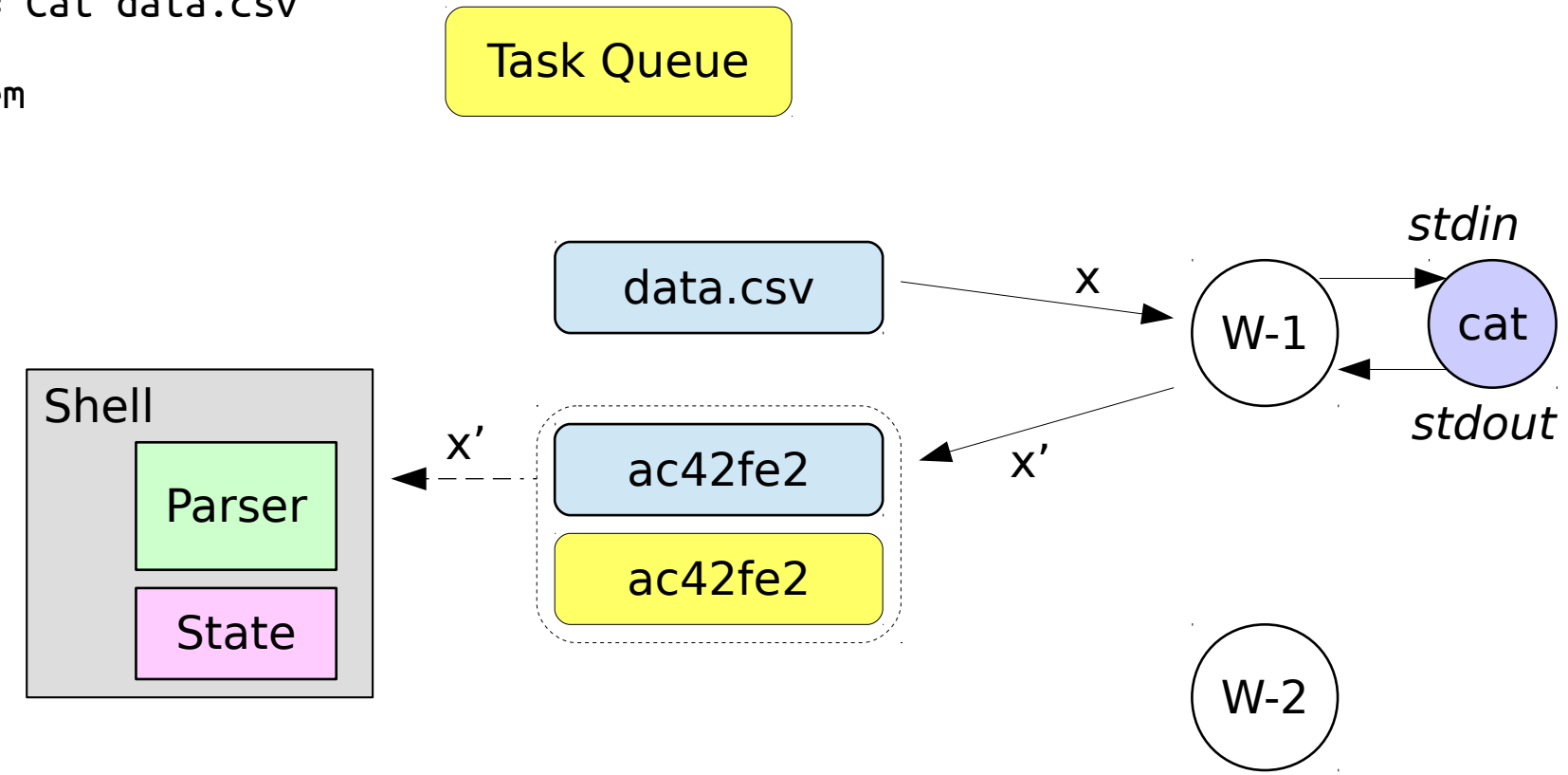


Name	Queue
data.csv	data.csv
foo	*ac42fe2*



Drift Execution

```
.> foo = Cat data.csv
.> $foo
    Lorem
```



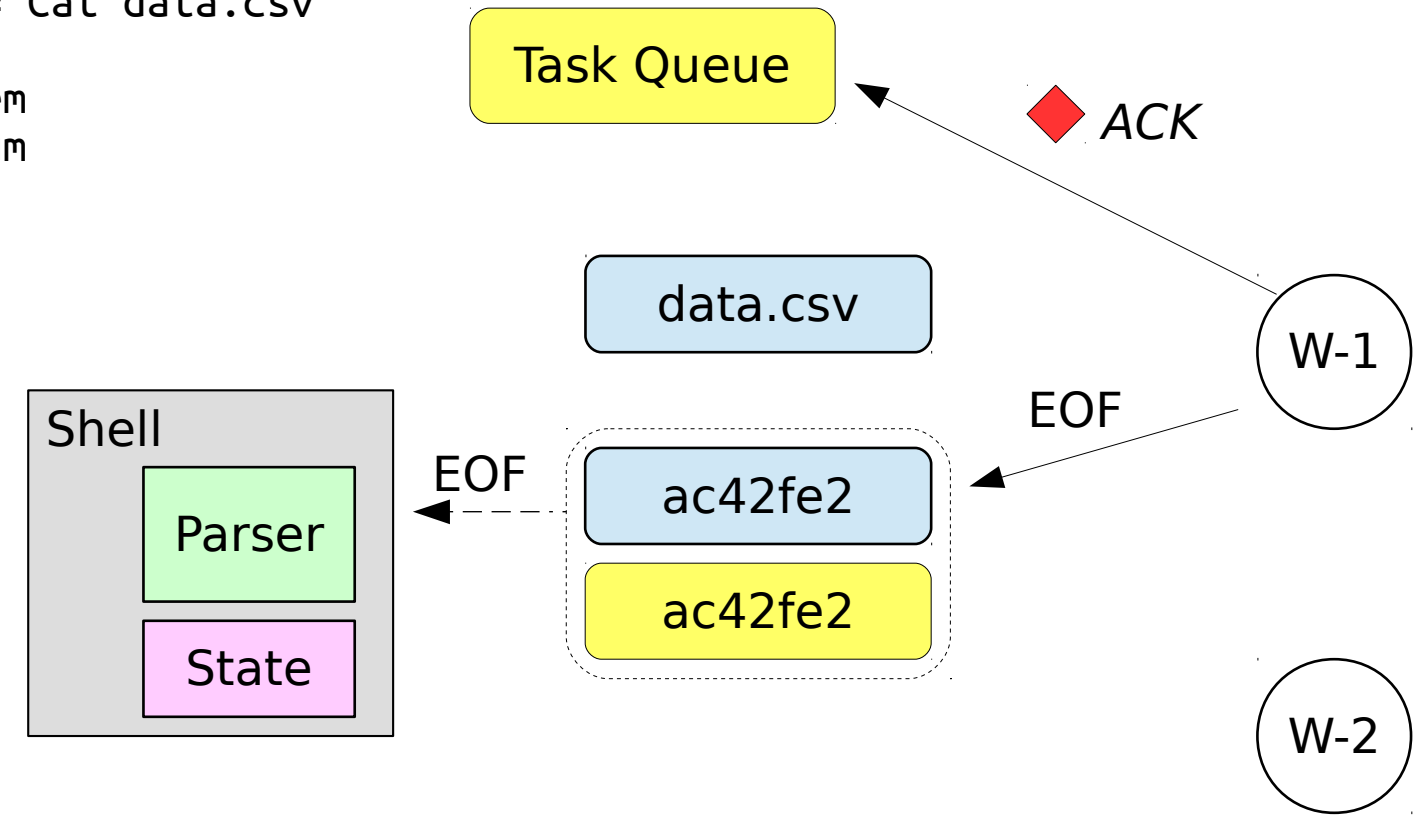
Name	Queue
data.csv	data.csv
foo	*ac42fe2*

RabbitMQ - Signal

Kafka - Data

Drift Execution

```
.> foo = Cat data.csv
.> $foo
    Lorem
    ipsum
```



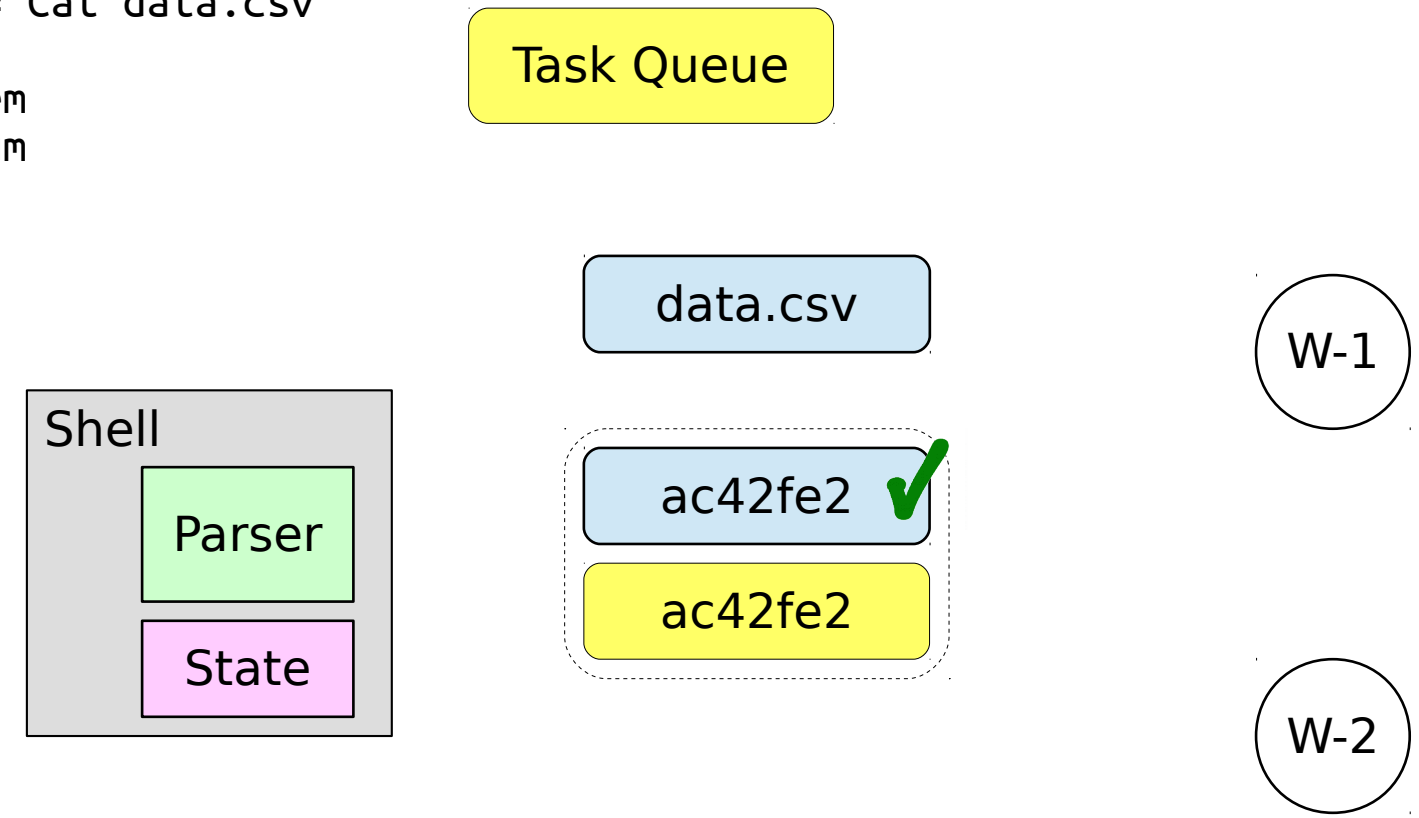
Name	Queue
data.csv	data.csv
foo	*ac42fe2*

