Drip: Instant Replay

persistent tuple space and stream

seki@ruby-lang.org

author of dRuby, Rinda, ERB and Drip

id:secondlife and me

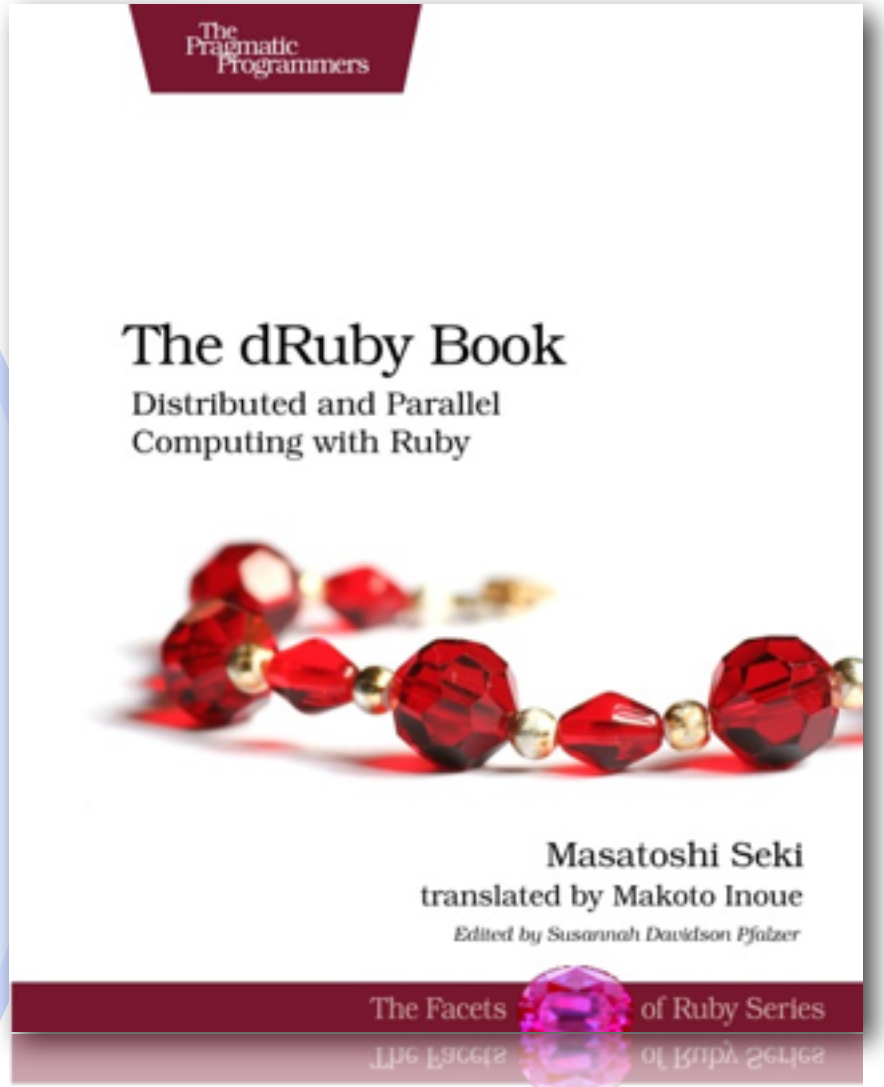
- RubyKaigi 2006
- はてなスクリーンショット using dRuby

日本先行発売

- まさかほんとに

The dRuby Book

- 索引作成中
- 素敵forward by matz
- 豪華quotes
- amazonで予約now



ちゃんとした会社員

- わりと大きな会社
- プロの無職
  - スペアプログラマ

たまにコミッタ

- dRuby
- Rinda
- ERB

RubyKaigi x 6

- 2006 dRuby, Again.
- 2007 Answering dRuby and Rinda
- 2008 erbを偲んで
- 2009 偉大なBigTableとぼくのおもちゃ
- 2010 RWikiと怠惰な私の10年間
- 2011 persistant tuple space and stream

この辺のネタと関係がある

- 2007 Answering dRuby and Rinda
  - TupleSpaceの永続化発表
- 2009 偉大なBigTableとぼくのおもちゃ
  - 世界を並べてseek & read
- 2010 RWikiと怠惰な私の10年間
  - in-memory databaseの10年分の実例