RabbitMQ - Signal

Kafka - Data

Drift Execution

```
.> foo = Cat data.csv
.> $foo
    Lorem
    ipsum
.>
```



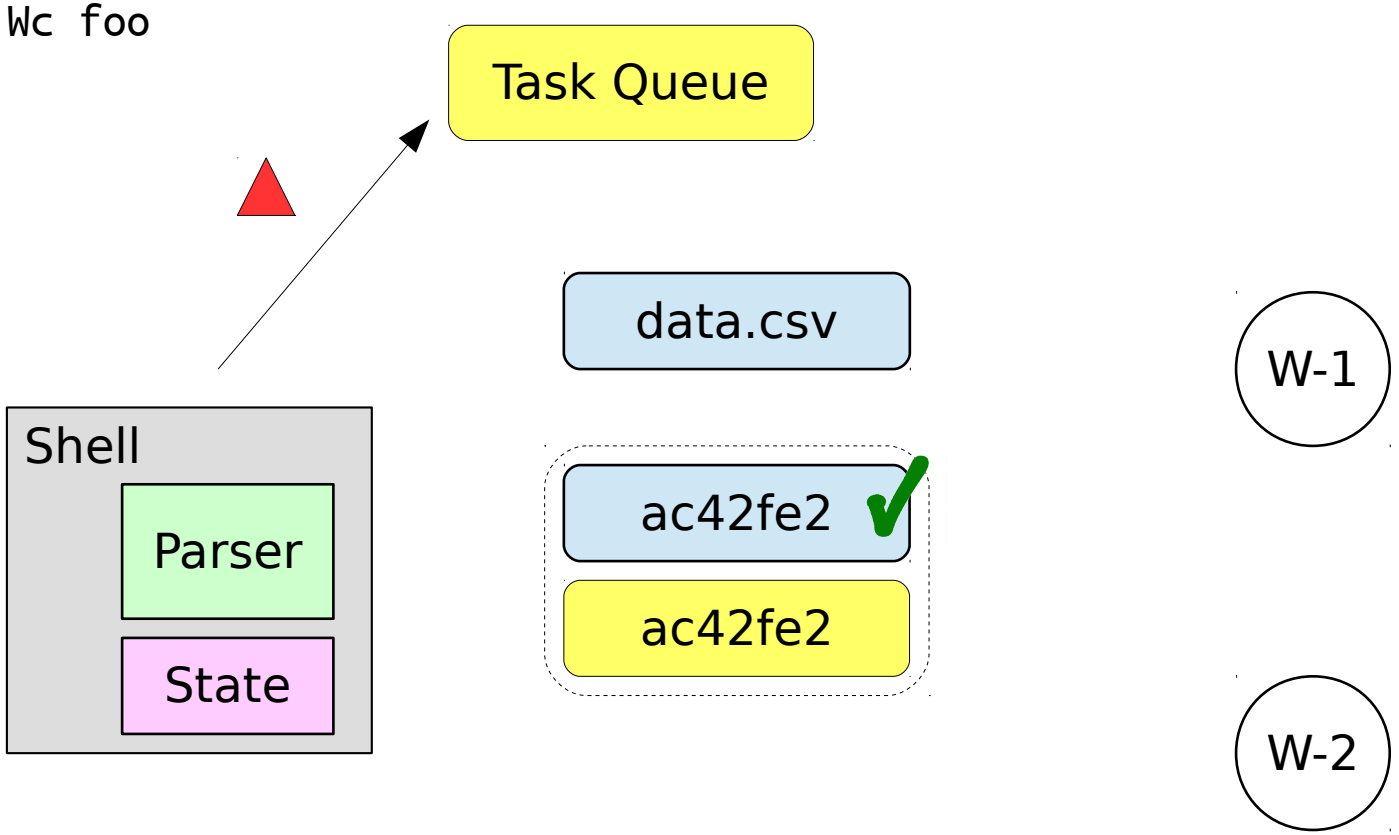
Name	Queue
data.csv	data.csv
foo	*ac42fe2*

RabbitMQ - Signal

Kafka - Data

Drift Execution

```
.> bar = Wc foo
```



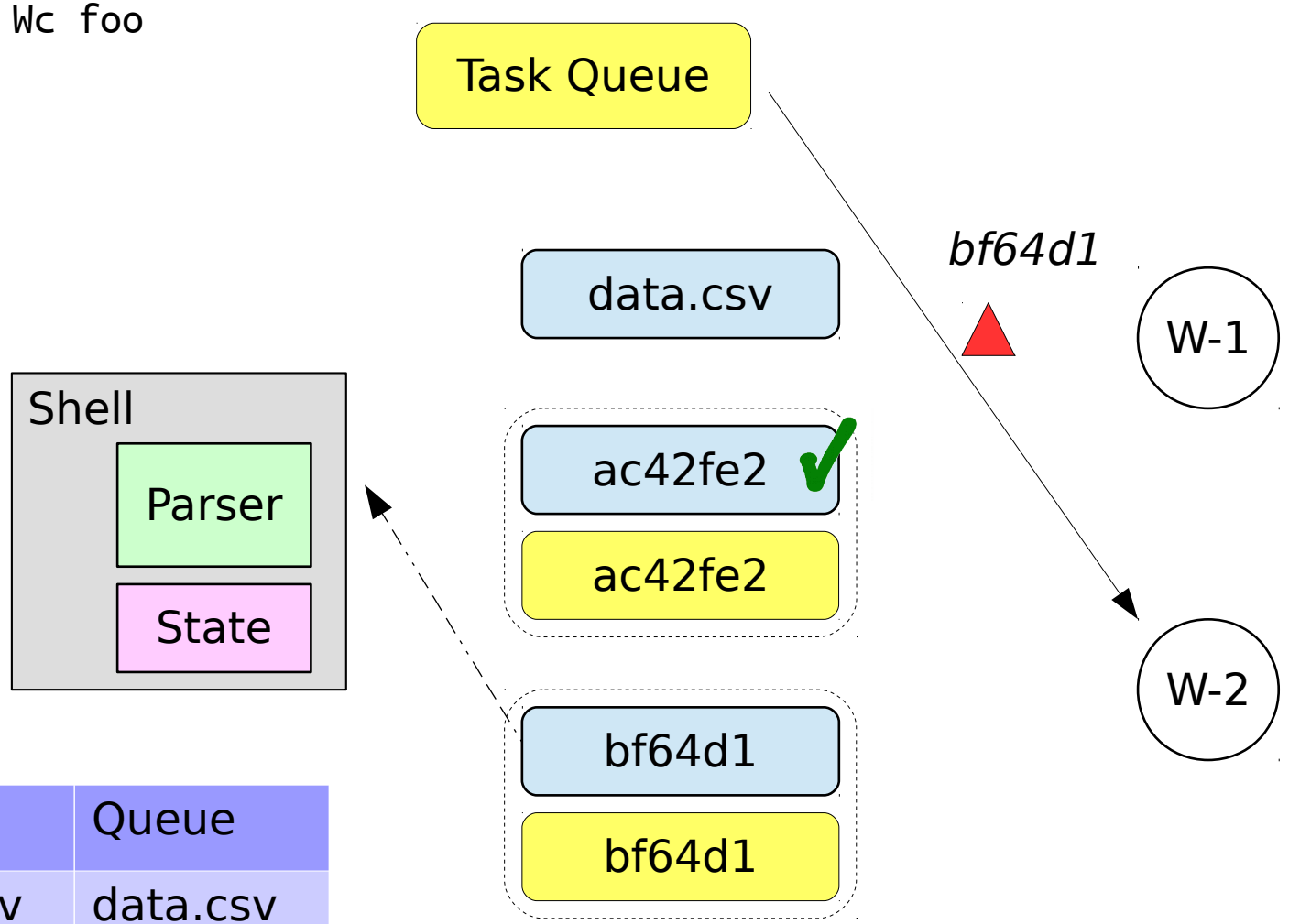
Name	Queue
data.csv	data.csv
foo	*ac42fe2*