今日の話

- 永続版TupleSpace、PTuplespaceの反省
- ストリーム指向のストレージDripの紹介
- 言い訳

Drip

A stream oriented storage

Dripとはなにか

- かっこいいログ
  - 追記のみ。削除・修正はできない
  - 新しいログが書かれるまで待合せ
  - 局所的で安価なブラウズAPI

PTupleSpaceへの別の解

- 失敗してもなんとかやり直せる協調機構
- 履歴付きHashにも見えるストレージ

応用例

- RWiki全文検索
- 電力消費量レポートシステム
- タイムラインのアーカイブ
  - botフレームワーク
- irb中のちょっとしたオブジェクトの保存

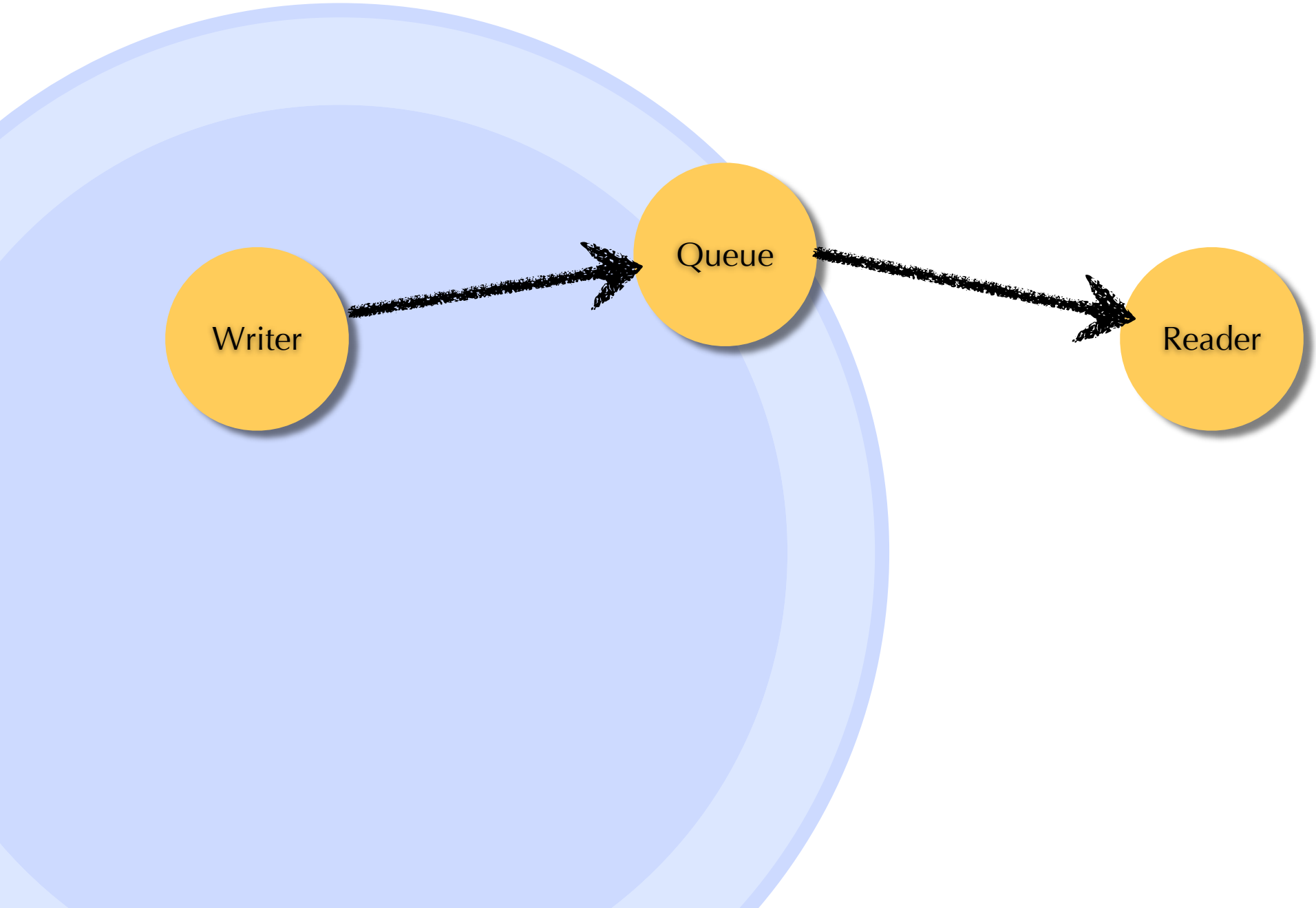
身近な例で説明

- Queueとの比較 / 協調
- Hashとの比較 / ストレージ

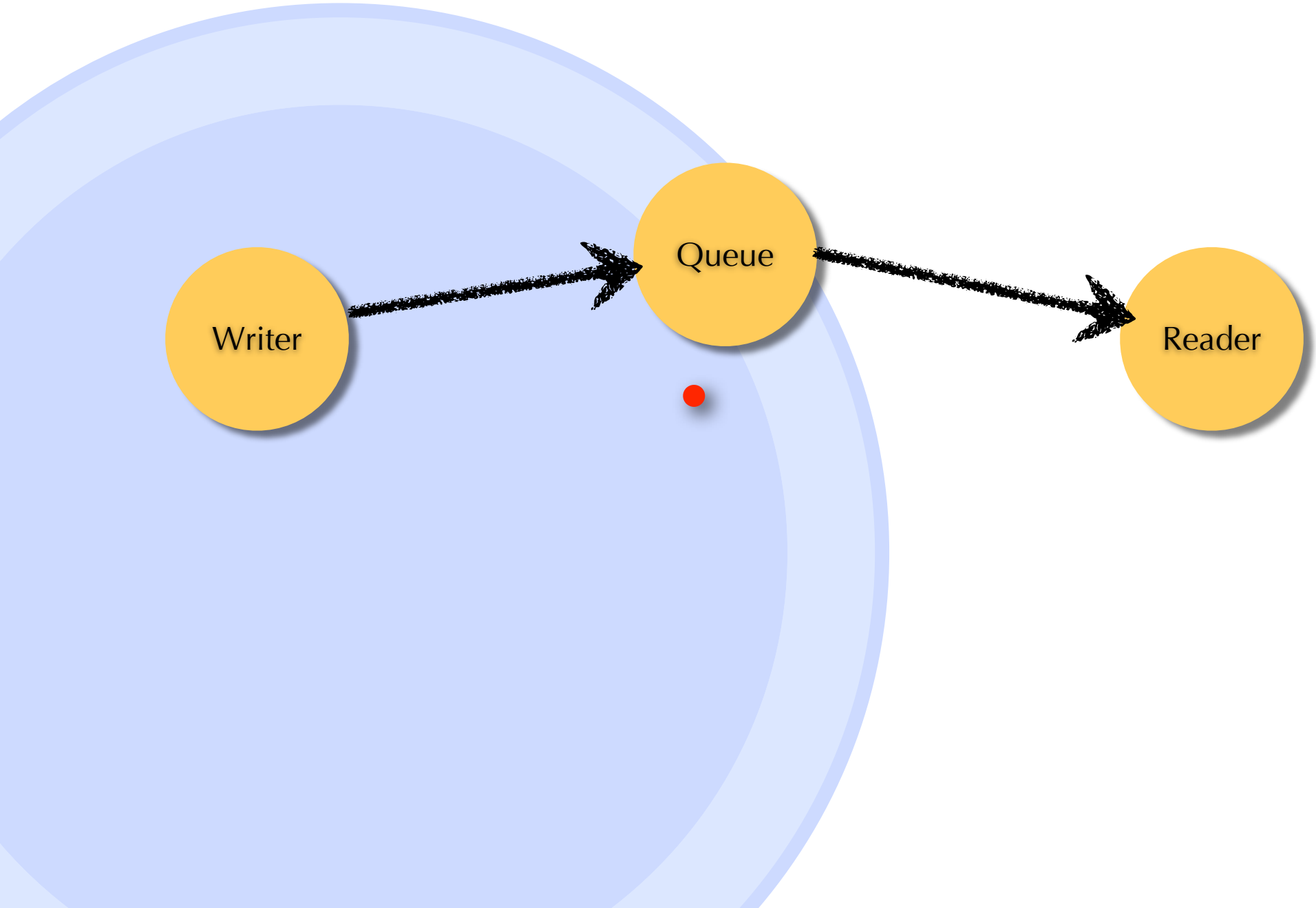
Queueとの比較

- Queueの特徴
  - オブジェクトが移動する
  - 待合せしてくれる

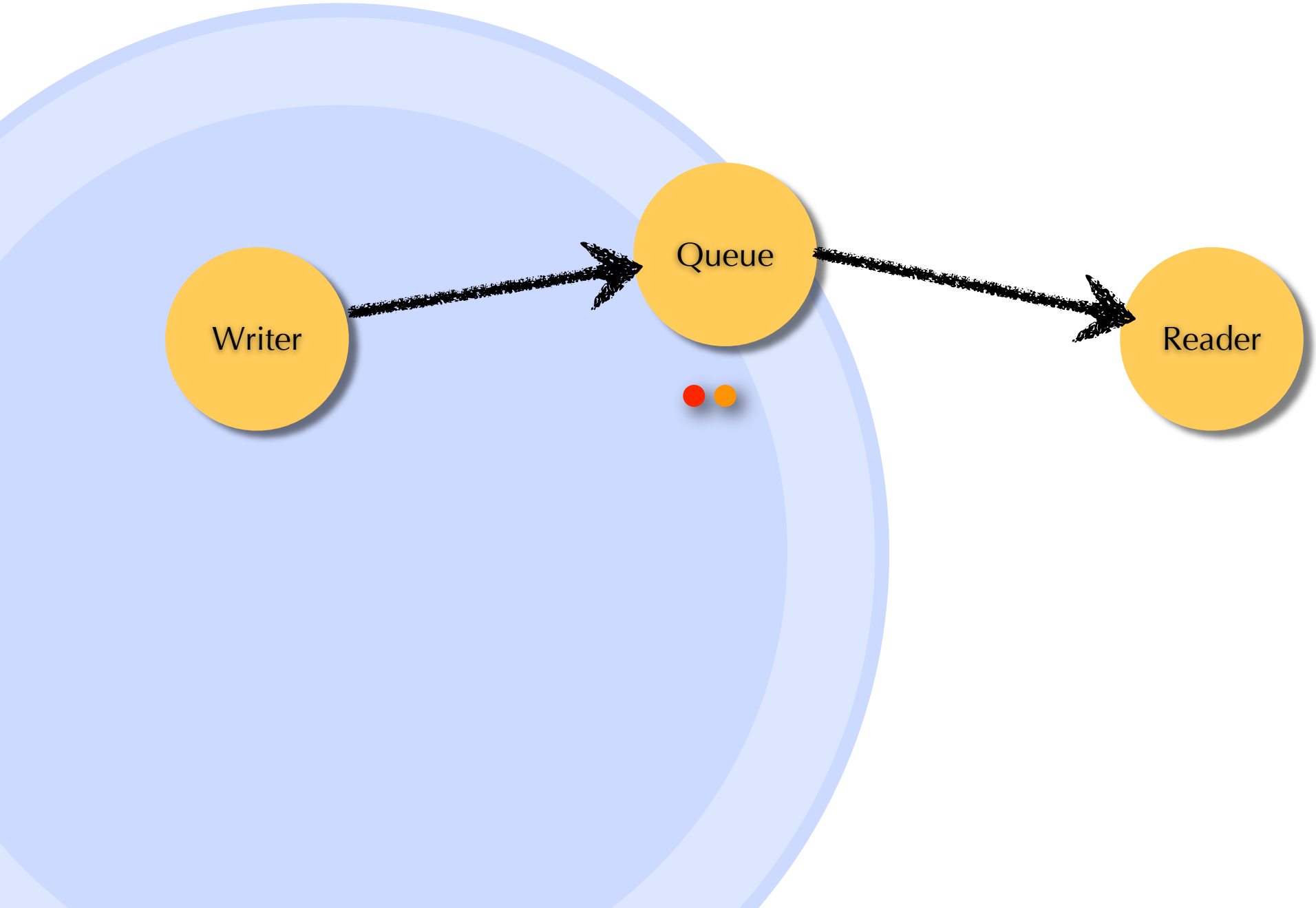
Queueだとこんな感じ