RabbitMQ - Signal

Kafka - Data

Drift Execution

```
.> bar = Wc foo
.> $bar
    ...
```



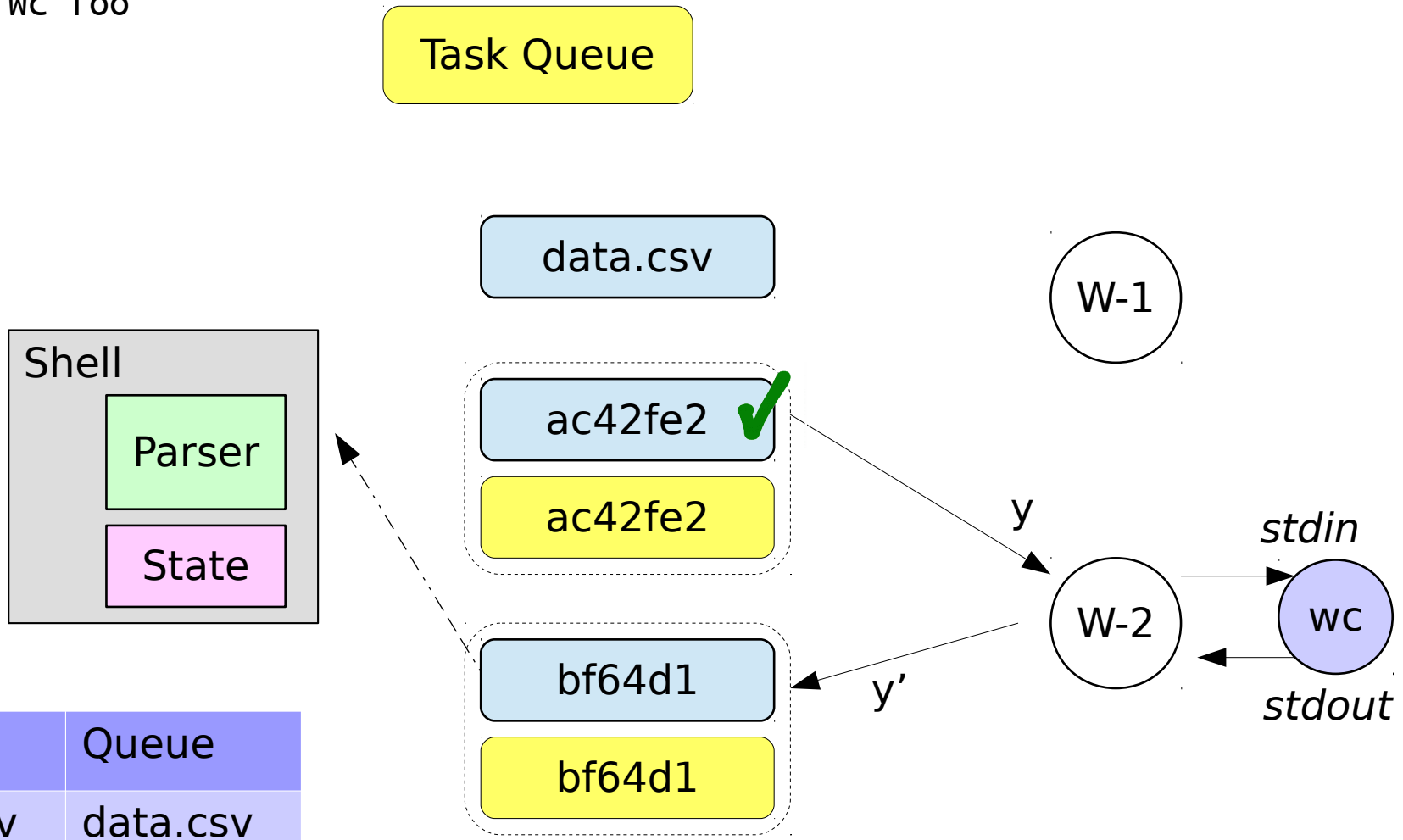
Name	Queue
data.csv	data.csv
foo	*ac42fe2*
bar	*bf64d1*

RabbitMQ - Signal

Kafka - Data

Drift Execution

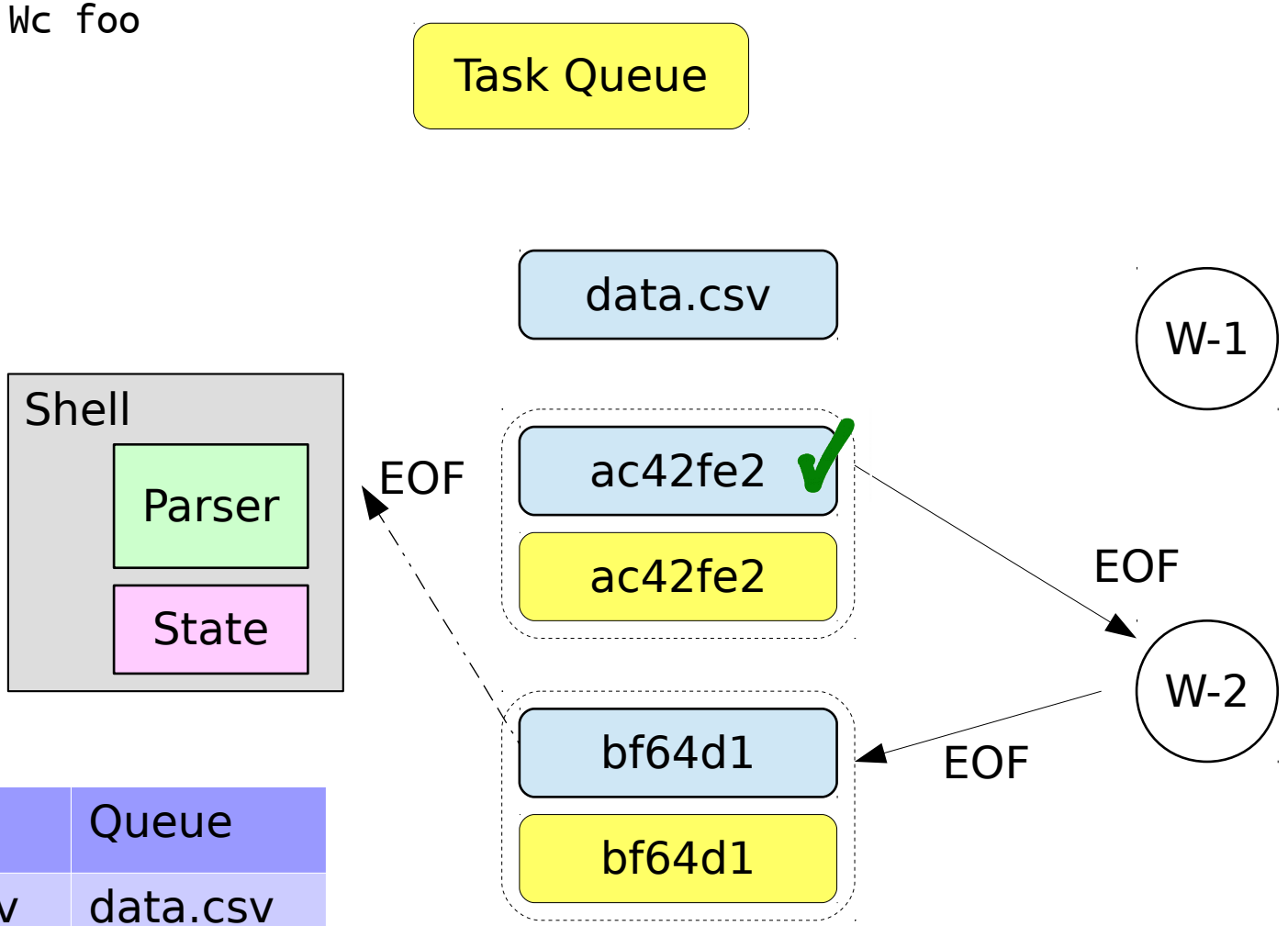
```
.> bar = Wc foo
.> $bar
    2
```



Name	Queue
data.csv	data.csv
foo	*ac42fe2*
bar	*bf64d1*

Drift Execution

```
.> bar = Wc foo
.> $bar
    2
.>
```



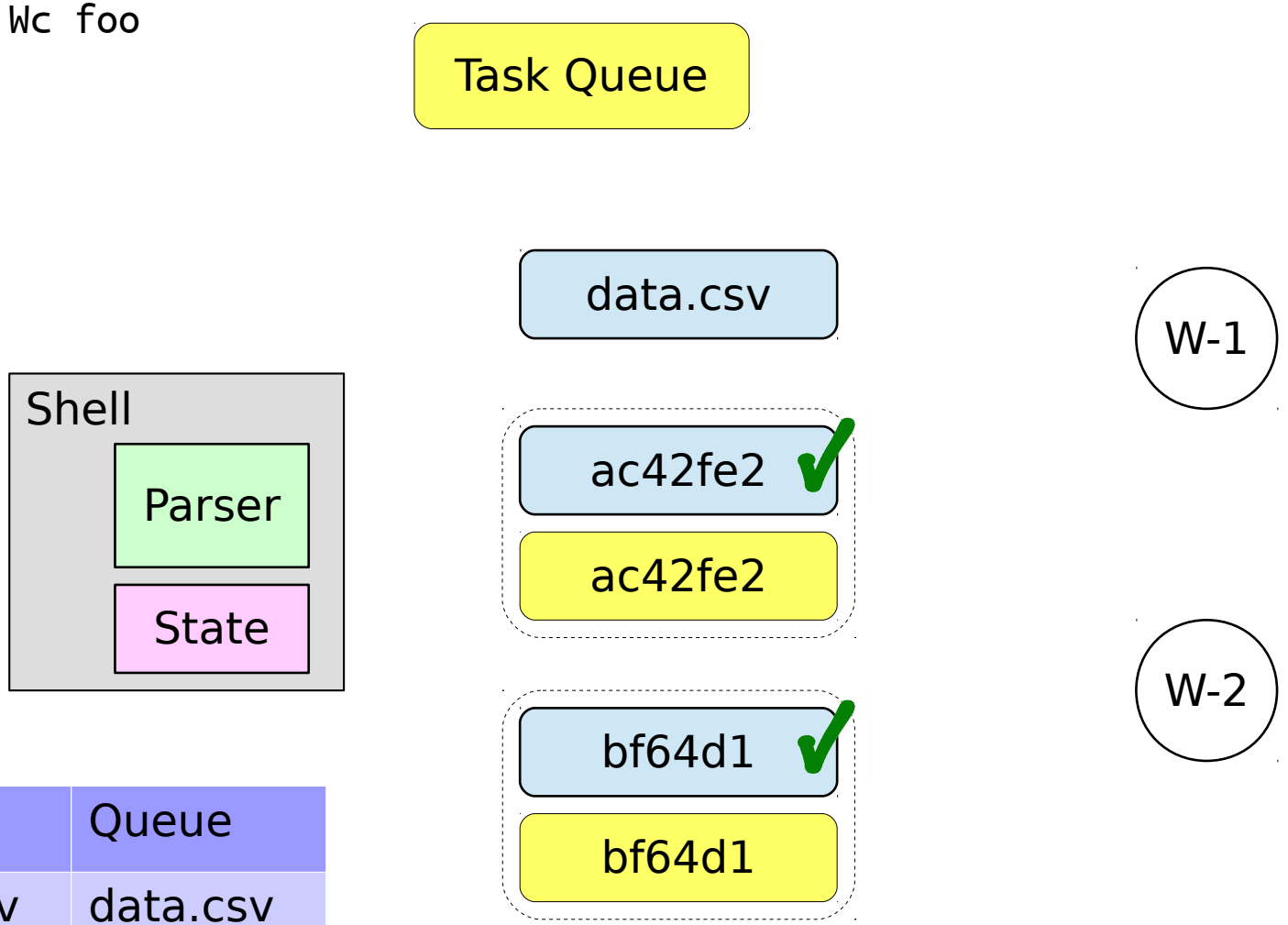
Name	Queue
data.csv	data.csv
foo	*ac42fe2*
bar	*bf64d1*

RabbitMQ - Signal

Kafka - Data

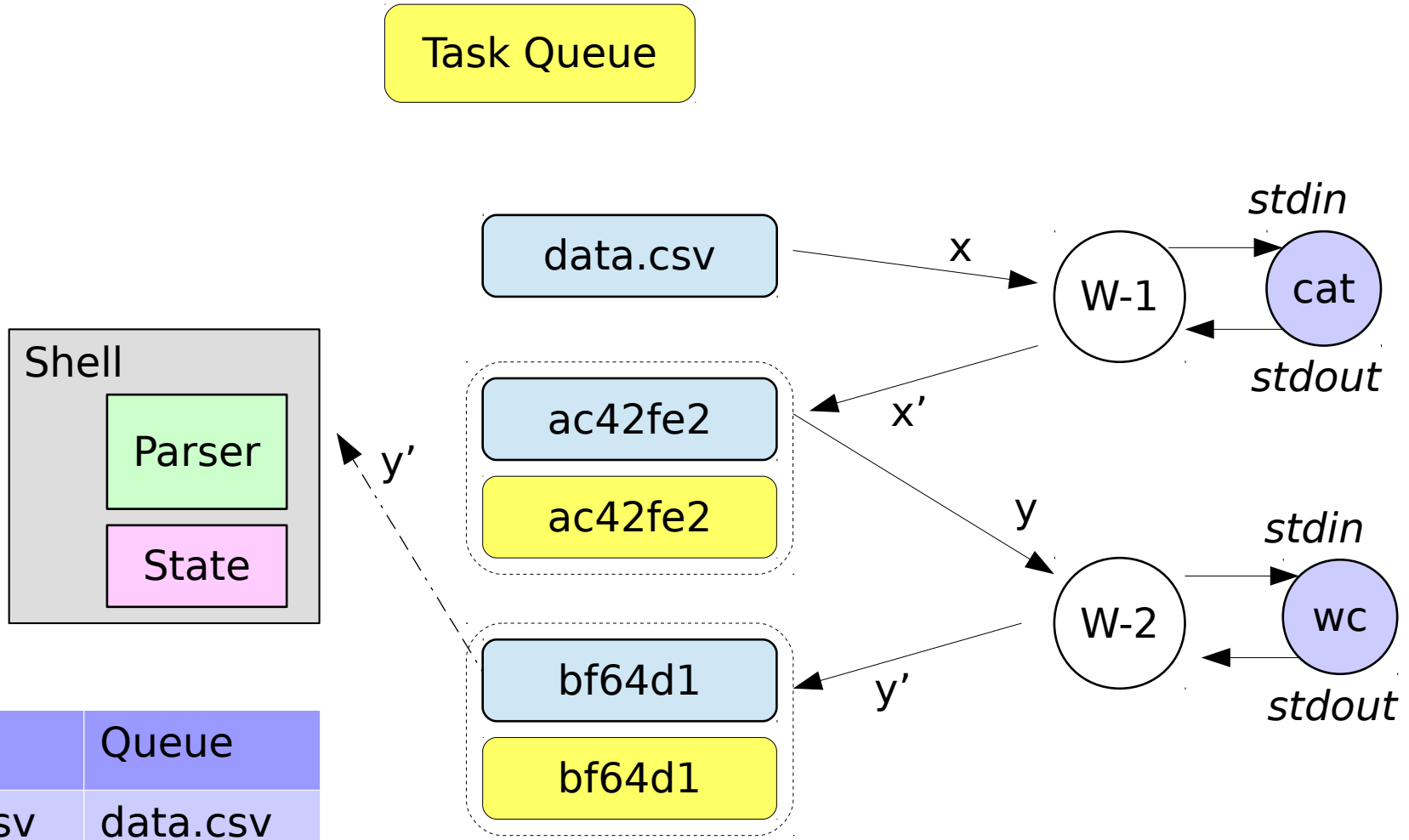
Drift Execution

```
.> bar = Wc foo
.> $bar
    2
.>
```