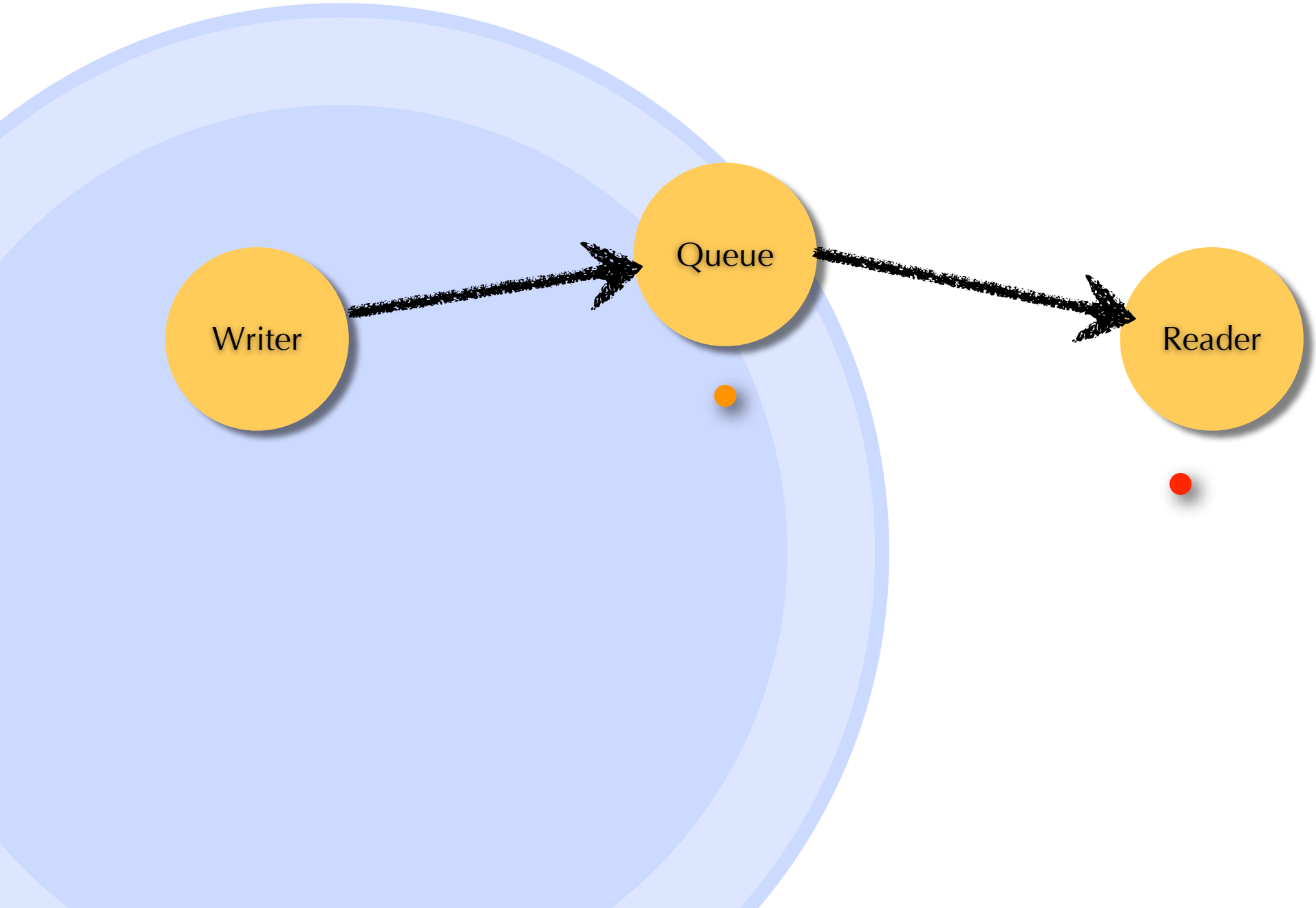
Queueだとこんな感じ



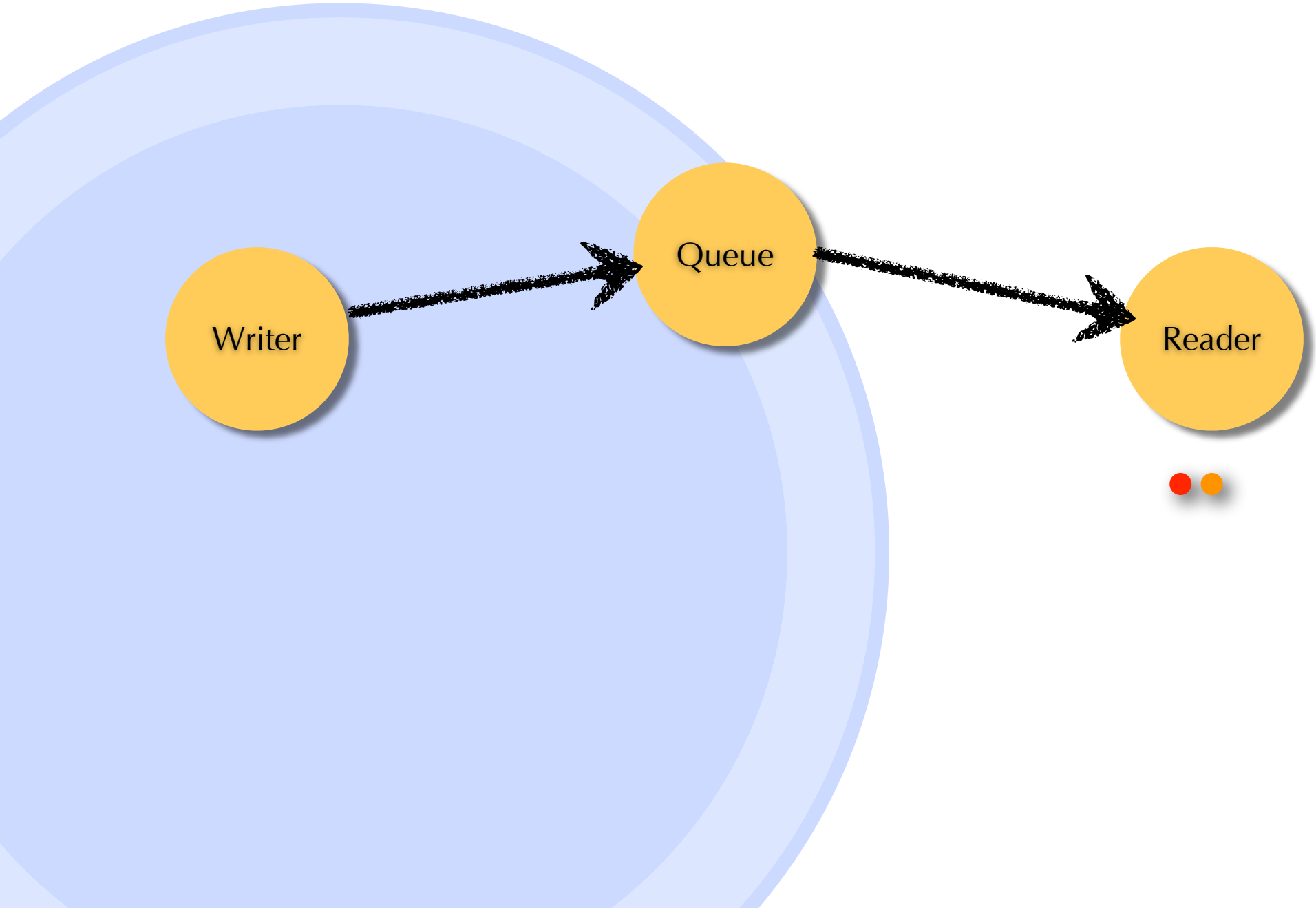
Queueだとこんな感じ



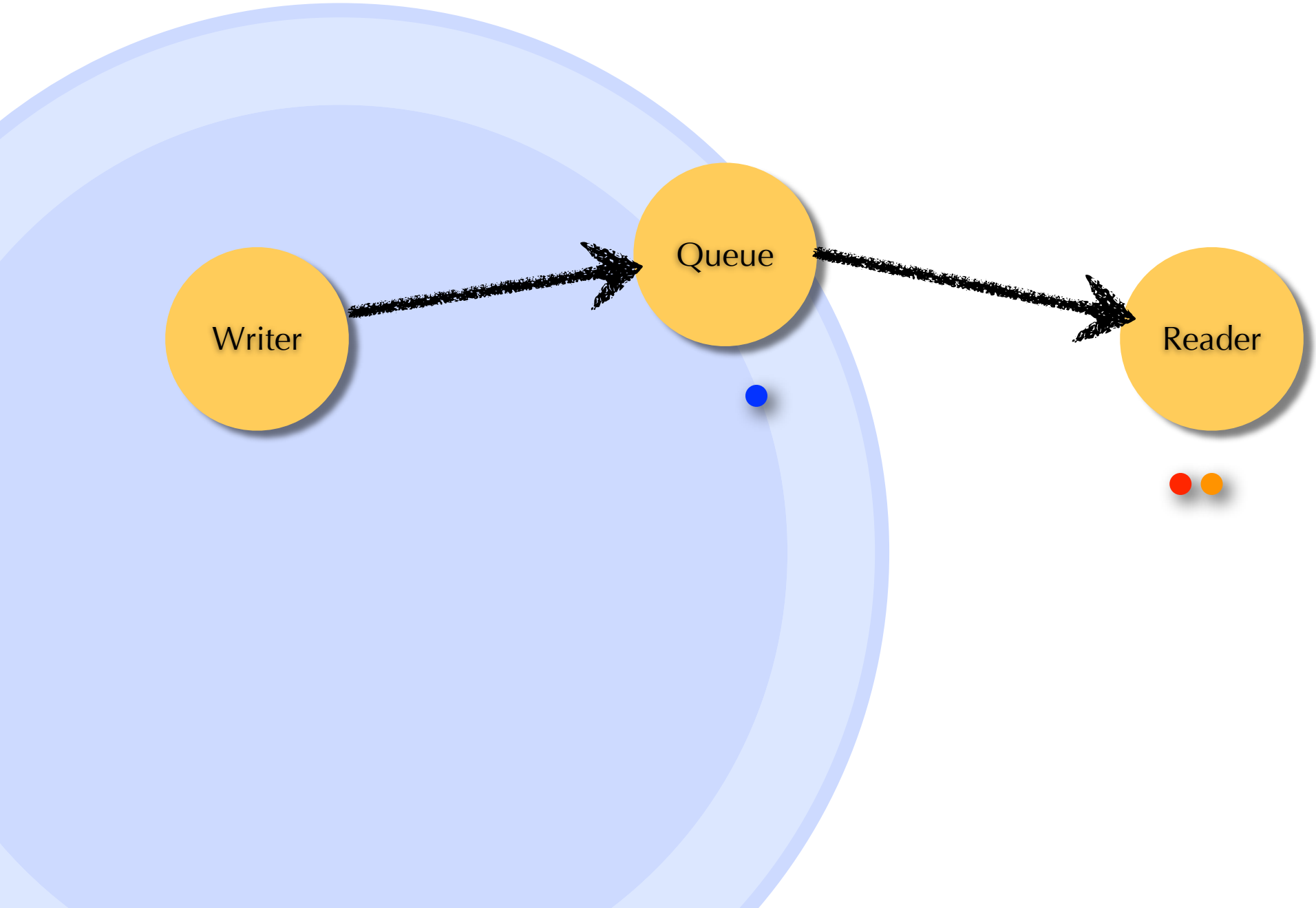
Queueだとこんな感じ



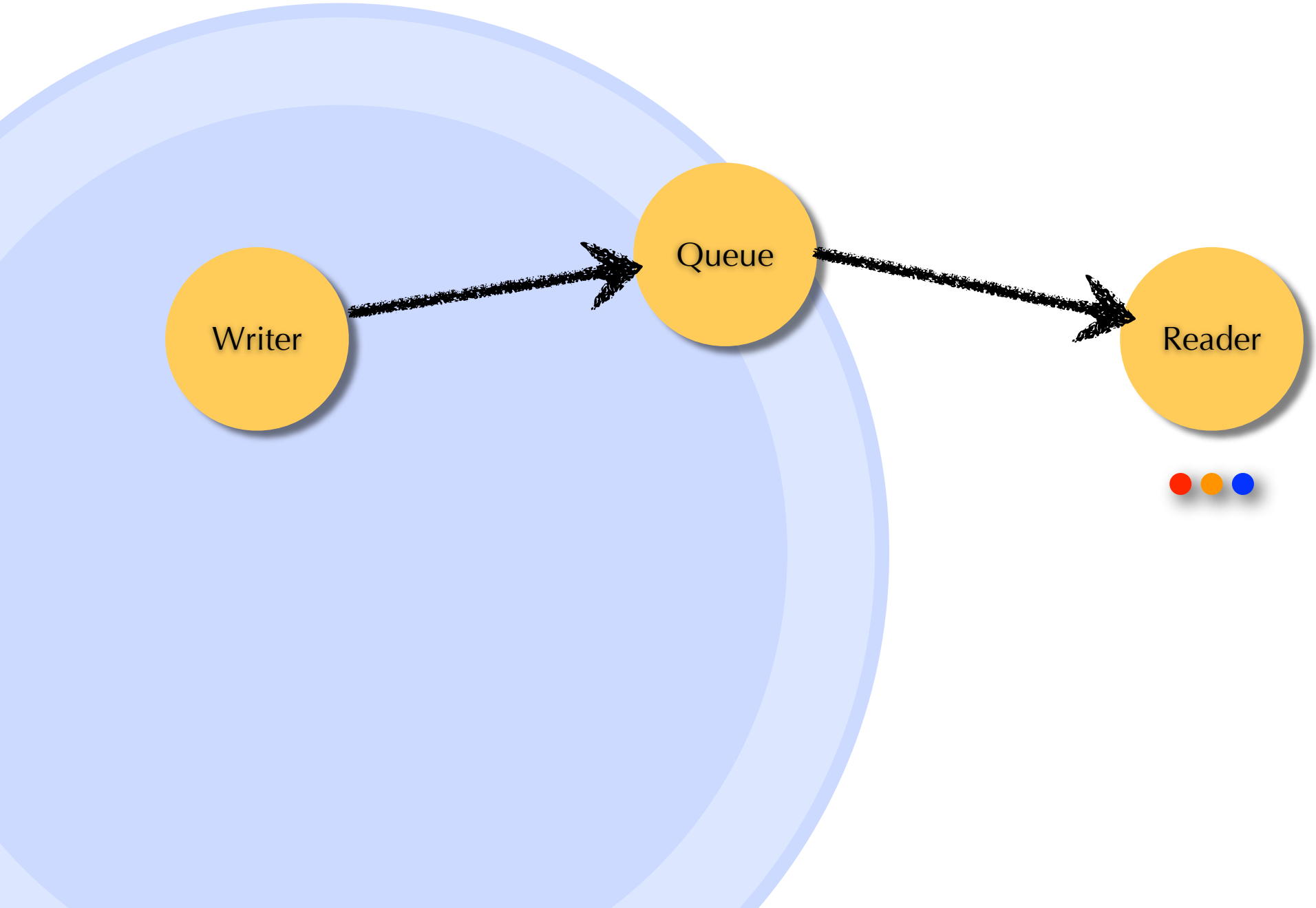
Queueだとこんな感じ



Queueだとこんな感じ



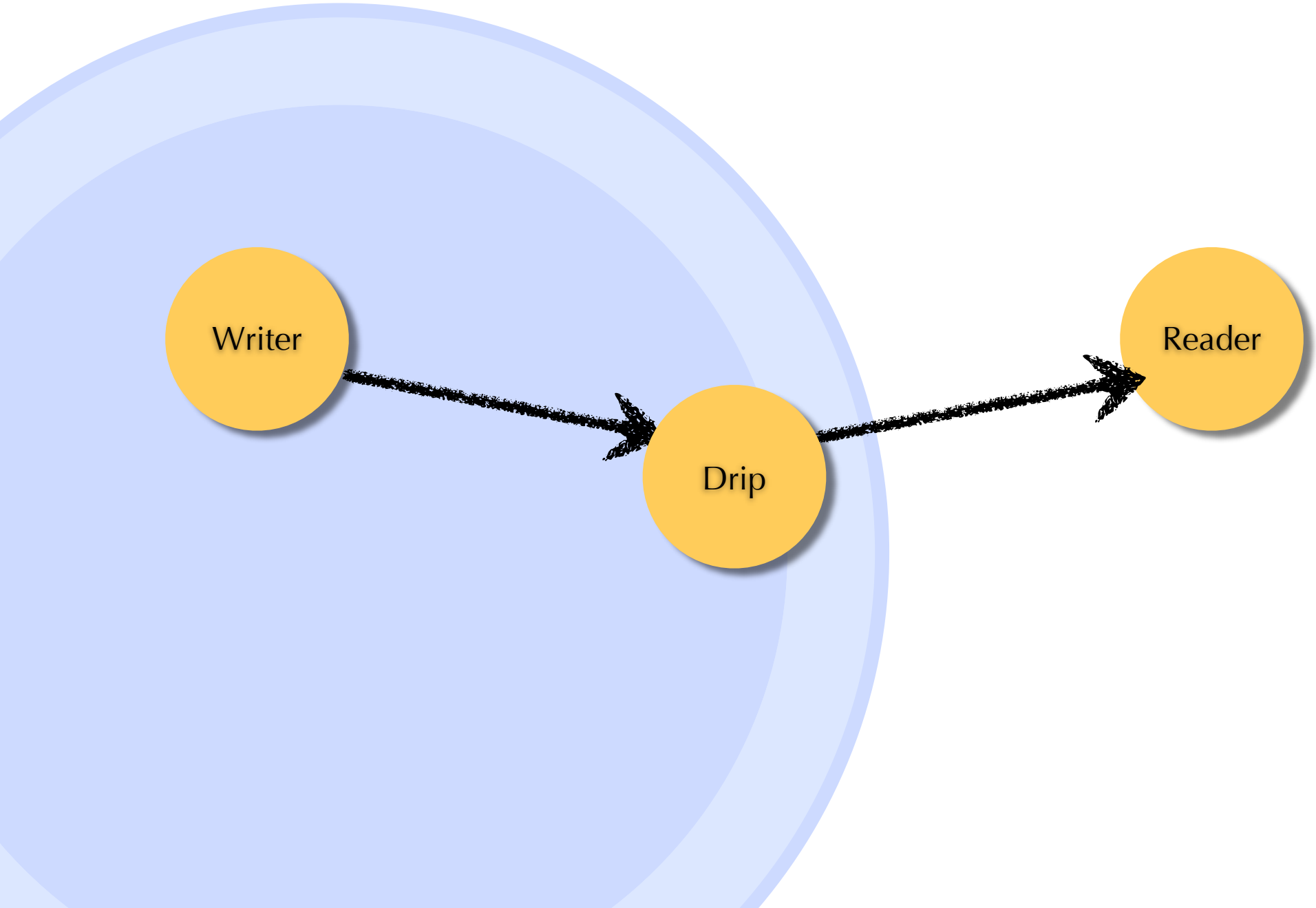
Queueだとこんな感じ



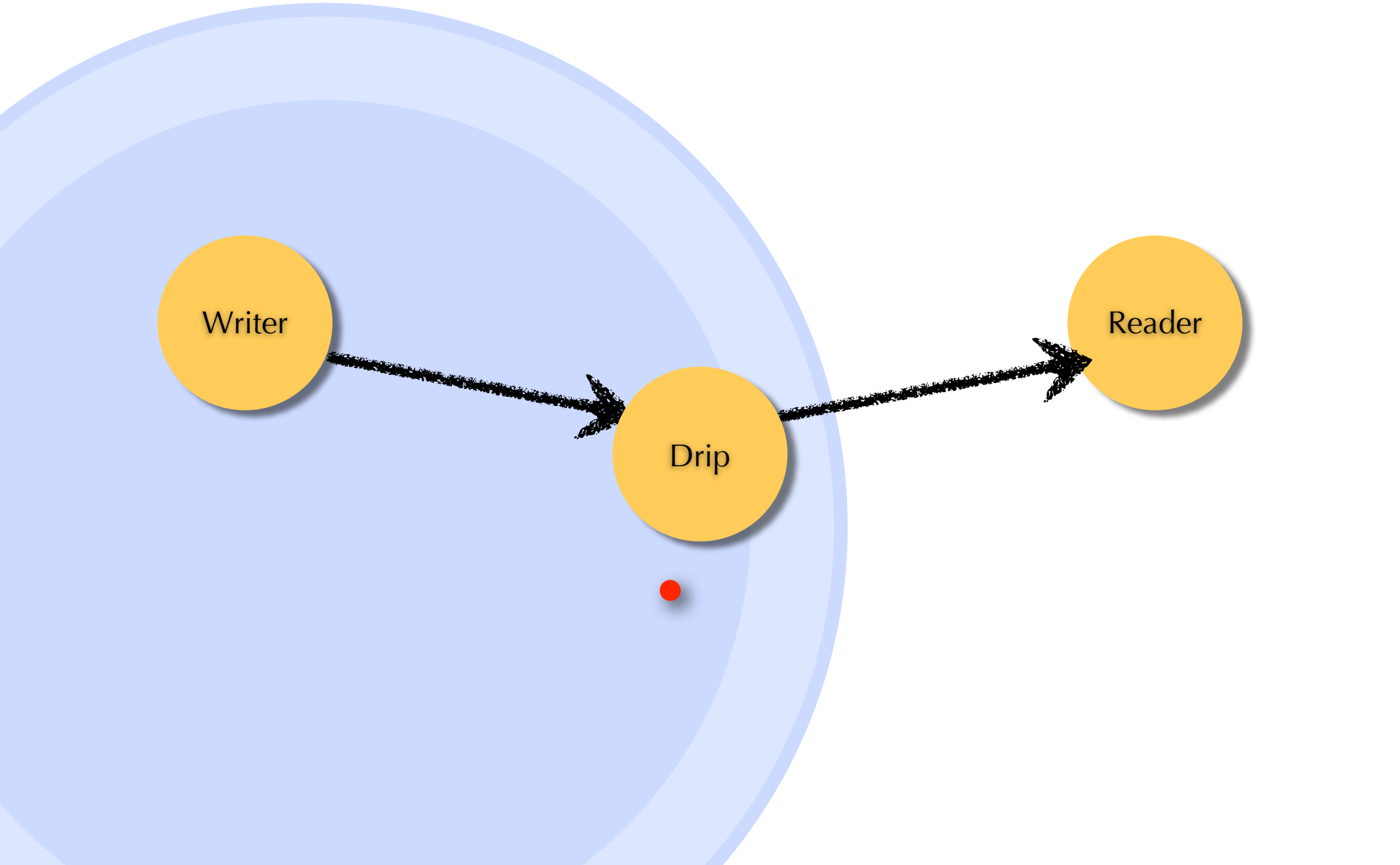
Queue#pop

- 要素は消費される

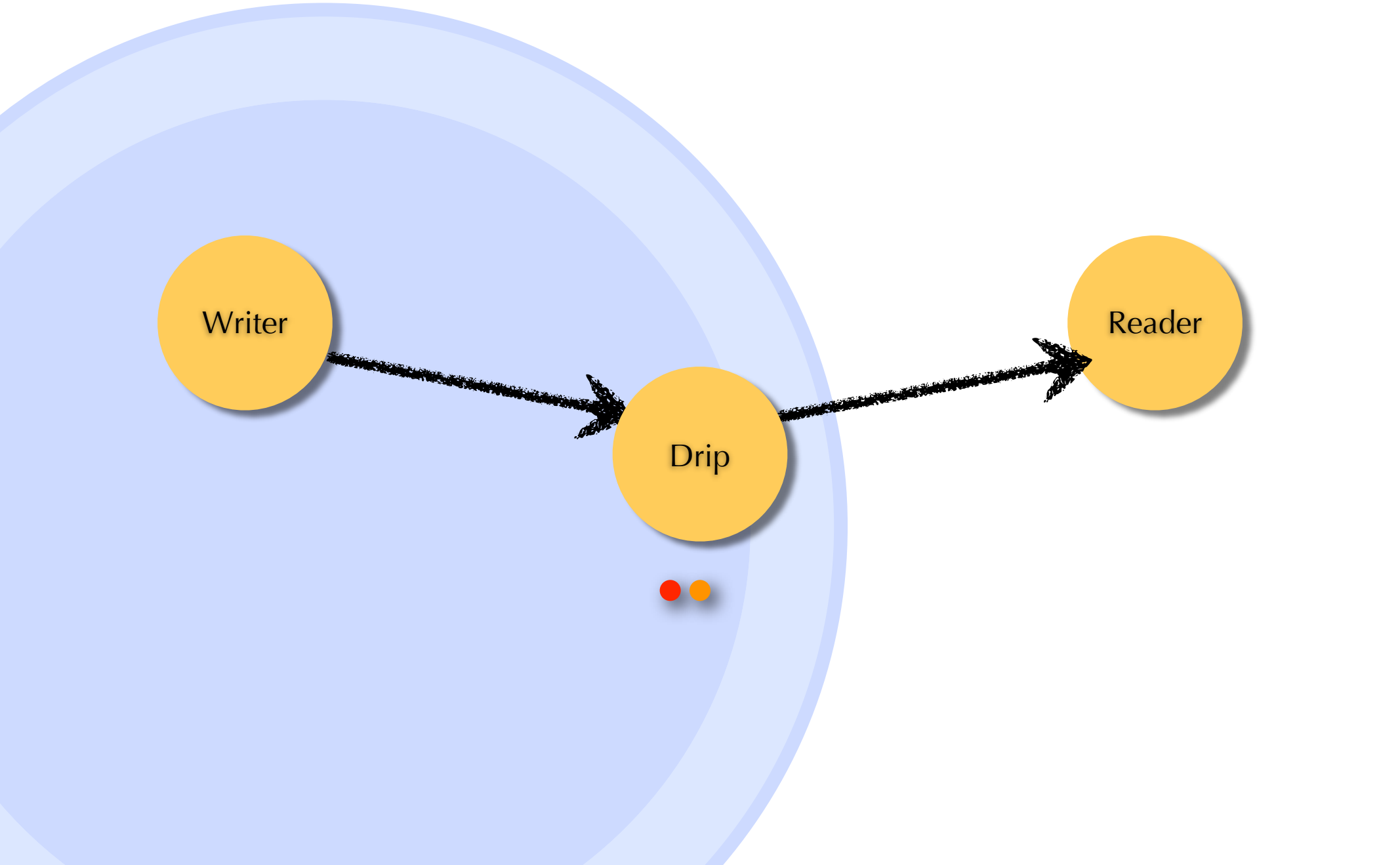
Dripだとこんな感じ



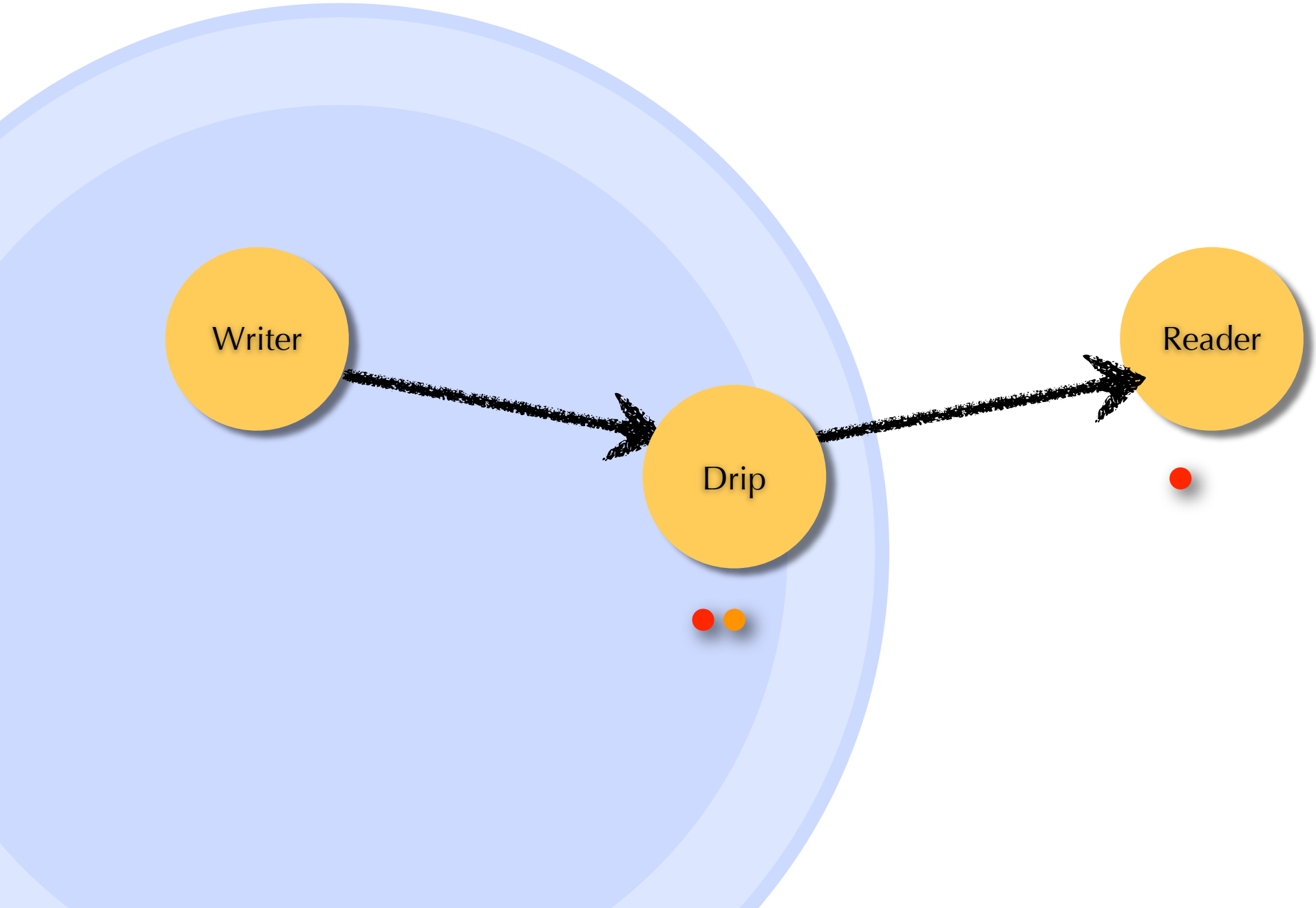
Dripだとこんな感じ



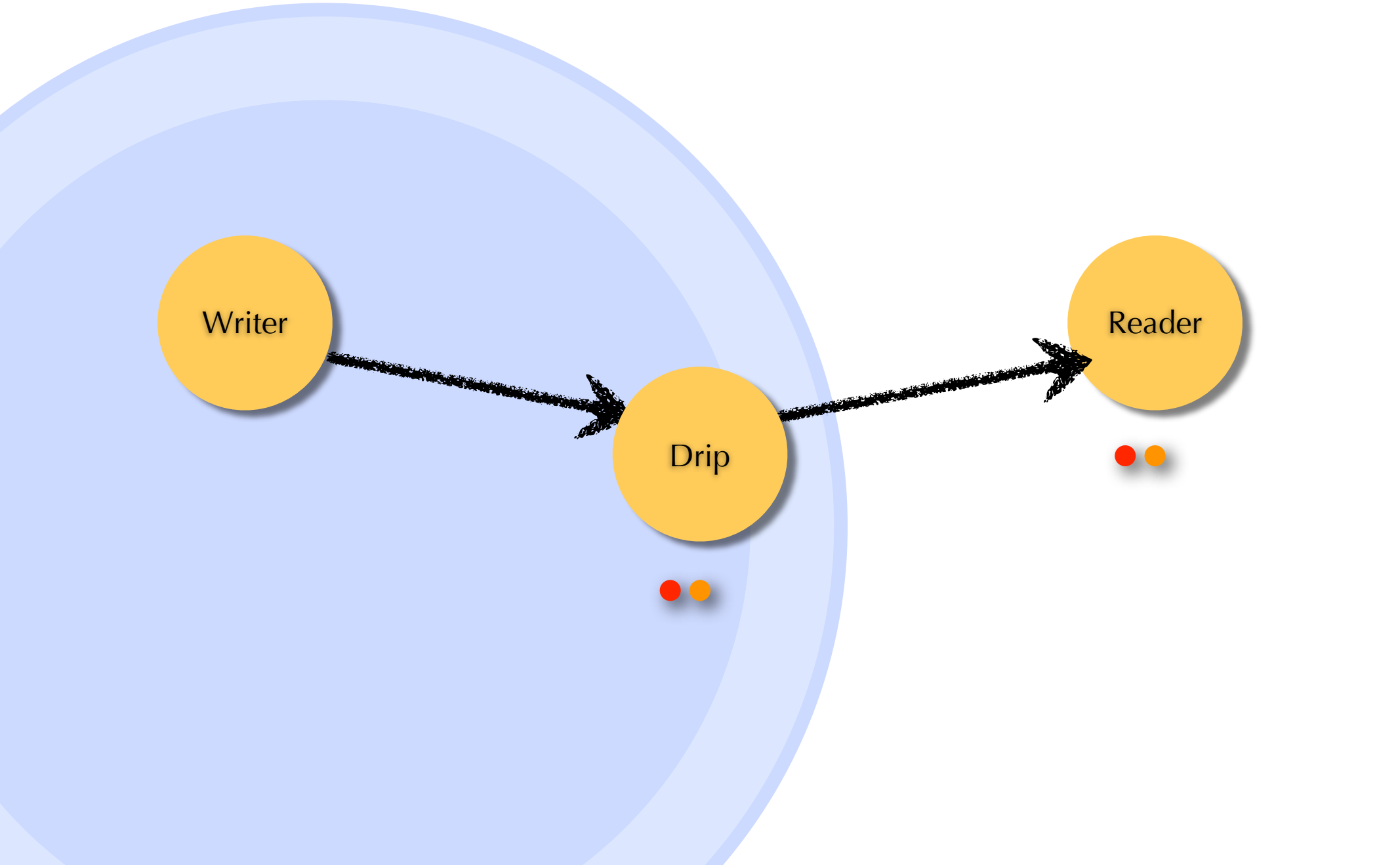
Dripだとこんな感じ



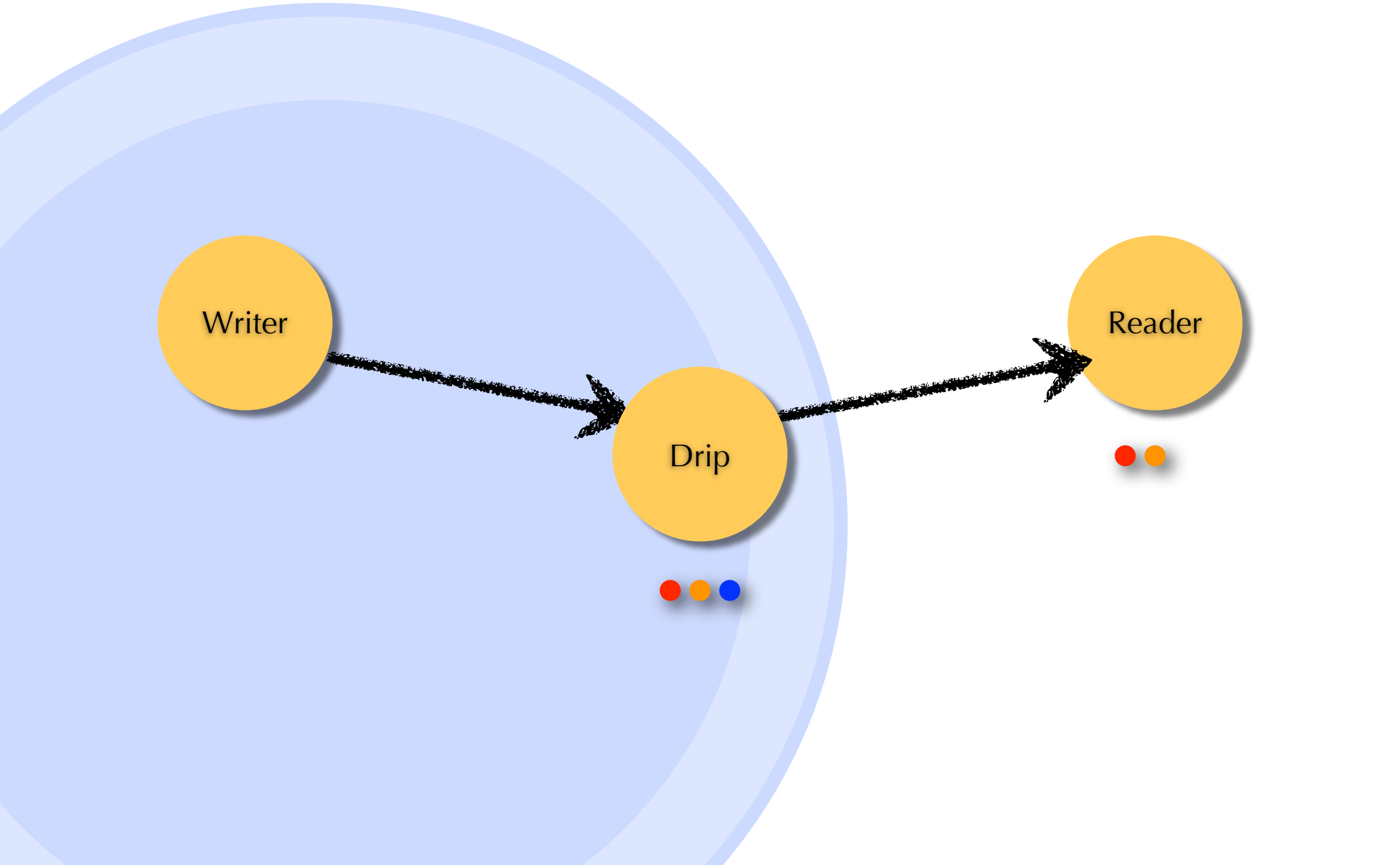