Name	Queue
data.csv	data.csv
foo	*ac42fe2*
bar	*bf64d1*

Drift Execution



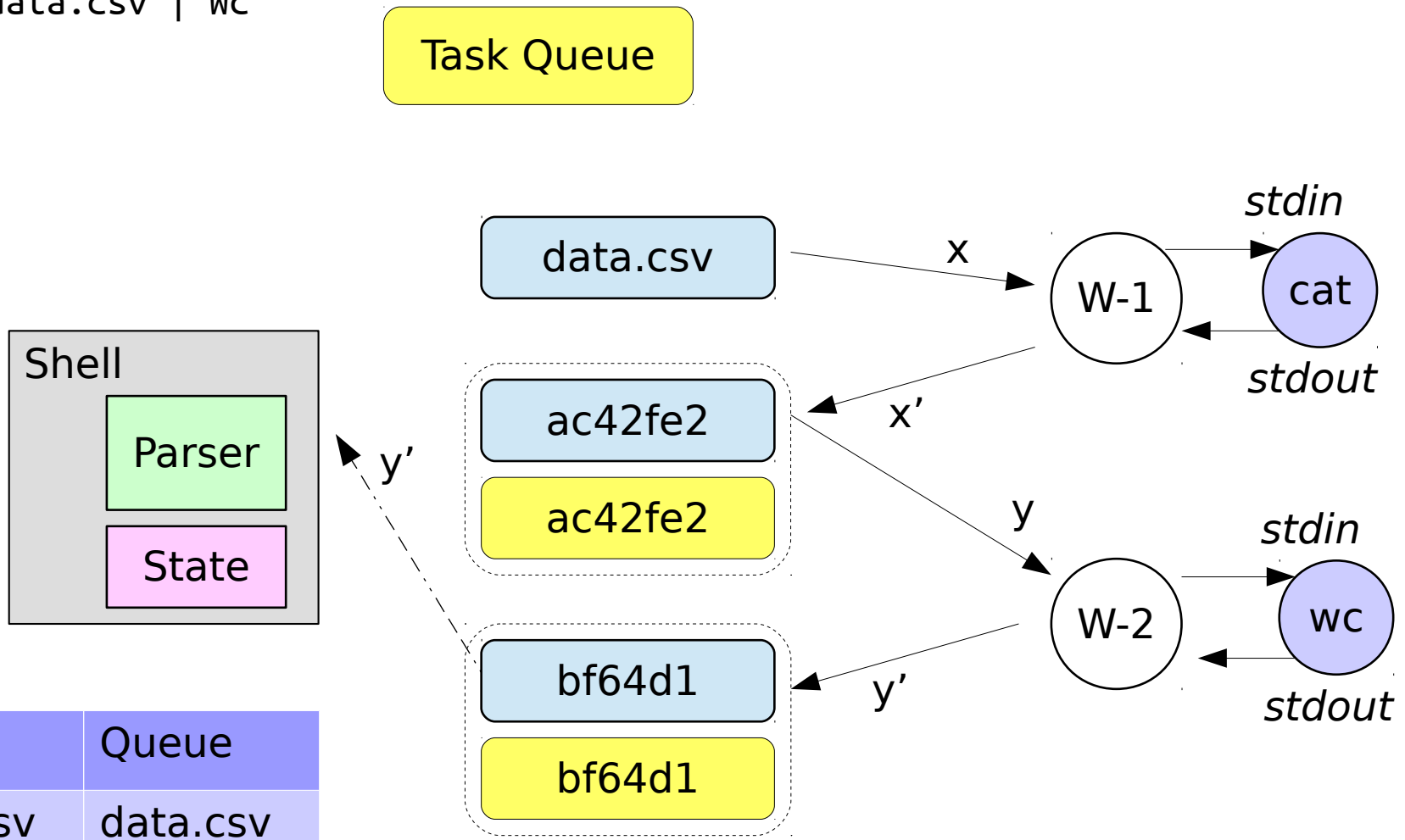
Name	Queue
data.csv	data.csv

RabbitMQ - Signal

Kafka - Data

Drift Execution

```
.> Cat data.csv | Wc
```



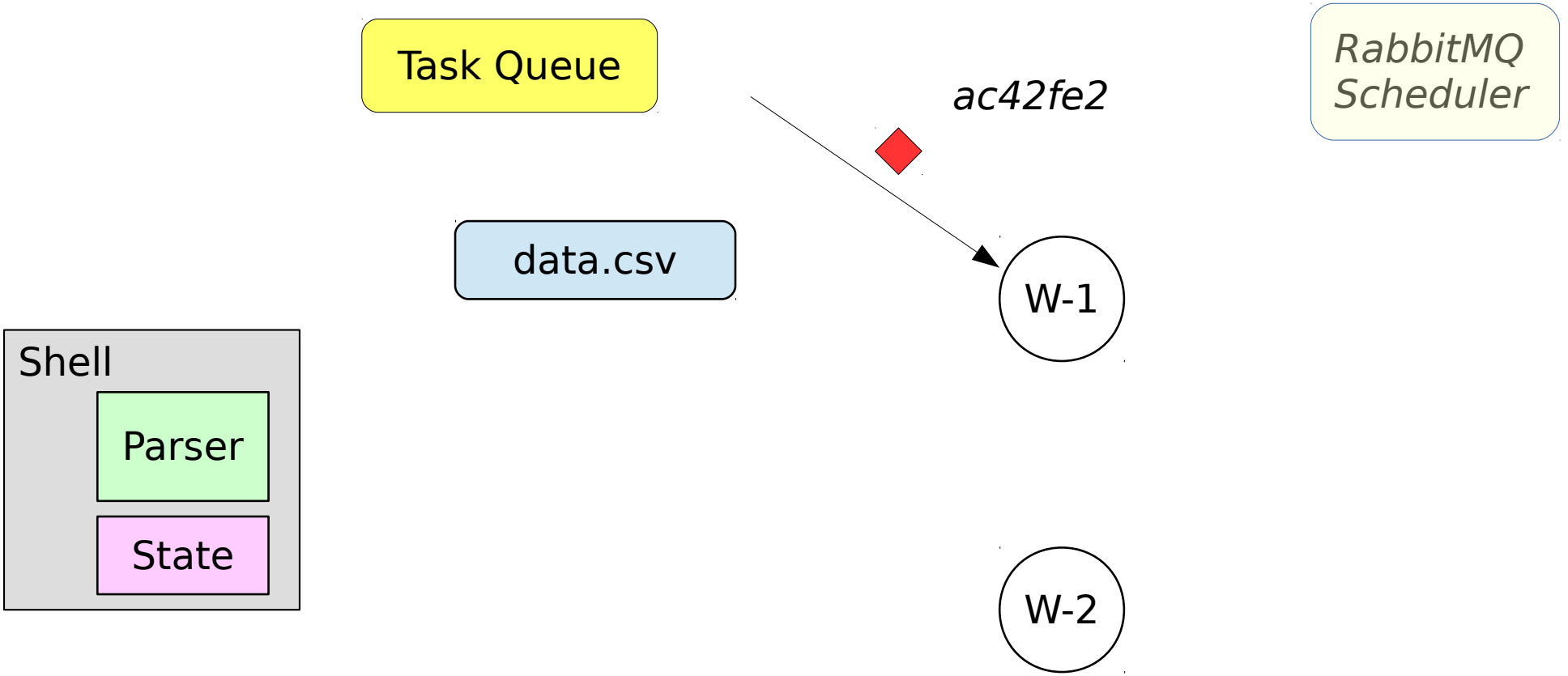
Name	Queue
data.csv	data.csv

RabbitMQ - Signal

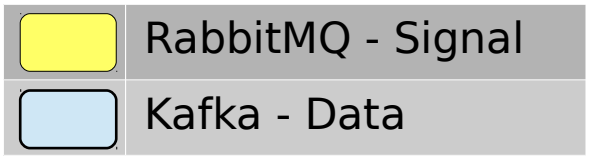
Kafka - Data

Error Cases

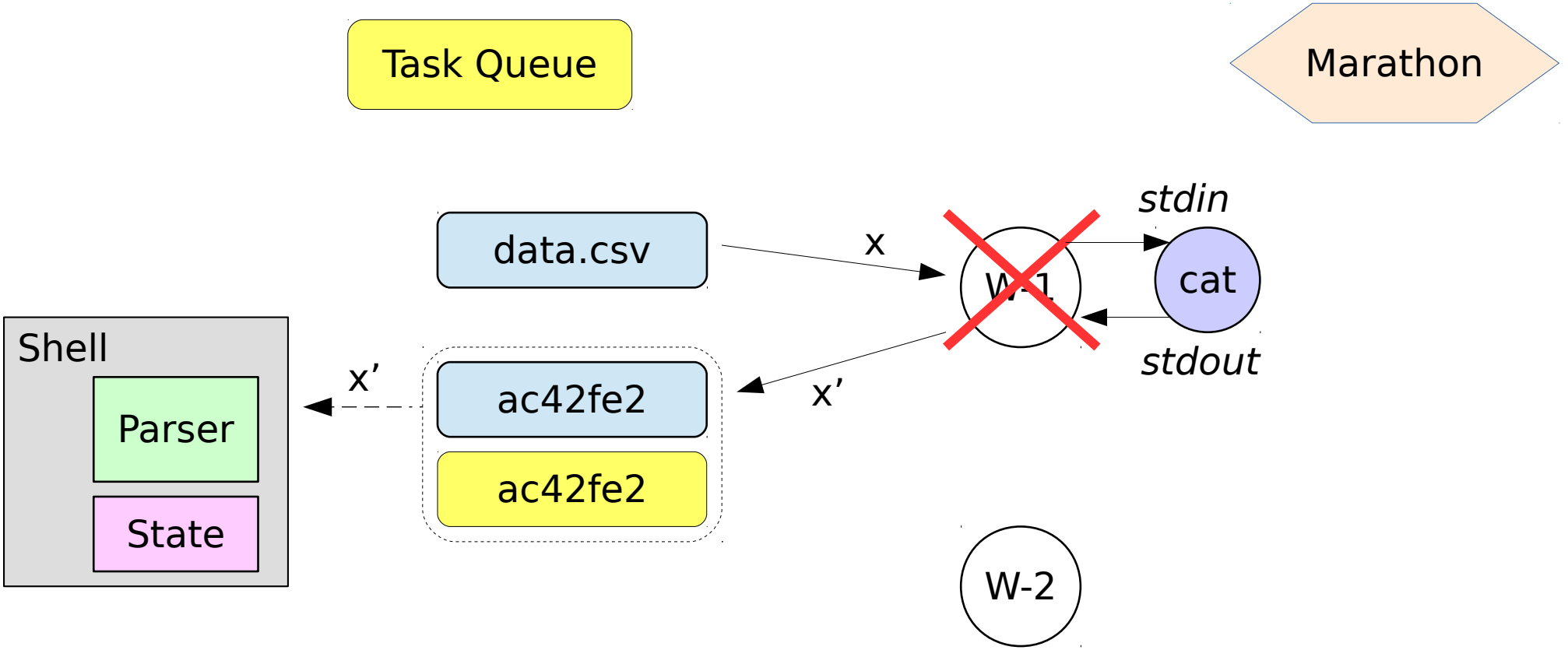
Drift Execution



Name	Queue
data.csv	data.csv
foo	*ac42fe2*



Drift Execution