Dripだとこんな感じ



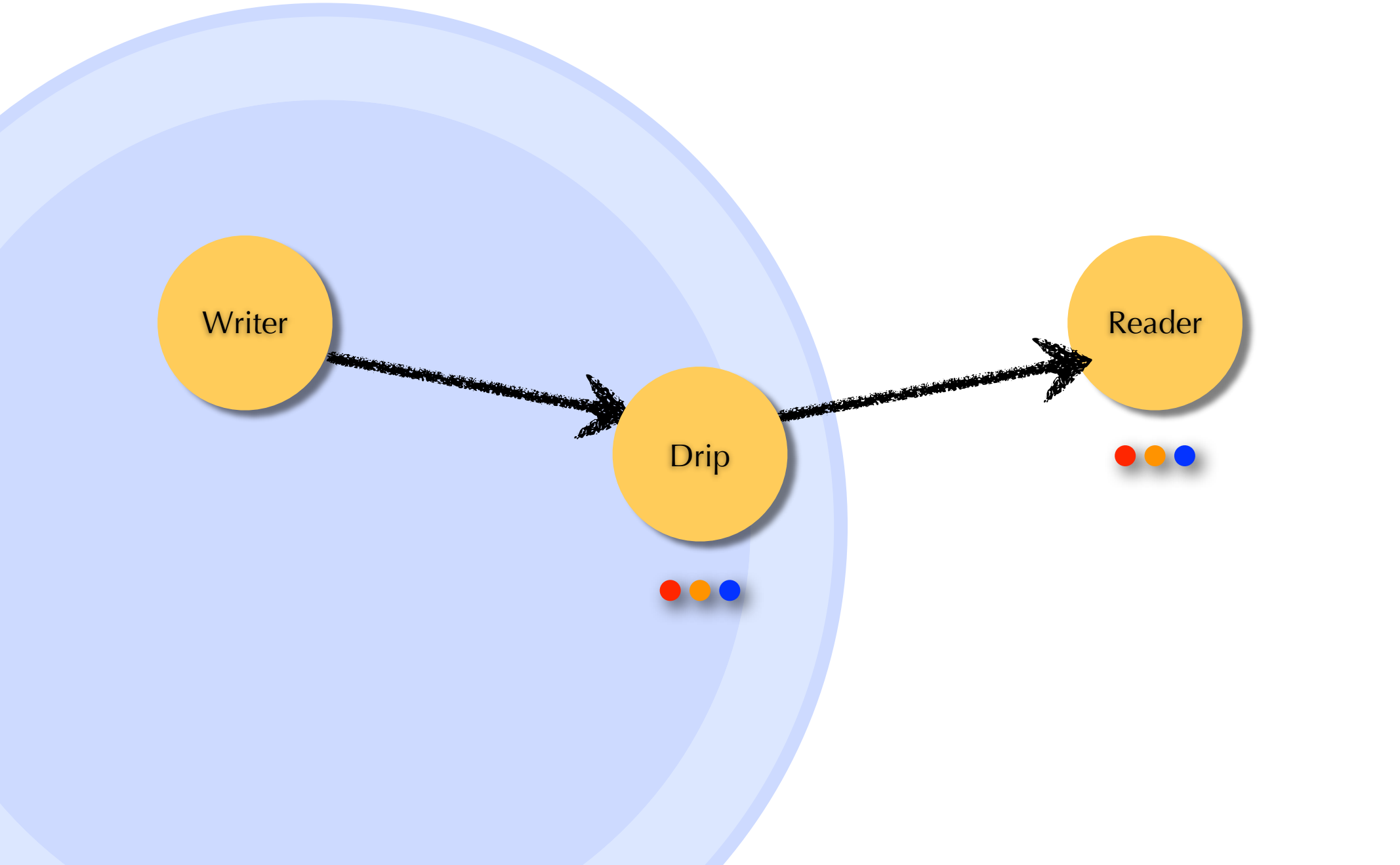
Dripだとこんな感じ



Dripだとこんな感じ



Dripだとこんな感じ



Queueと違う

- 要素は減らない
  - 何度でも読める・何人でも読める
  - 注目点を移動させて読む

Queueと同じ

- データの到着を待つことができる

TupleSpaceと違う

- タプルの紛失を気にしなくて良い
  - 要素は減らない

ここで使うAPI

- Drip#write
- Drip#read

Drip#write

```
def write(obj, *tags)
```

- オブジェクトを書き込む
- Dripの状態を変化させる唯一のメソッド
- タグを指定できる

Drip#read

```
def read(key, n=1, at_least=1, timeout=nil)
```

- 「これより新しいのn個よこせ」
  - keyの次にある要素を最大n個、最小でもat_least個読む

writer

```
drip.write('hello')
drip.write('world')
```

key	value
2	'hello'
3	'world'