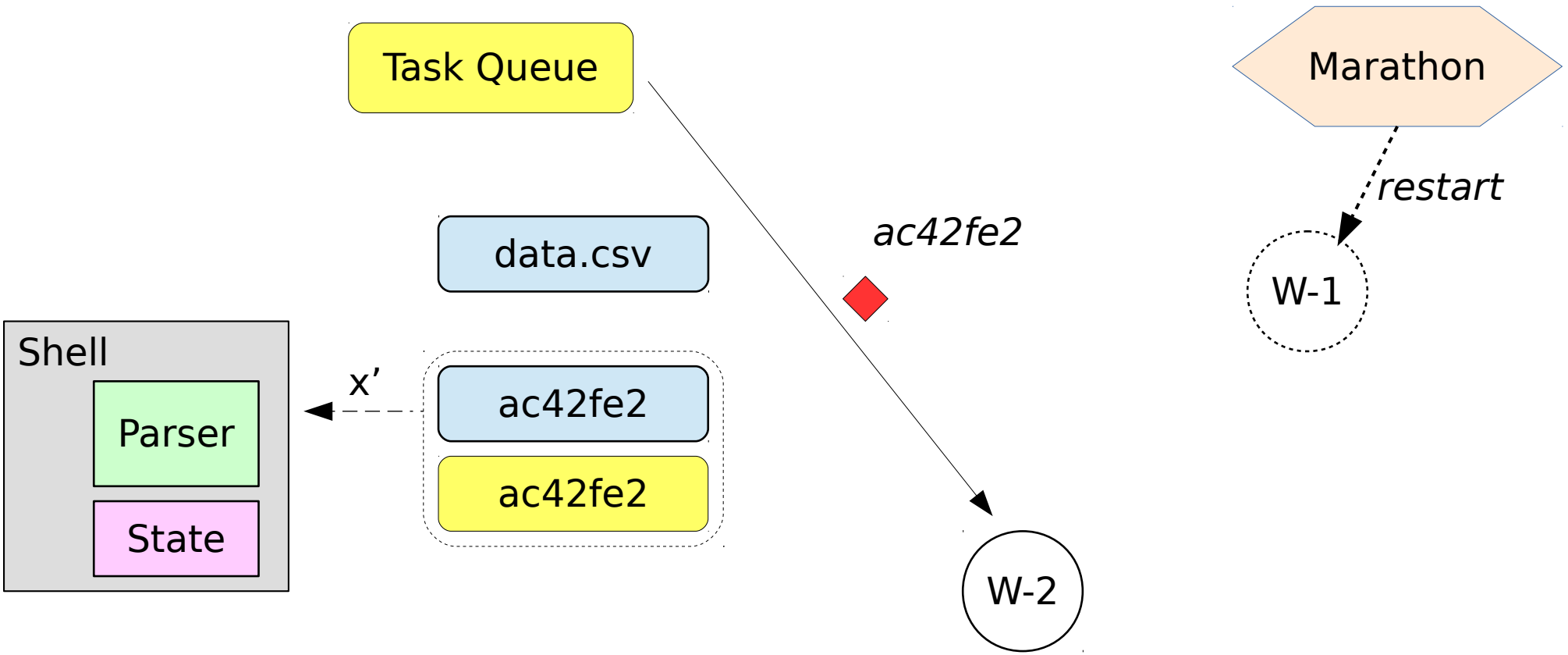


Name	Queue
data.csv	data.csv
foo	*ac42fe2*

RabbitMQ - Signal

Kafka - Data

Drift Execution

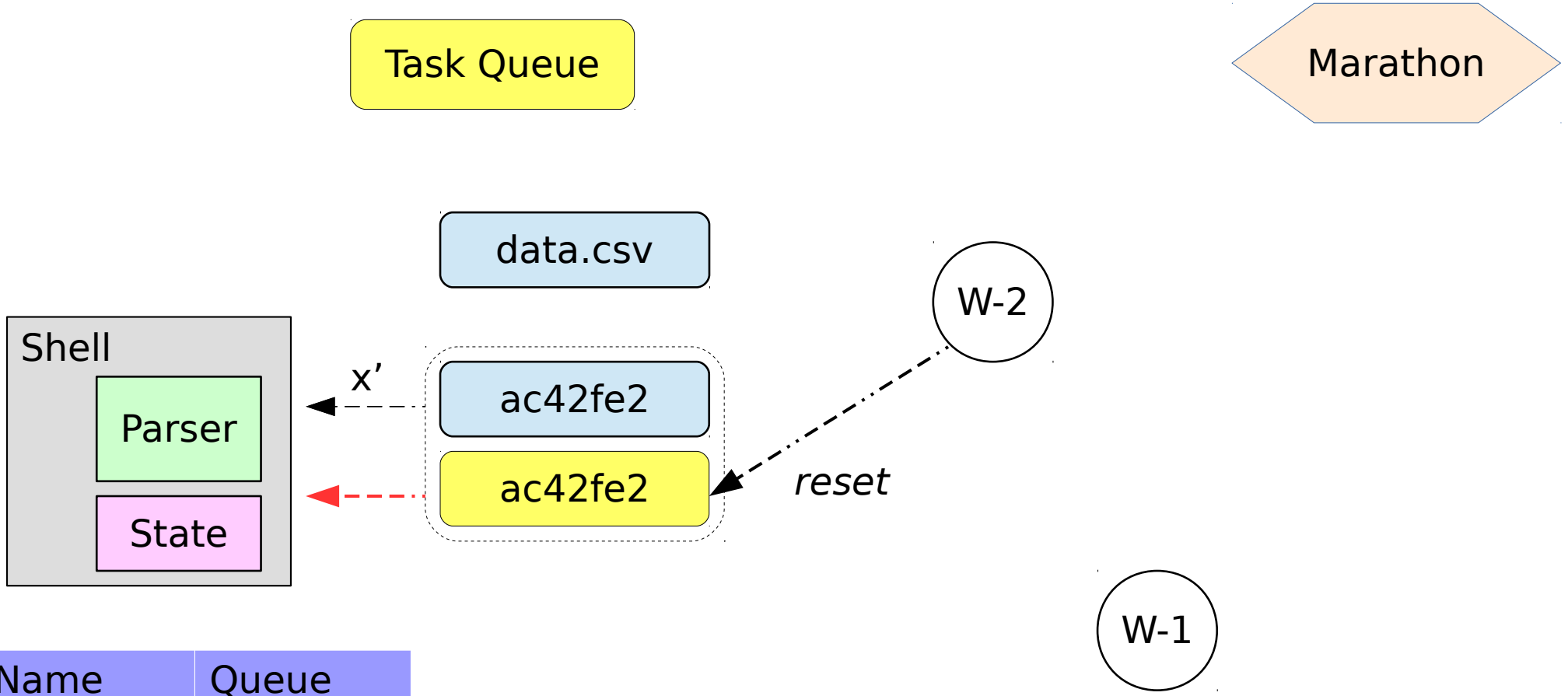


Name	Queue
data.csv	data.csv
foo	*ac42fe2*

RabbitMQ - Signal

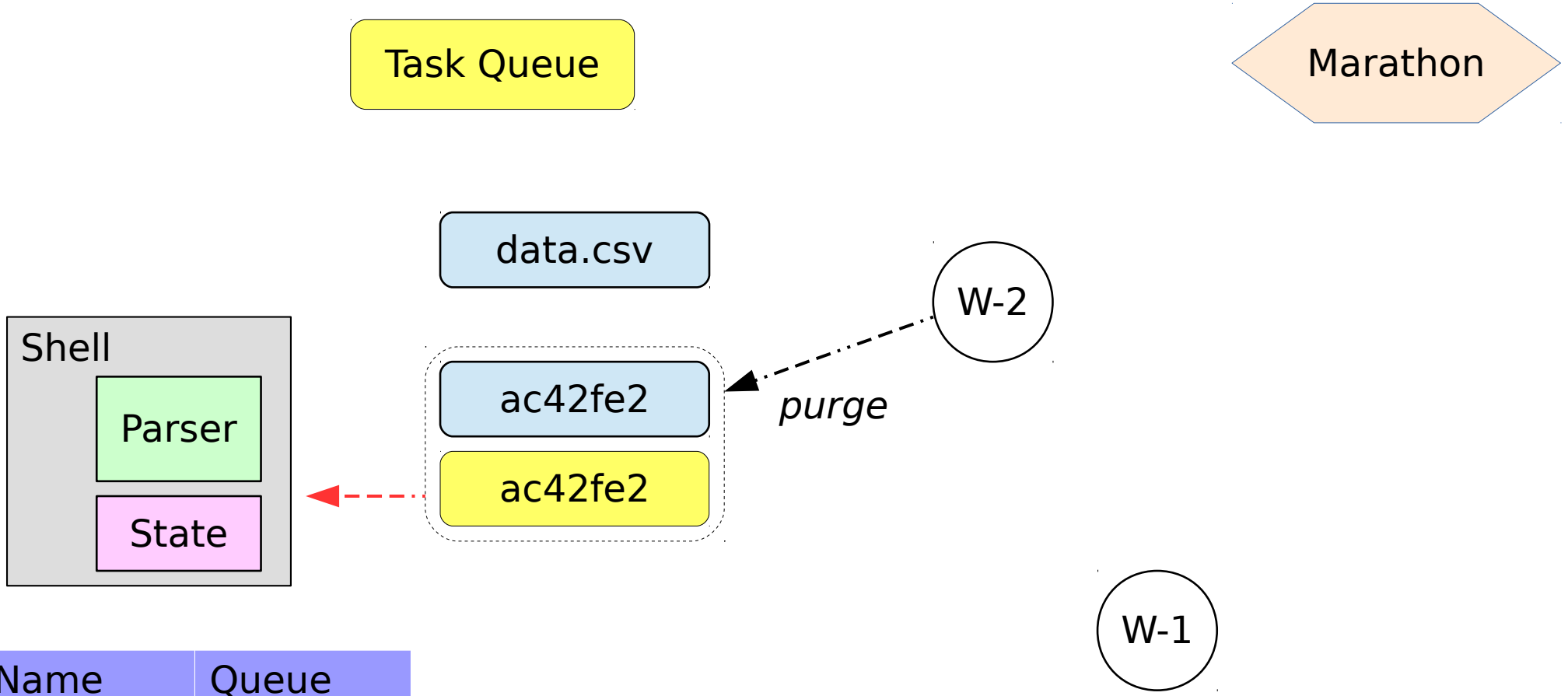
Kafka - Data

Drift Execution



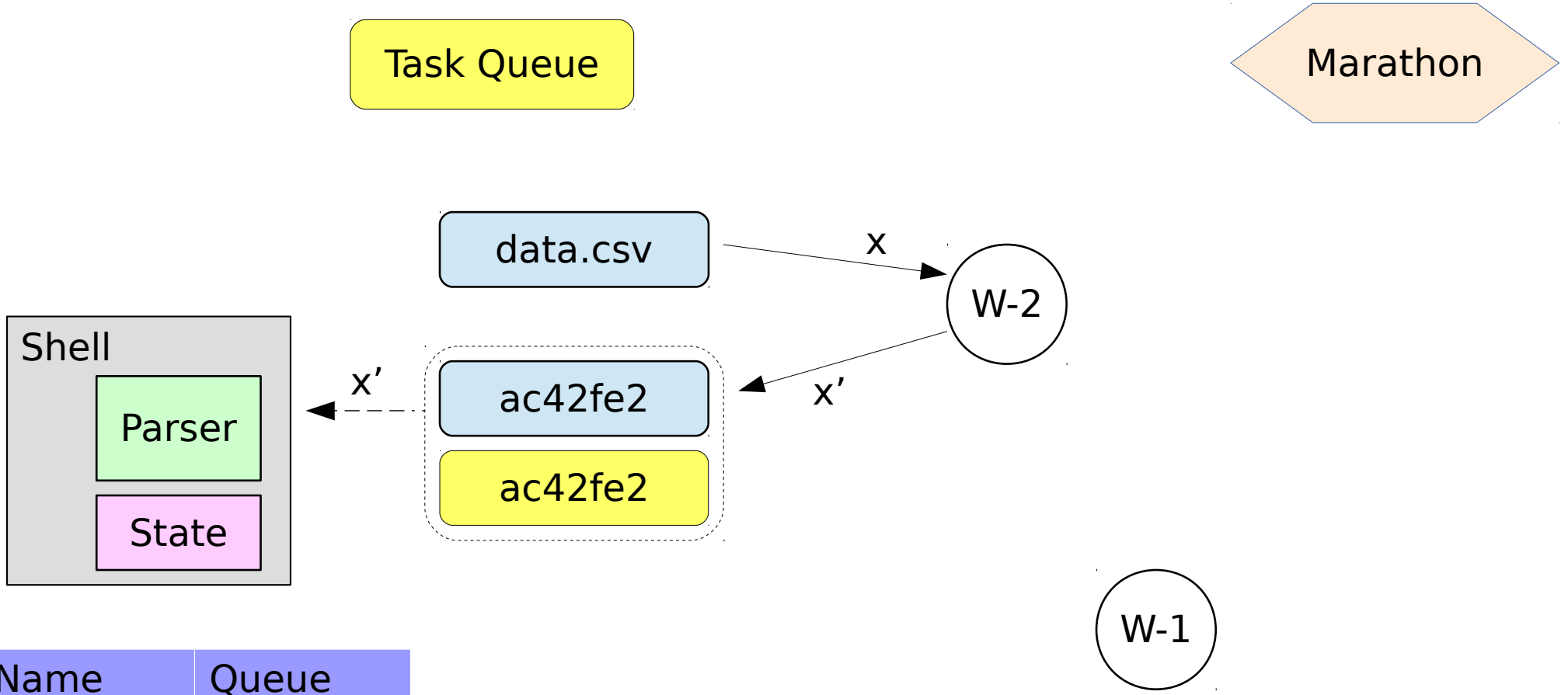
Name	Queue
data.csv	data.csv
foo	*ac42fe2*

Drift Execution



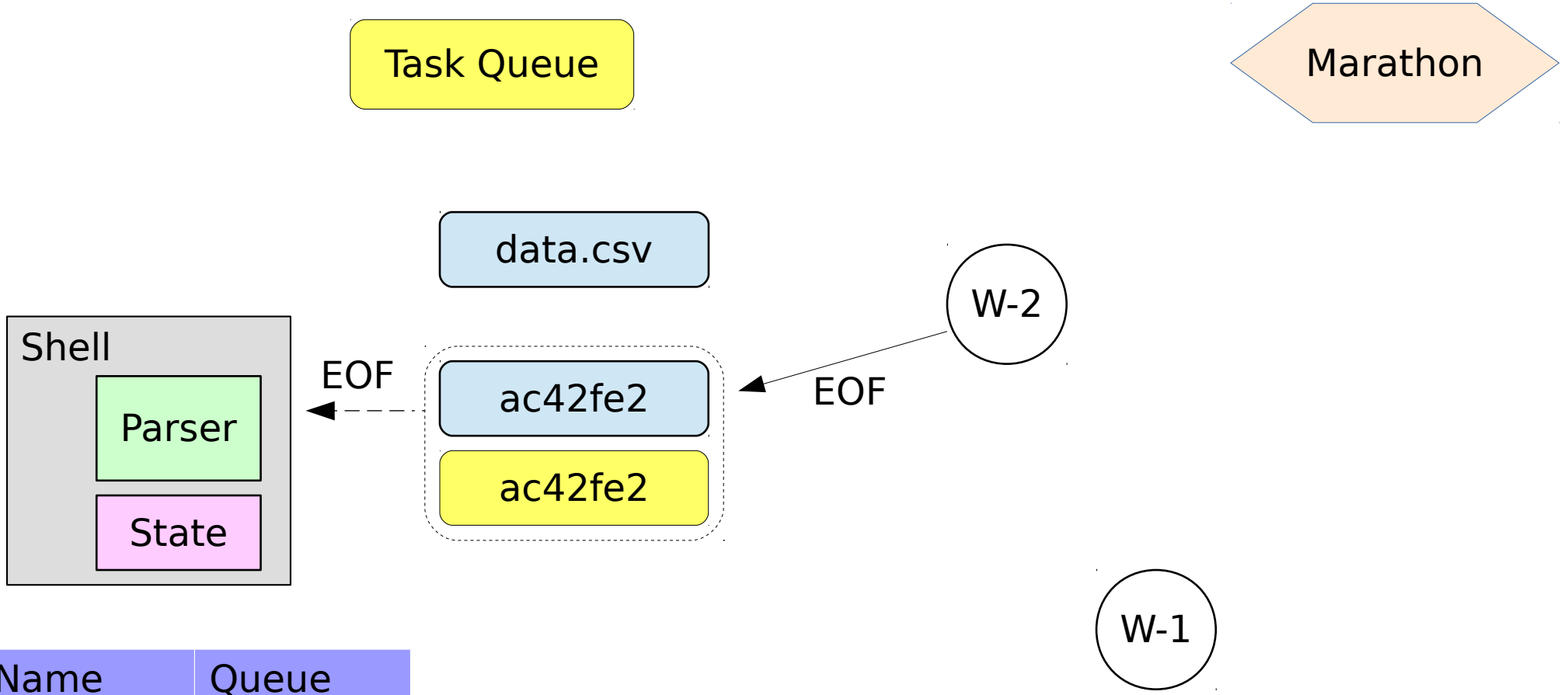
Name	Queue
data.csv	data.csv
foo	*ac42fe2*