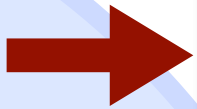
reader

```
def reader(drip, k = 0)
  while true
    k, v = drip.read(k, 1)[0]
    yield(v)
  end
rescue
  return k
end
```

key	value
2	'hello'
3	'world'

read(0, 1)

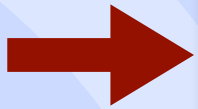
```
drip.read(0, 1)[0]
# [2, 'hello']
```



key	value
2	'hello'
3	'world'

read(2, 1)

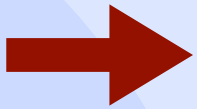
```
drip.read(2, 1)[0]
# [3, 'world']
```



key	value
2	'hello'
3	'world'

waiting...

```
drip.read(3, 1)[0]
```



key	value
2	'hello'
3	'world'

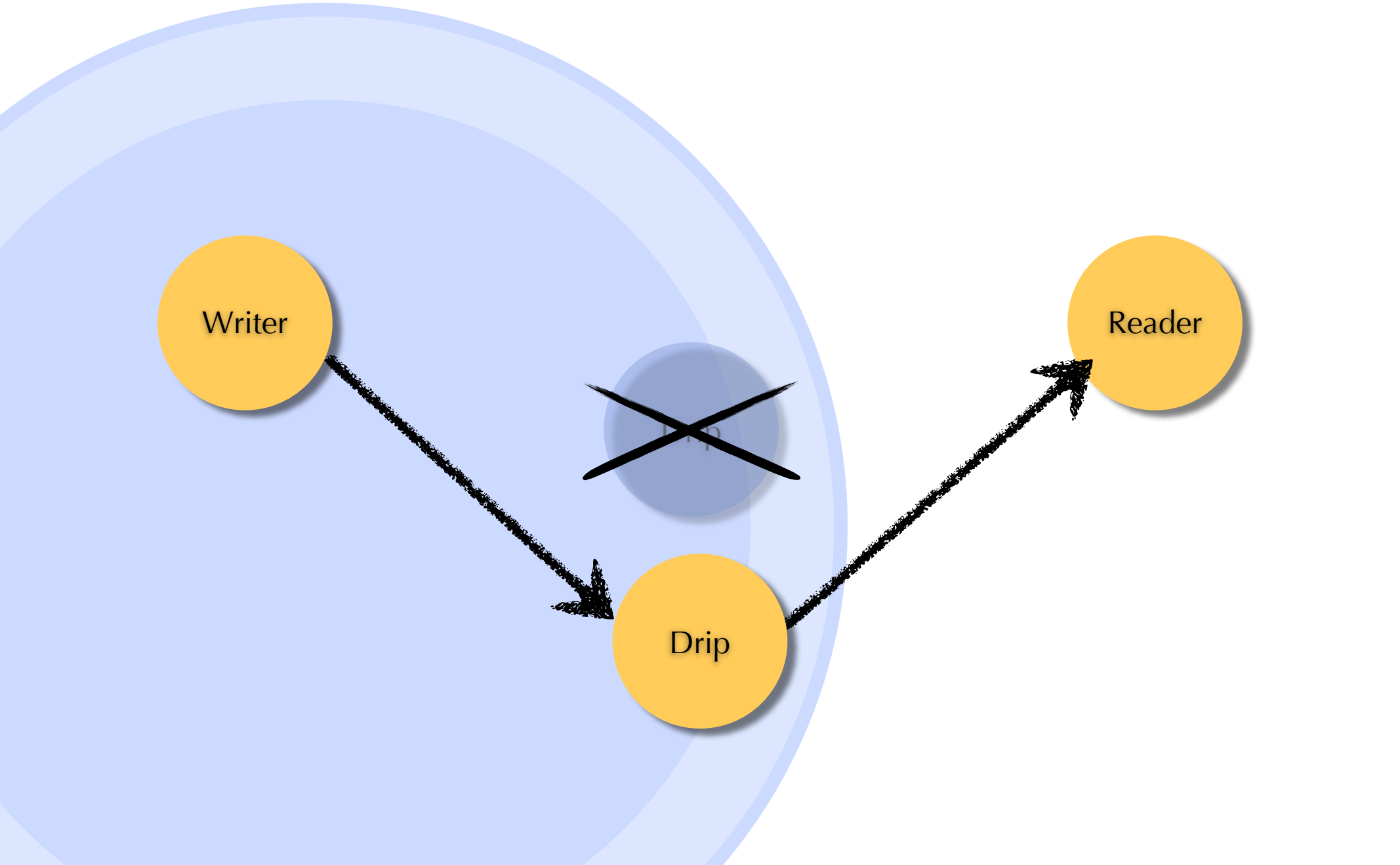
write, and read

```
drip.write('Hello, Again.')
```

```
drip.read(3, 1)[0]
# [5, 'Hello, Again.']
```

key	value
2	'hello'
3	'world'
5	'Hello, Again.'

Dripを再起動

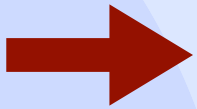


Dripを再起動

- Dripはさっきの状態で生き返る
- readerは注目点を覚えておく

再開

```
drip.read(5, 1)[0]
```

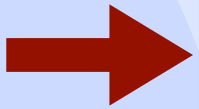


key	value
2	'hello'
3	'world'
5	'Hello, Again.'

write, and read

```
drip.write('RubyKaigi')
```

```
drip.read(5, 1)[0]
# [8, 'RubyKaigi']
```



key	value
2	'hello'
3	'world'
5	'Hello, Again.'
8	'RubyKaigi'

バッチ処理は失敗するじゃん

- なんとかやり直せる協調の仕組みを目指す
- 失敗してもさっきのところから始められる
  - 最初からやり直してもかまわない

しばらくお待ちください

- ✓ +8:00
- ✓ Queueは減るけど、Dripは減らないよ
- ✓ バッチ処理は失敗するよ
- ✓ 工夫したらぎりぎりやり直せるよ
- ✓ 次はHashで

Hashぽく

- 辞書として使う
  - タグを使うと履歴つきの辞書になるよ

タグ

- オブジェクトにタグを付けて整理できる
- 複数のタグを付けられる

タグをKVSのキーだと思う

- タグをつけてwriteする→値の設定
- タグをもつ最新の要素をreadする→値の取得
- keyよりも旧いタグをもつ要素をreadする→履歴のブラウズ

コードで

- もうちょっとコードっぽく説明します

ここで使うAPI

- Drip#head
- Drip#read_tag

Drip#head

```
def head(n=1, tag=nil)
```

- 最新の要素をn個返す
- tagを指定したらそのタグを持つものだけ

Drip#read_tag

```
def read_tag(key, tag, n=1, at_least=1, timeout=nil)
```

- だいたいread
- keyの次にある要素を最大n個、最小でもat_least個読む
  - ただし、tagを持つ要素だけ

drip['seki.age']=29

```
drip.write(29, 'seki.age')
```

key	tag	value
2	'seki.age'	29

drip['seki.age']

```
drip.head(1, 'seki.age')
# [[2, 29, 'seki.age']]
```

key	tag	value
2	'seki.age'	29

drip['seki.age']=49

```
drip.write(49, 'seki.age')
```

key	tag	value
2	'seki.age'	29
3	'seki.age'	49

drip['seki.age']

```
drip.head(1, 'seki.age')
# [[3, 49, 'seki.age']]
```

key	tag	value
2	'seki.age'	29
3	'seki.age'	49

drip['sora_h.age']

```
drip.head(1, 'sora_h.age')
# []
```

key	tag	value
2	'seki.age'	29
3	'seki.age'	49

waiting...

```
drip.read_tag(0, 'sora_h.age')
```

key	tag	value
2	'seki.age'	29
3	'seki.age'	49

write and read_tag

```
drip.write(12, 'sora_h.age')
```

```
drip.read_tag(0, 'sora_h.age')
# [5, 12, 'sora_h.age']
```

key	tag	value
2	'seki.age'	29
3	'seki.age'	49
5	'sora_h.age'	12

Hashと同じ

- hash[key]=value
  - drip.write(value, key)
- hash[key]
  - drip.head(1, key)

Hashと違う

- 要素は消せない
- 履歴がある
  - drip.head(5, key)
- keys/eachがない
  - 遺恨を残す気がするので作ったけど消した

TupleSpaceと同じ

- ある要素の追加や更新を待てる
  - パターンは限定されてるけど

Hashの代わりになる？

- 簡単な辞書としては使えそう
- 削除はないけどnilとかで代用できるかも
- keys/eachはできない
  - わざと

PTupleSpaceとの違い

- TupleSpaceらしさをばっさり切ってる
- 永続化を前提として協調機構を考えなおした
- TupleSpace違うかっこよさ

しばらくお待ちください

- ✓ +13:00
- ✓ 履歴つきの辞書になるよ
- ✓ 待合せもできるよ
- ✓ 自画自賛はほどほどにする
- ✓ ちがう言葉で説明しなおす

おさらい

- Dripは待合せのできるログ
  - 追記だけ
  - 削除・修正はできない
  - 新しいログが書かれるまでブロック
- タグでフィルタ

一次元のストリームも

- 見ようと思えばQueueにもHashにも見える
- その他の構造に見ることもきっとできる
  - たいていの集合はeachできるんだもん
- 信じるっていうのはそういうこと

ブラウズの例

- 未来へ - read, read_tag, newer
- 過去へ - head, older
- 時間軸にそって前後へ
- タグを使ってスキップしたり

読み進める

- keyより新しい4つくれ
- 1つ揃うまで待ってて

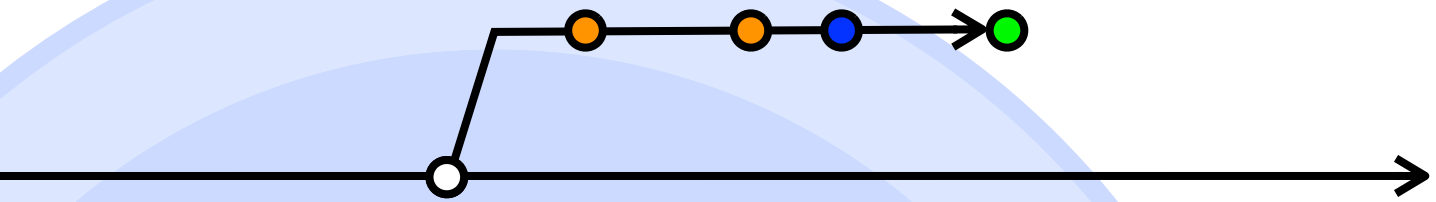
```
while ture
  ary = drip.read(key, 4, 1)
  ...
  key = ary[-1][0]
end
```

読み進める

- keyより新しい4つくれ
- 1つ揃うまで待ってて

```
while ture
  ary = drip.read(key, 4, 1)
  ...
  key = ary[-1][0]
end
```

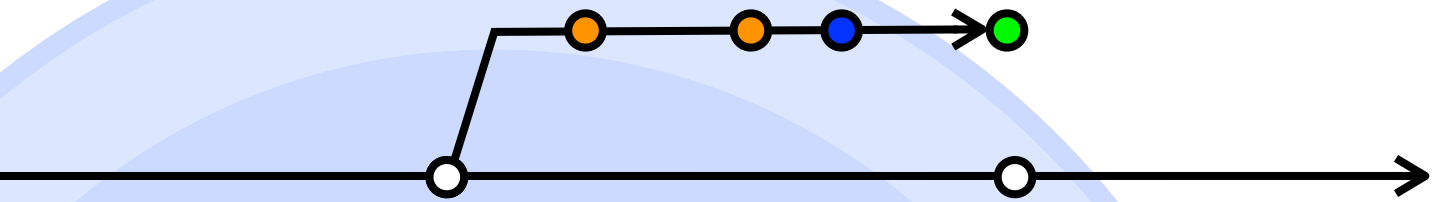
読み進める



- keyより新しい4つくれ
- 1つ揃うまで待ってて

```
while ture
  ary = drip.read(key, 4, 1)
  ...
  key = ary[-1][0]
end
```

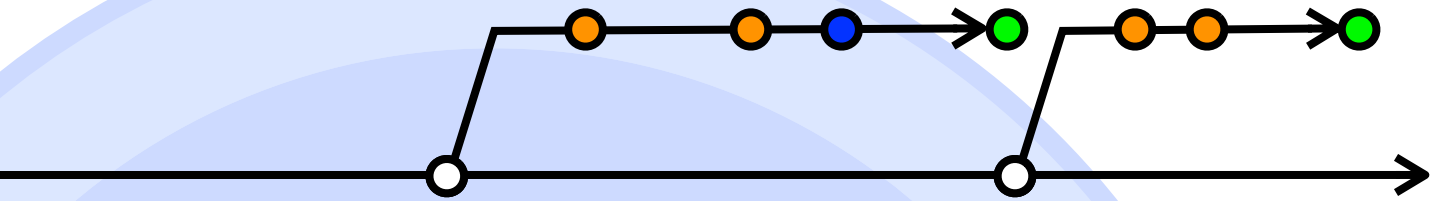
読み進める



- keyより新しい4つくれ
- 1つ揃うまで待ってて

```
while ture
  ary = drip.read(key, 4, 1)
  ...
  key = ary[-1][0]
end
```

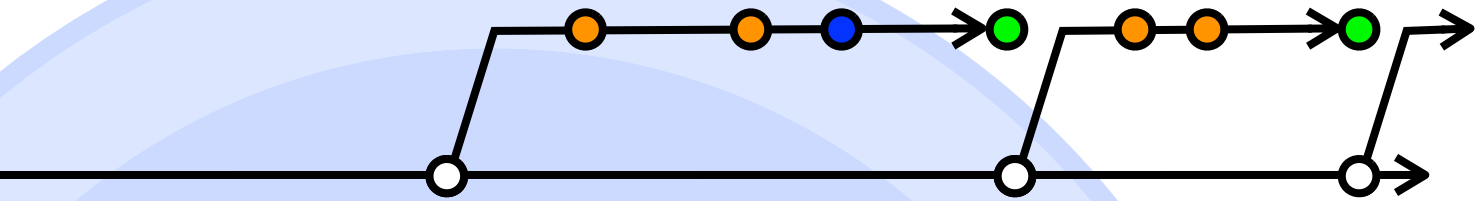
読み進める



- keyより新しい4つくれ
- 1つ揃うまで待ってて

```
while ture
  ary = drip.read(key, 4, 1)
  ...
  key = ary[-1][0]
end
```

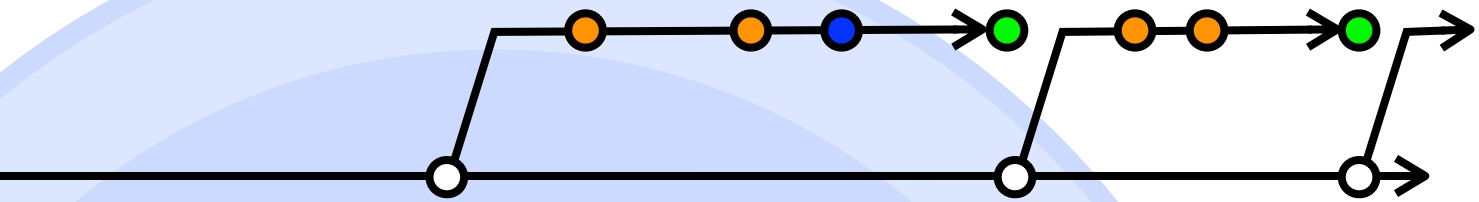
読み進める



- keyより新しい4つくれ
- 1つ揃うまで待ってて

```
while ture
  ary = drip.read(key, 4, 1)
  ...
  key = ary[-1][0]
end
```

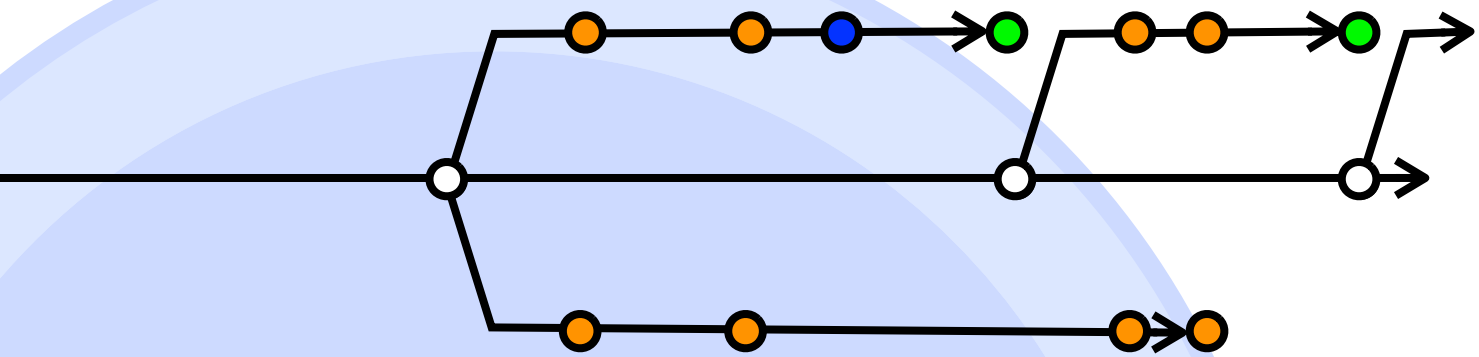
フィルタして読む



- 興味があるものだけ返す

```
read_tag(key, 'orange', 4, 1)
```

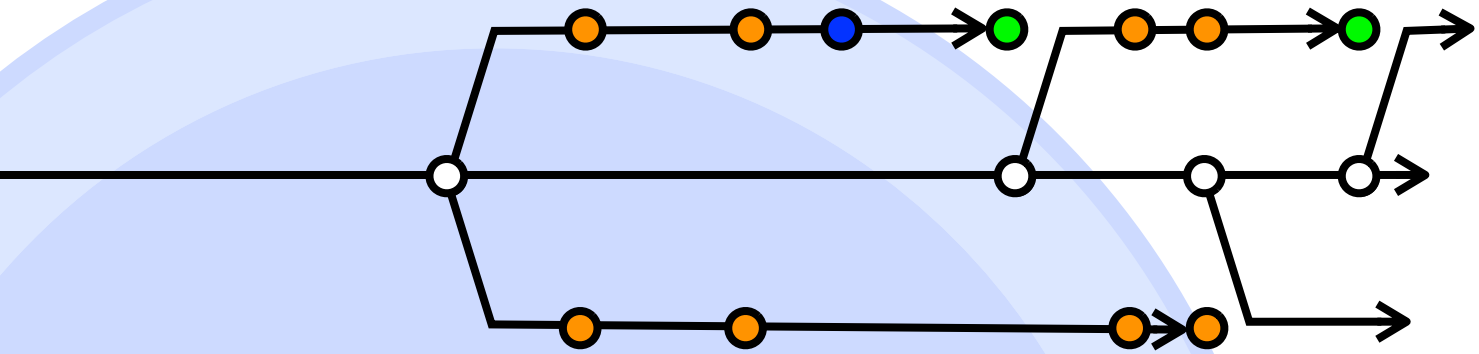
フィルタして読む



- 興味があるものだけ返す

```
read_tag(key, 'orange', 4, 1)
```

フィルタして読む



- 興味があるものだけ返す

```
read_tag(key, 'orange', 4, 1)
```

ちょっと戻ってから読む

- 最新の'blue'から5つずつください

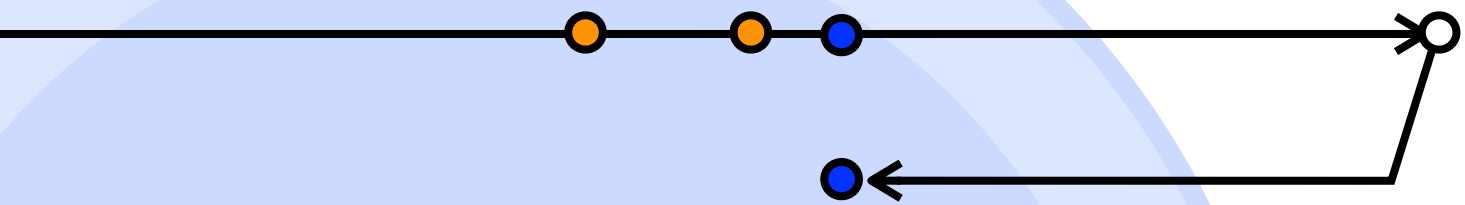
```
k, v = head(1, 'blue')[0]
read(k, 5)
```

ちょっと戻ってから読む

- 最新の'blue'から5つずつください

```
k, v = head(1, 'blue')[0]
read(k, 5)
```

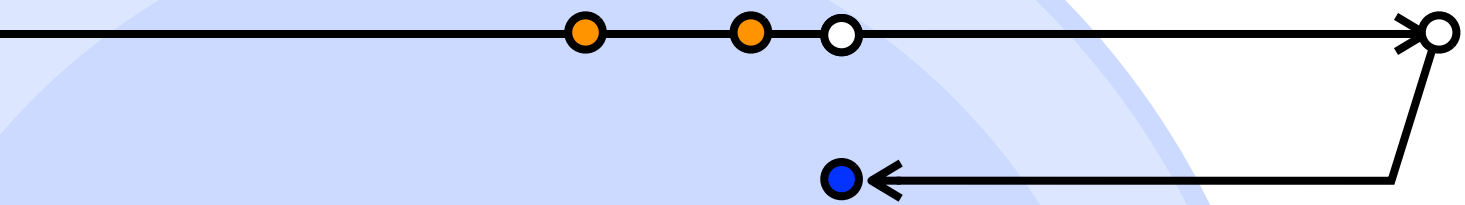
ちょっと戻ってから読む



- 最新の'blue'から5つずつください

```
k, v = head(1, 'blue')[0]
read(k, 5)
```

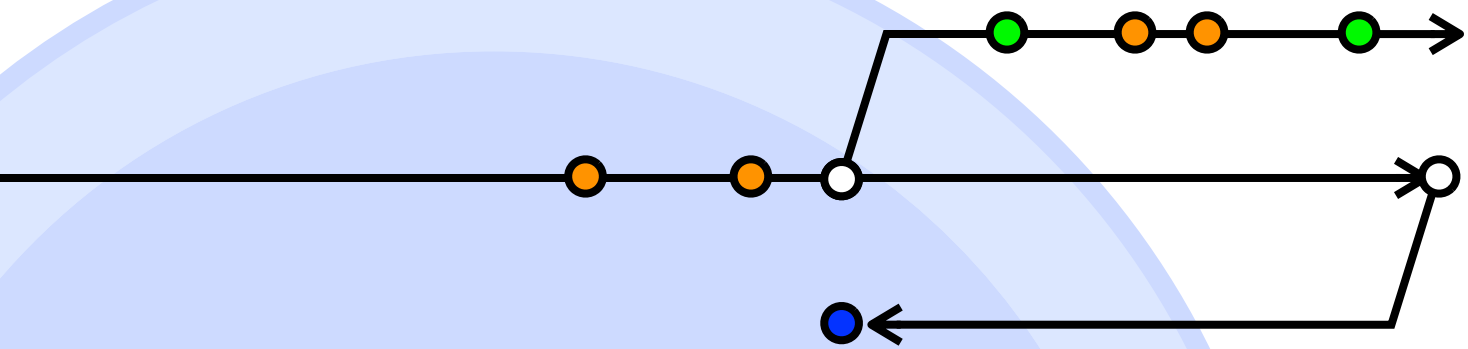
ちょっと戻ってから読む



- 最新の'blue'から5つずつください

```
k, v = head(1, 'blue')[0]
read(k, 5)
```

ちょっと戻ってから読む



- 最新の'blue'から5つずつください

```
k, v = head(1, 'blue')[0]
read(k, 5)
```

しばらくお待ちください

- ✓ +18:00
- ✓ ブラウズのようすをなんとなくわかって
- ✓ 次は実装について

実装について

- Key
- Index

Key

- 識別子の安価な生成
- OODBの習作の頃からの主要なテーマ
  - 私にとってのObjectは集合の要素ではない

Key

- Timeのsecとusecから作られる整数
  - LionだとFixnum
- 非常に安い

Key

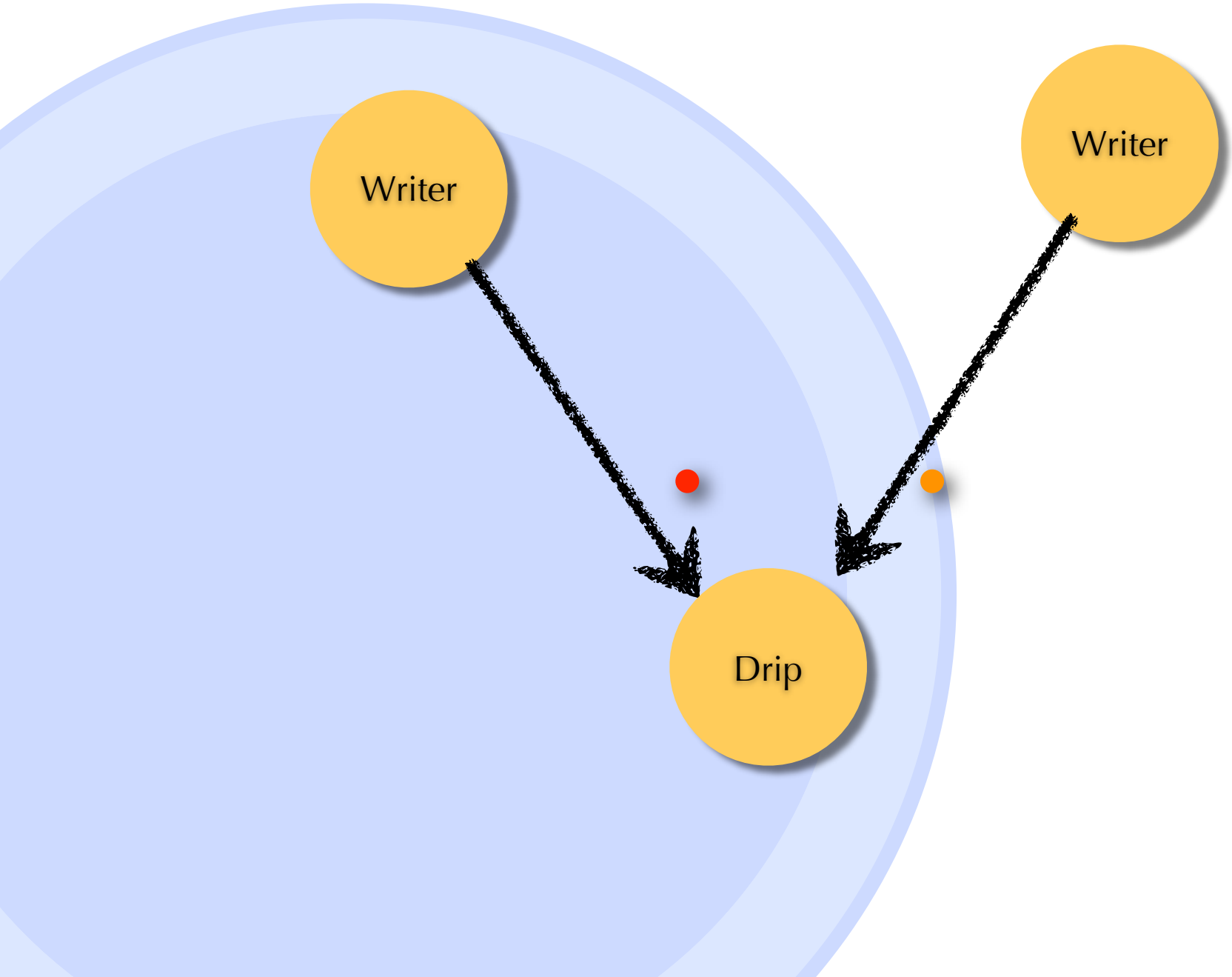
```
def time_to_key(time)
  time.tv_sec * 1000000 + time.tv_usec
end

def key_to_time(key)
  Time.at(*key.divmod(1000000))
end
```