Drift Execution



Name	Queue
data.csv	data.csv
foo	*ac42fe2*

Drift Execution

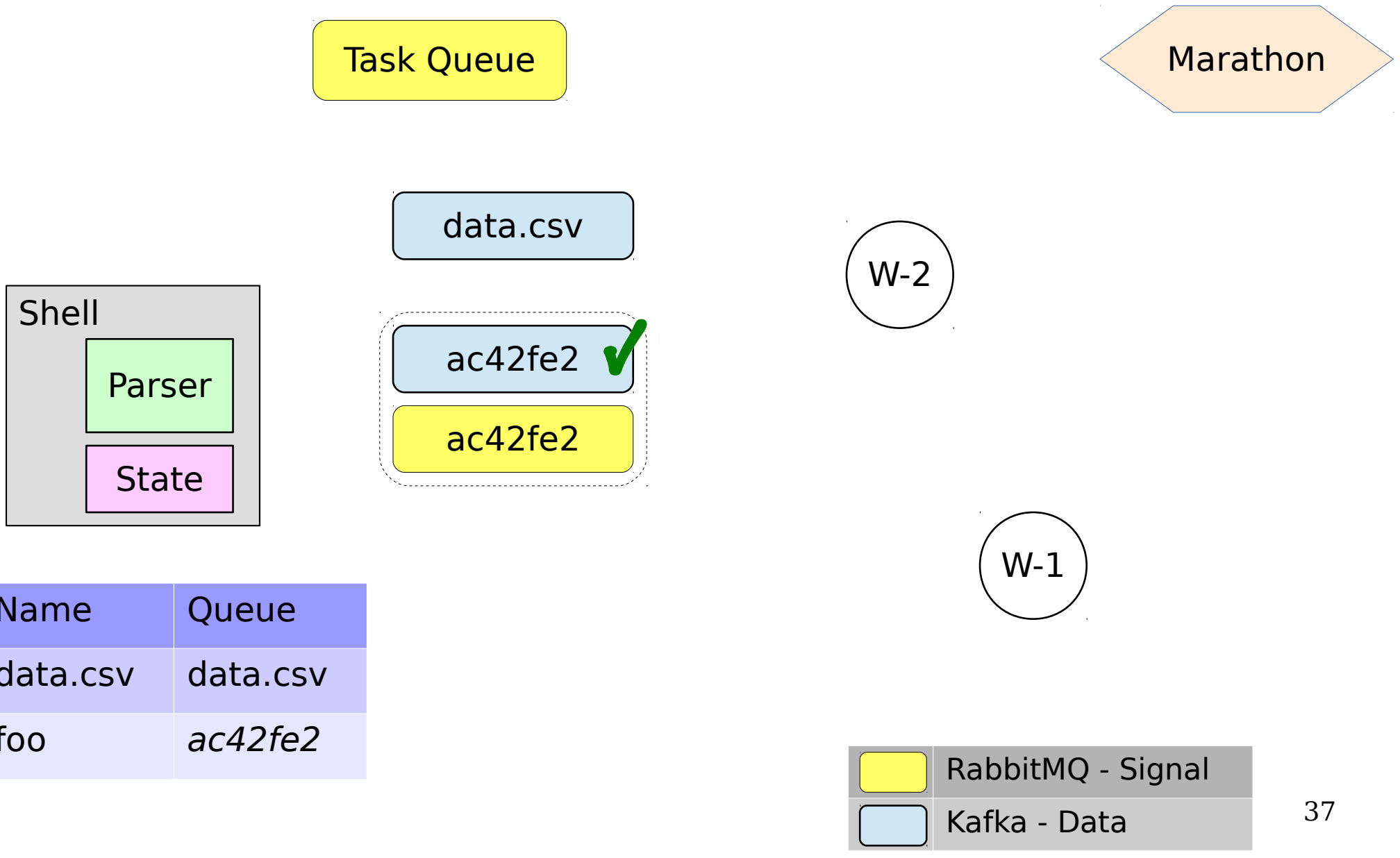


Name	Queue
data.csv	data.csv
foo	*ac42fe2*

RabbitMQ - Signal

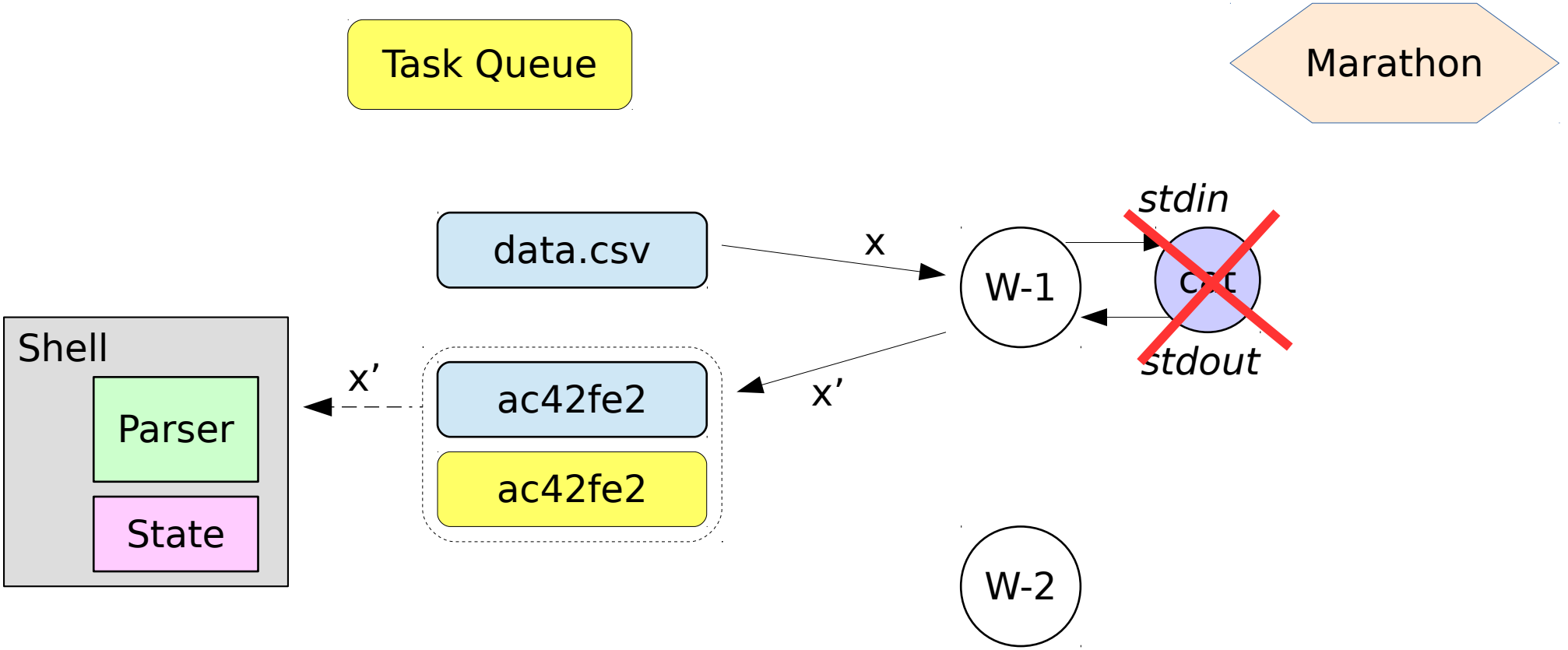
Kafka - Data

Drift Execution



Name	Queue
data.csv	data.csv
foo	*ac42fe2*

Drift Execution

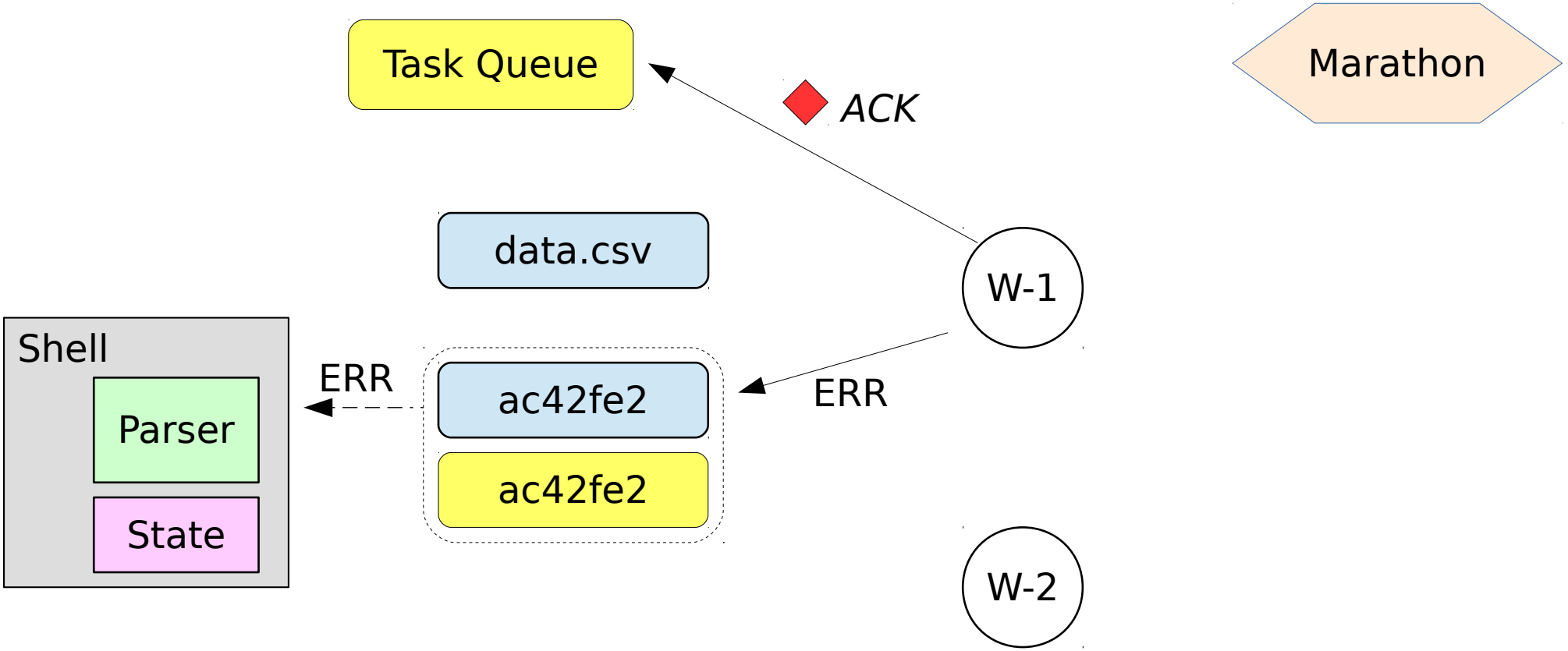


Name	Queue
data.csv	data.csv
foo	*ac42fe2*

RabbitMQ - Signal

Kafka - Data

Drift Execution

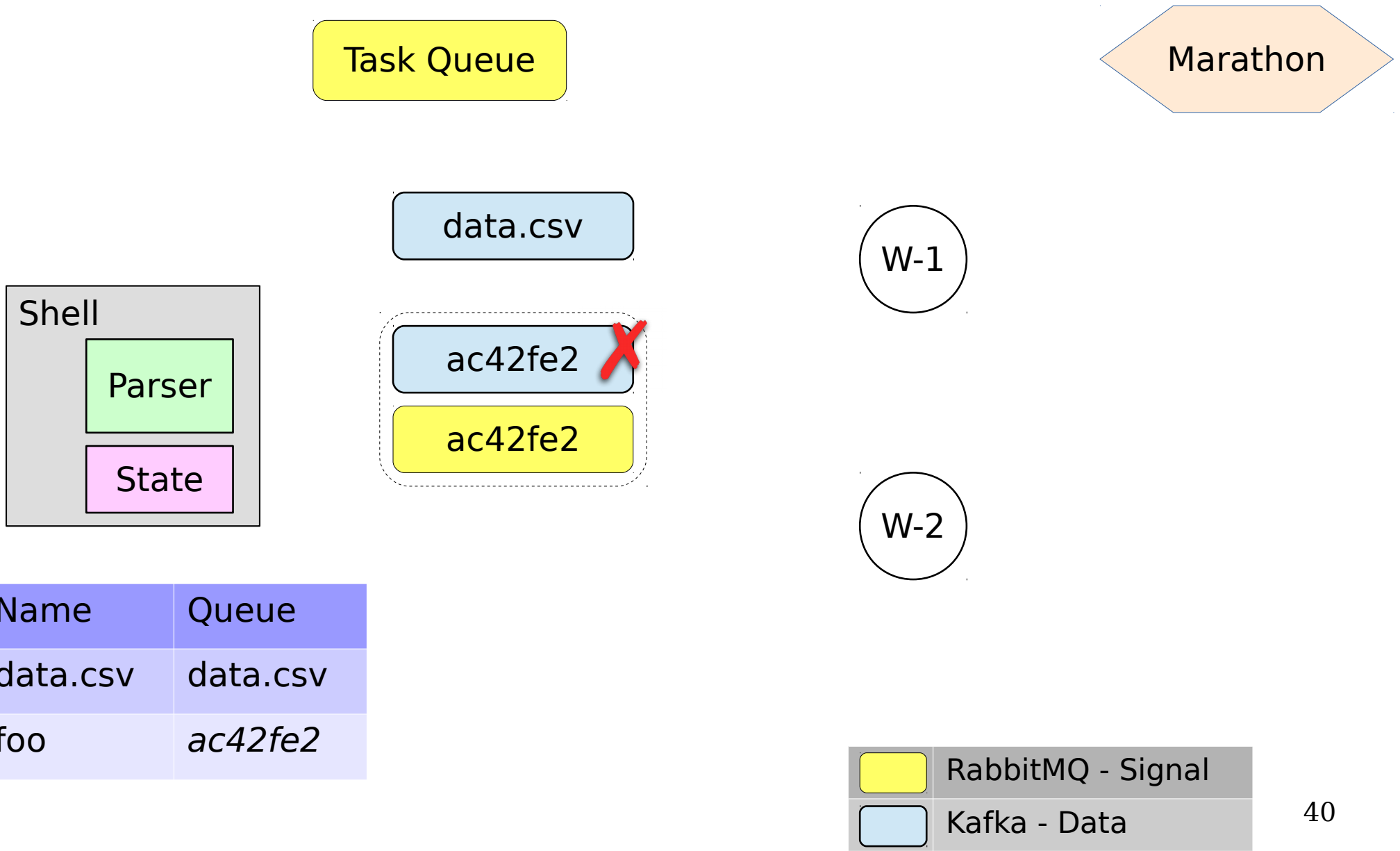


Name	Queue
data.csv	data.csv
foo	*ac42fe2*

RabbitMQ - Signal

Kafka - Data

Drift Execution



Name	Queue
data.csv	data.csv
foo	*ac42fe2*

Error Model

- Error Model: crash recovery
- System/Worker Failover → transparent
- Task Failure → permanent