- tv_usecの分解能

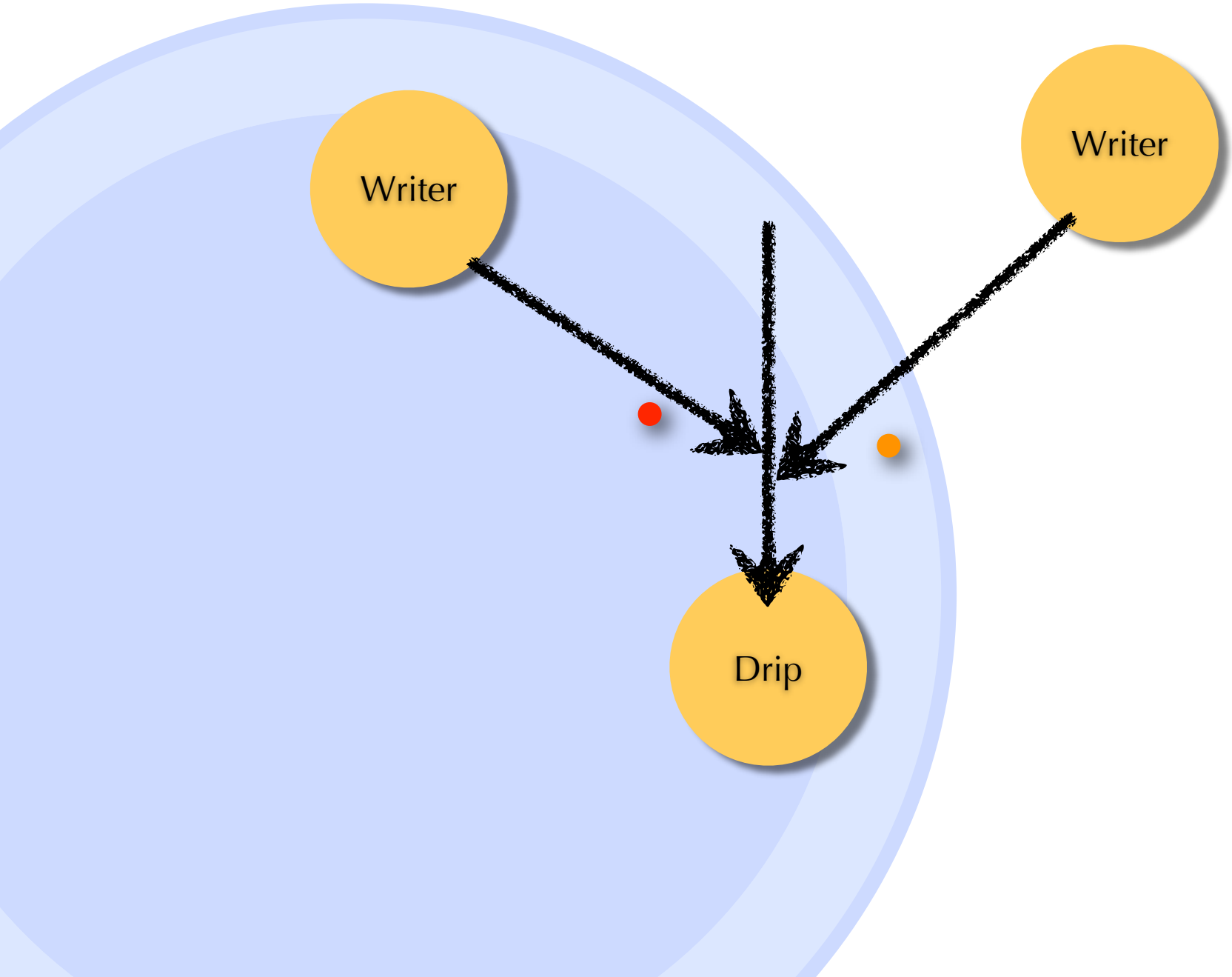
どの時刻か

- 情報の発生した時刻ではない
- Dripが受け付けた時刻
  - 世界中で同時に情報が発生しても、Dripは一つずつ順に受け付ける
- 常にシーケンシャルになる

同時に発生した情報



直列化される



コンフリクトを避ける

```
def make_key(at=Time.now)
  synchronize do |last|
    key = [time_to_key(at), last + 1].max
    yield(key)
    key
  end
end
```

- 現在の時刻 と 最後のキーの次 の大きい方
  - ほとんどぶつからないけど...

Index

- 二つの索引を持つ
  - [[key, value], [key, value], ...]
  - [[tag, key], [tag, key], ...]

key, value

- key順に並んだ集合
  - 要素は値相当
  - RBTree
- readなどで使う

tag, key

- [tag, key]の順に並んだ集合
  - 同じタグが集まる
  - やっぱりRBTree
- read_tagなどで使う

RBTree

- 赤黒木
  - RubyTreeではない
- 二分探索
- 隣の要素へのアクセス

ソート済みの配列

- 更新のない旧い集合は配列とBSearchで
  - 開発中 / ImmutableDrip
    - できてるけど飽きて仕上げてない

Partitioning

- 時刻順に分割するのが素直
  - 旧い情報は専用のマシン/プロセスに配置
  - 旬な情報は速いマシンで

Partitioning

- 時刻順に分割するのが素直
  - 旧い情報と新しい情報で集合を変える
  - 旧いのはArray、新しいのはRBTree

each

- なぜ避けたか
  - 内部イテレータはコールバックだよ問題
  - Partitioningの邪魔になる
    - タグでのループは辛い

keys

- keysなんてあり得ない
- 全部集めてRMIで返却とか

今日の話

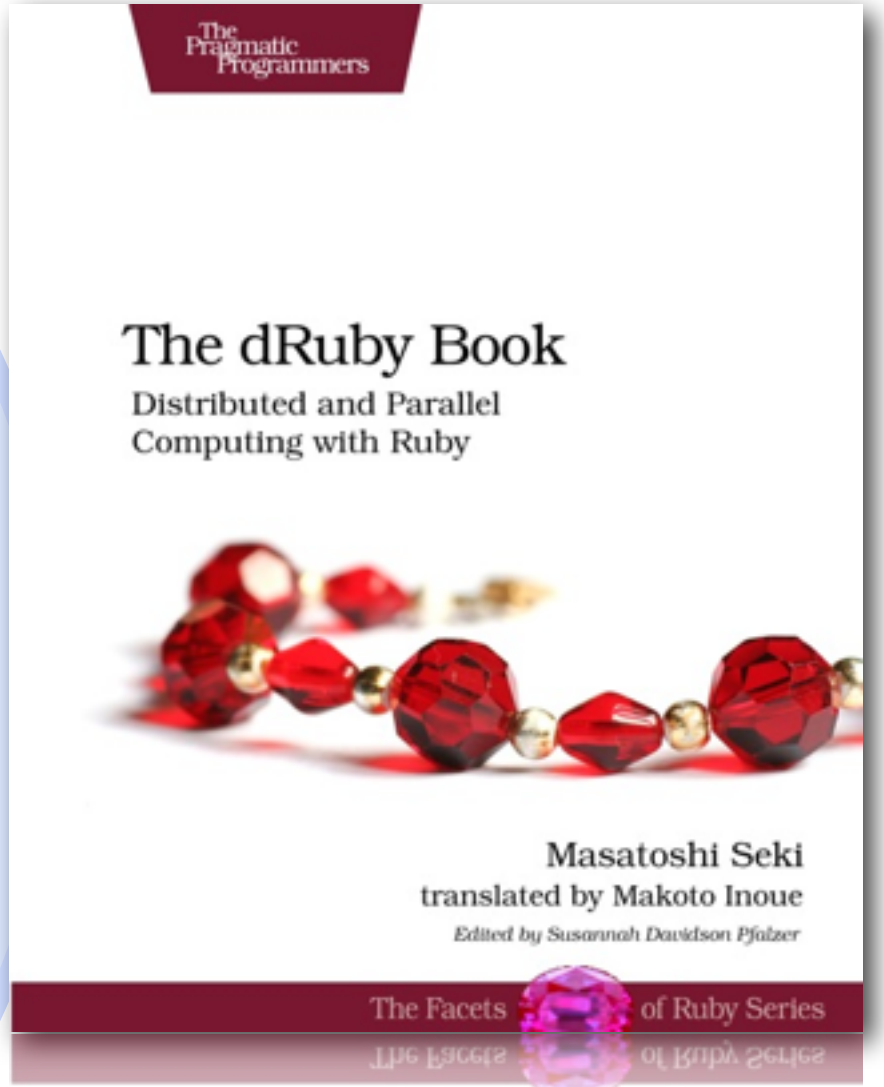
- Dripの紹介
  - かっこいいログ
  - 永続化を前提とした協調のしくみ
  - 実装の一部を紹介

日本先行発売

- まさかほんとに

The dRuby Book

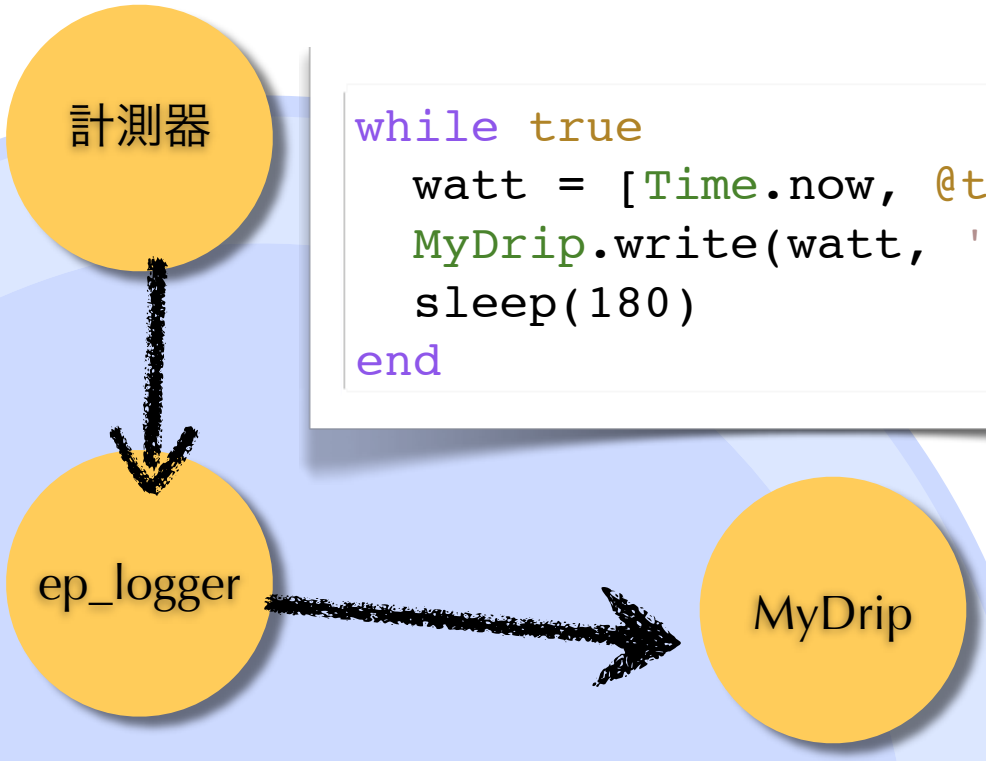
- 索引作成中
- 素敵forward by matz
- 豪華quotes
- amazonで予約now



STM

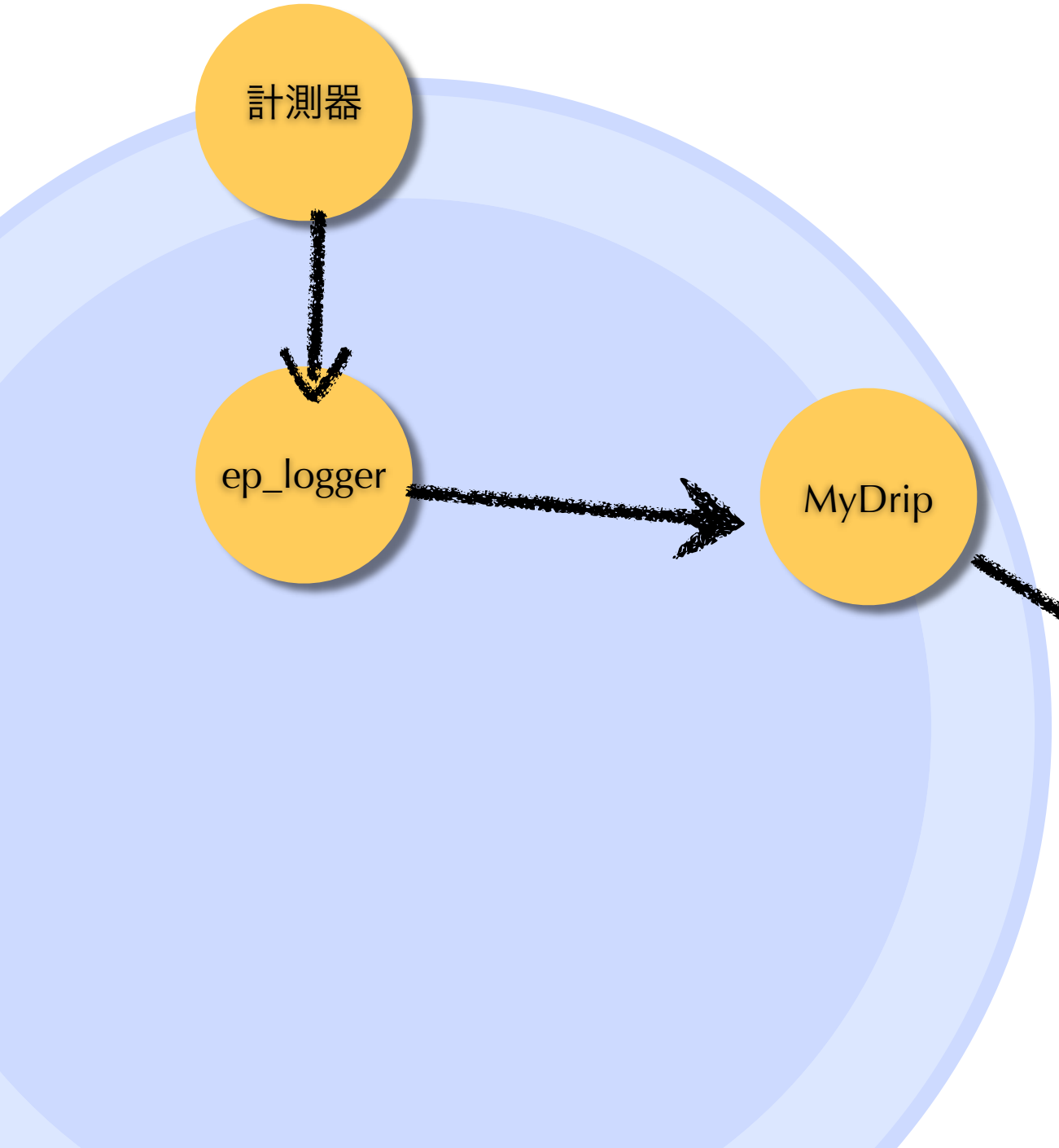
- なんとかトランザクションメモリ
- in memory版Dripを使って100行くらい
  - 情報に世代番号を持たせると簡単
    - 世代番号はDripのKeyにそっくり