  → because deterministic tasks

- queued tasks run until ERR token
- otherwise: task not accepted by interpreter

Invariants

System Invariants:

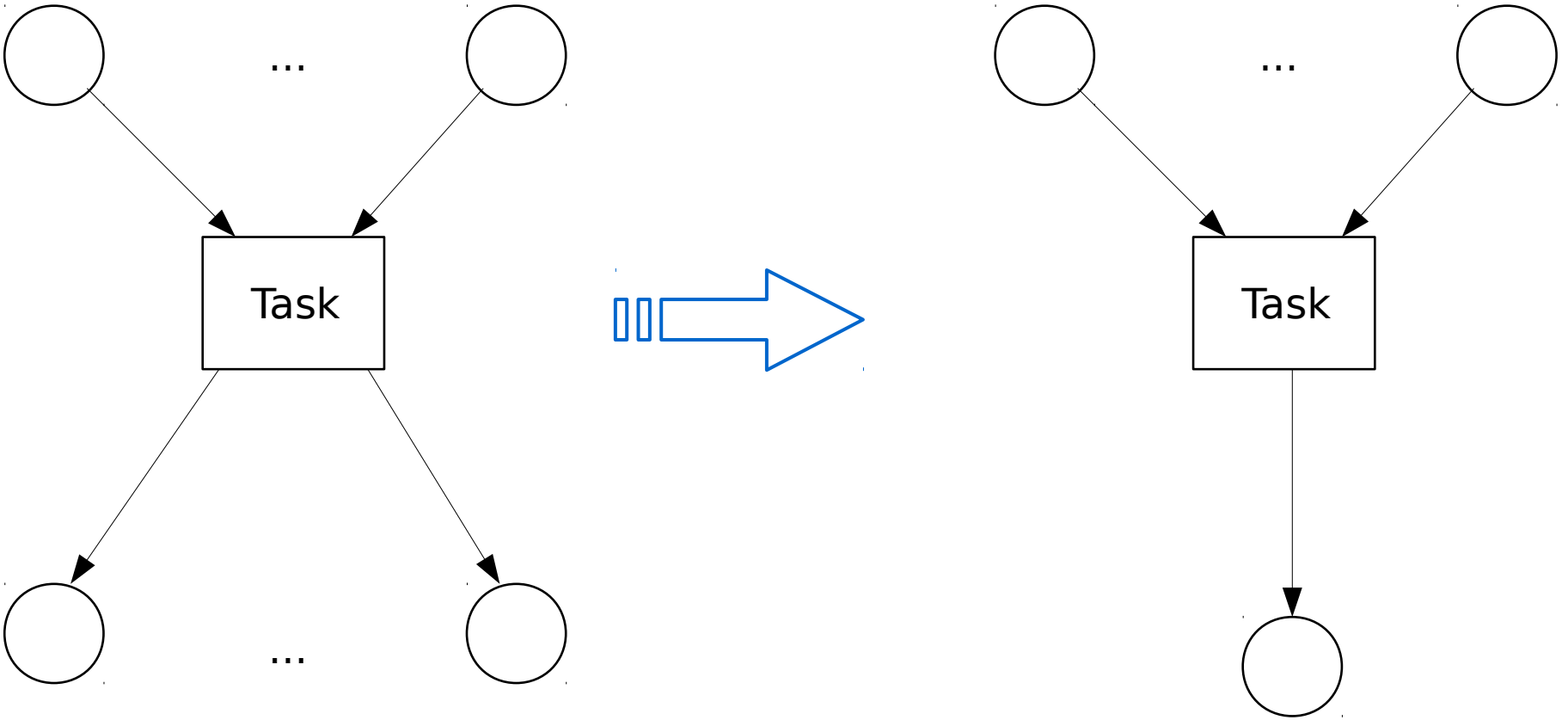
- Task scheduled exactly once or never
- Task executed exactly once or never
- Task always completed with EOF or ERR
- Task are deterministic

- possible to start unbounded tasks

→ language: unbounded

→ implementation: bounded

Invariants



Drift GUI

- Petri Net syntax natural fit for data + services
- different semantics:
  - no ‘occurence rule’
  - no markings consumption
- BUT same properties like:
  - transition locality
  - async by default

Future Work

Major Problems:

- Semantics ...
- Types / Safety ...
- Platform ...

Minor Problems:

- Visualization
- Generalization / API