電力消費量ロガー＆レポート



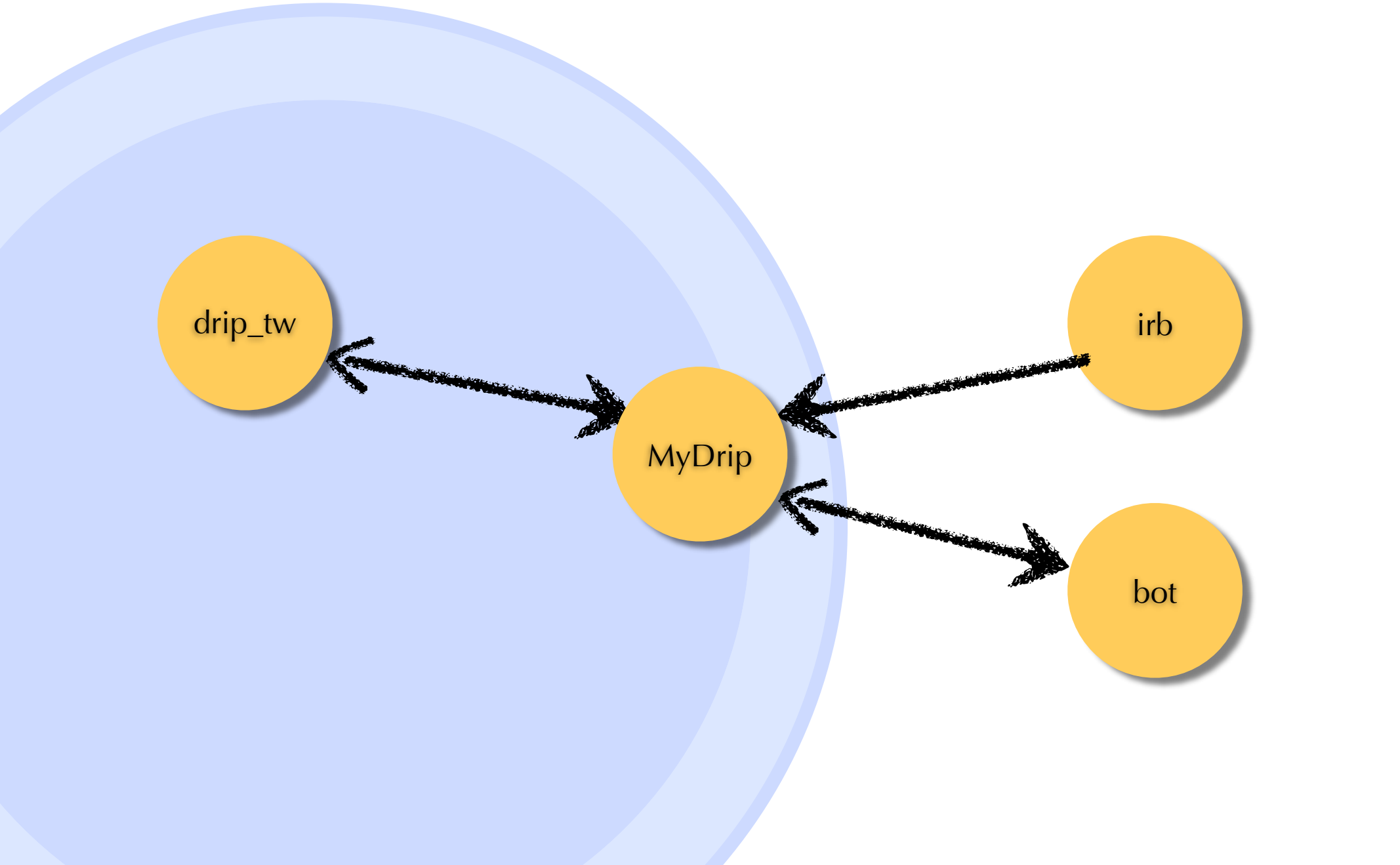

```
while true
  watt = [Time.now, @there.value]
  MyDrip.write(watt, 'Watt')
  sleep(180)
end
```

電力消費量ロガー＆レポート



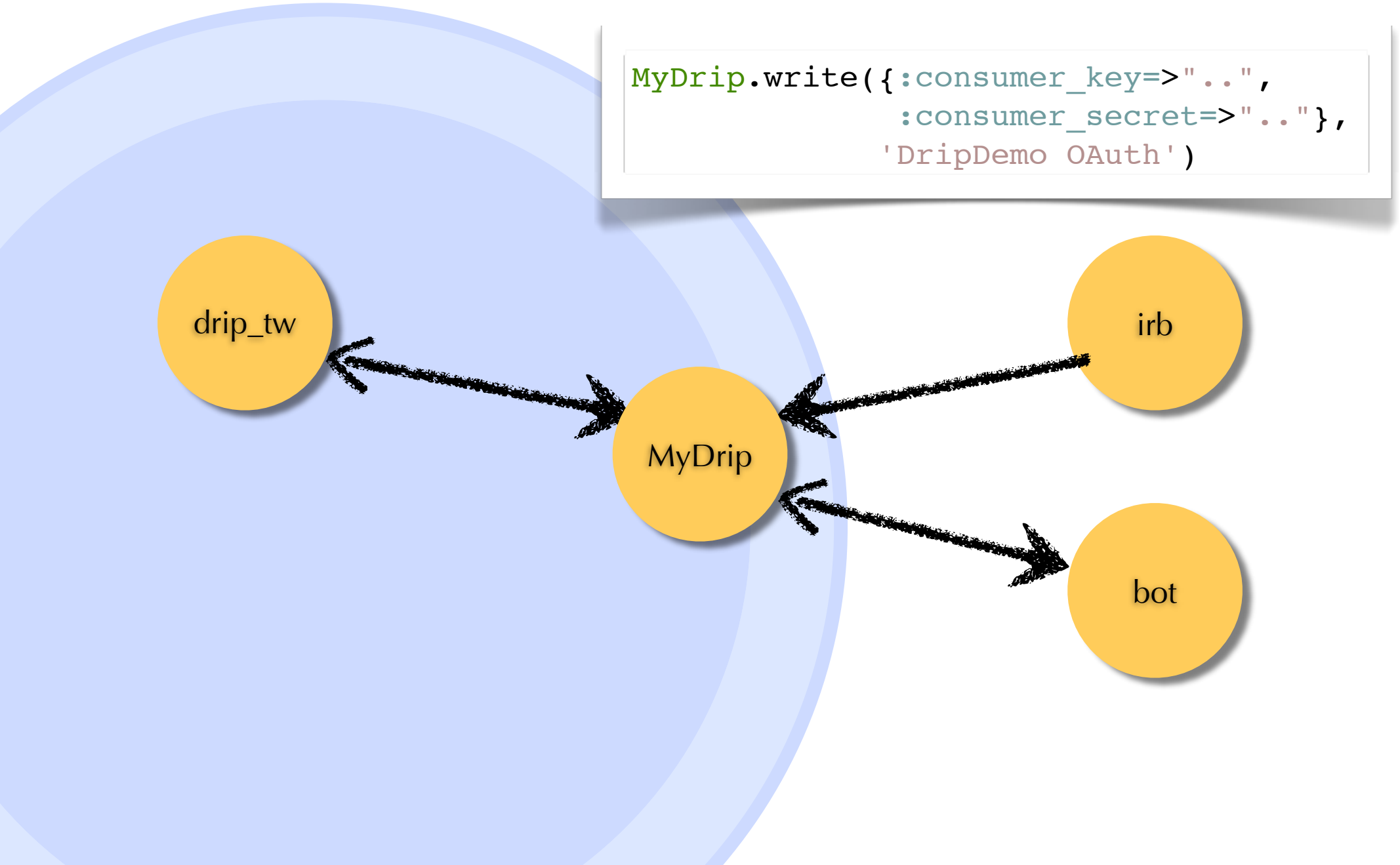

```
MyDrip.head(20, 'Watt')
```

sample/drip_tw.rb

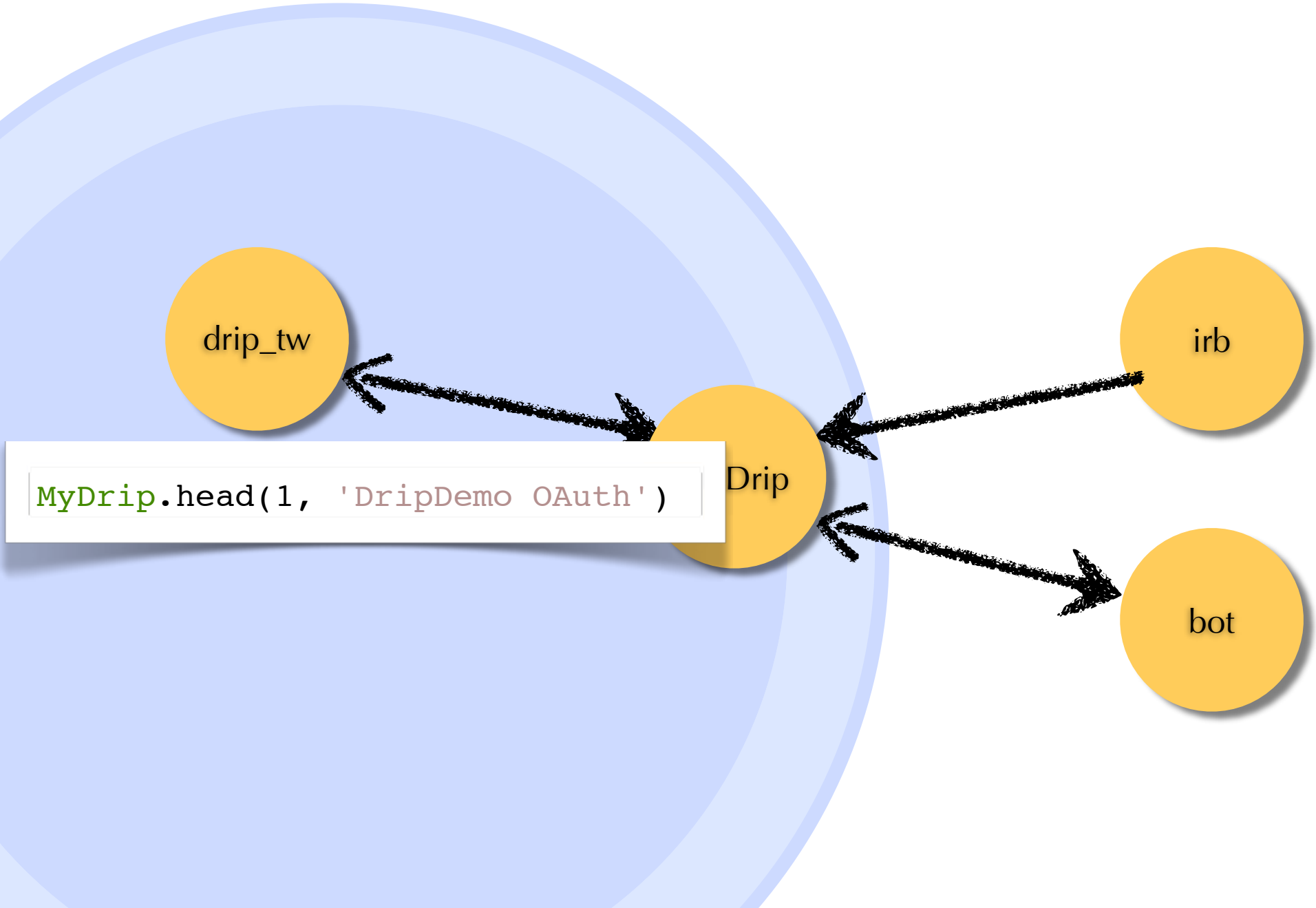


sample/drip_tw.rb

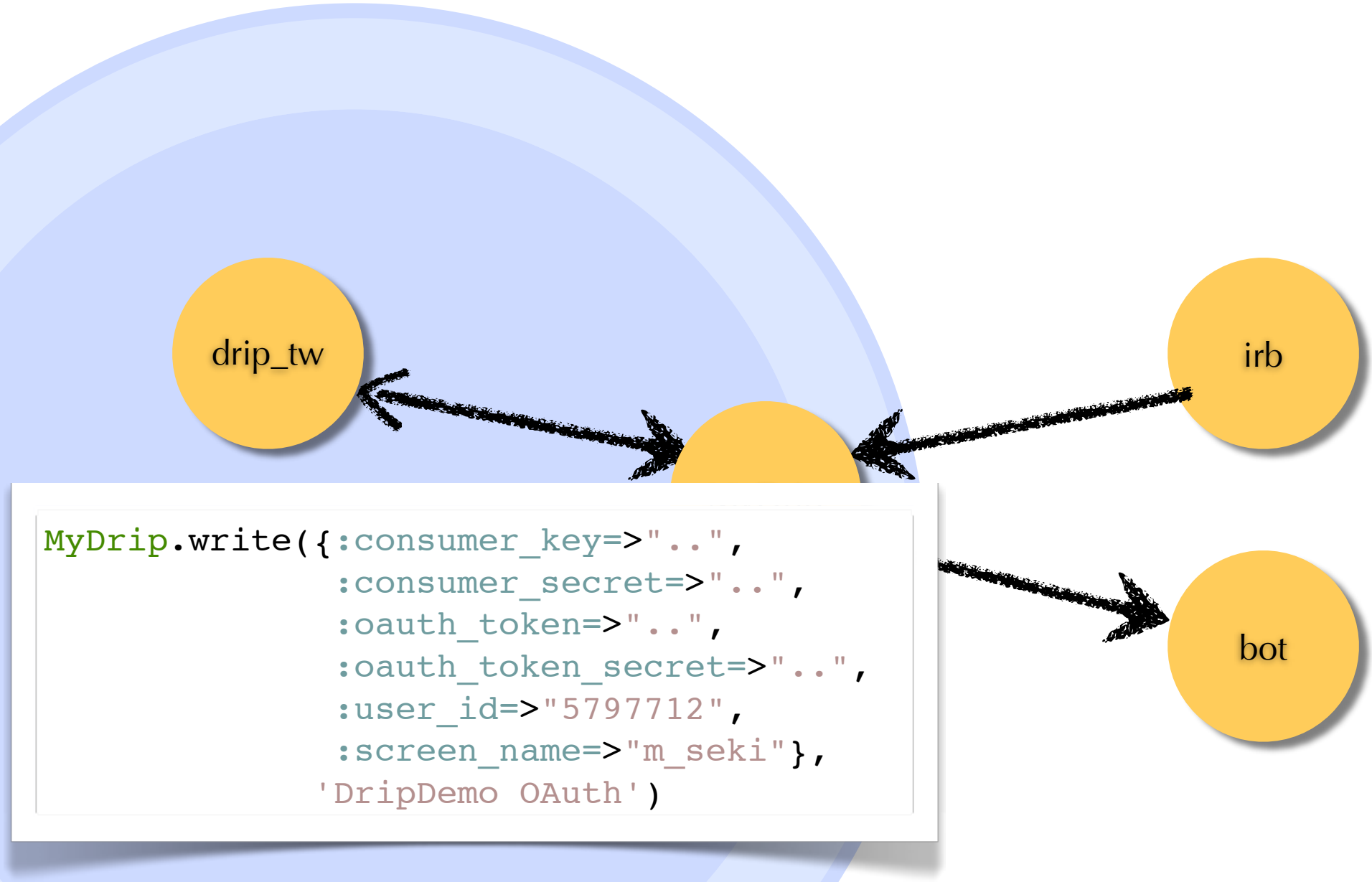
```
MyDrip.write({:consumer_key=>"..",
              :consumer_secret=>".."},
             'DripDemo OAuth')
```



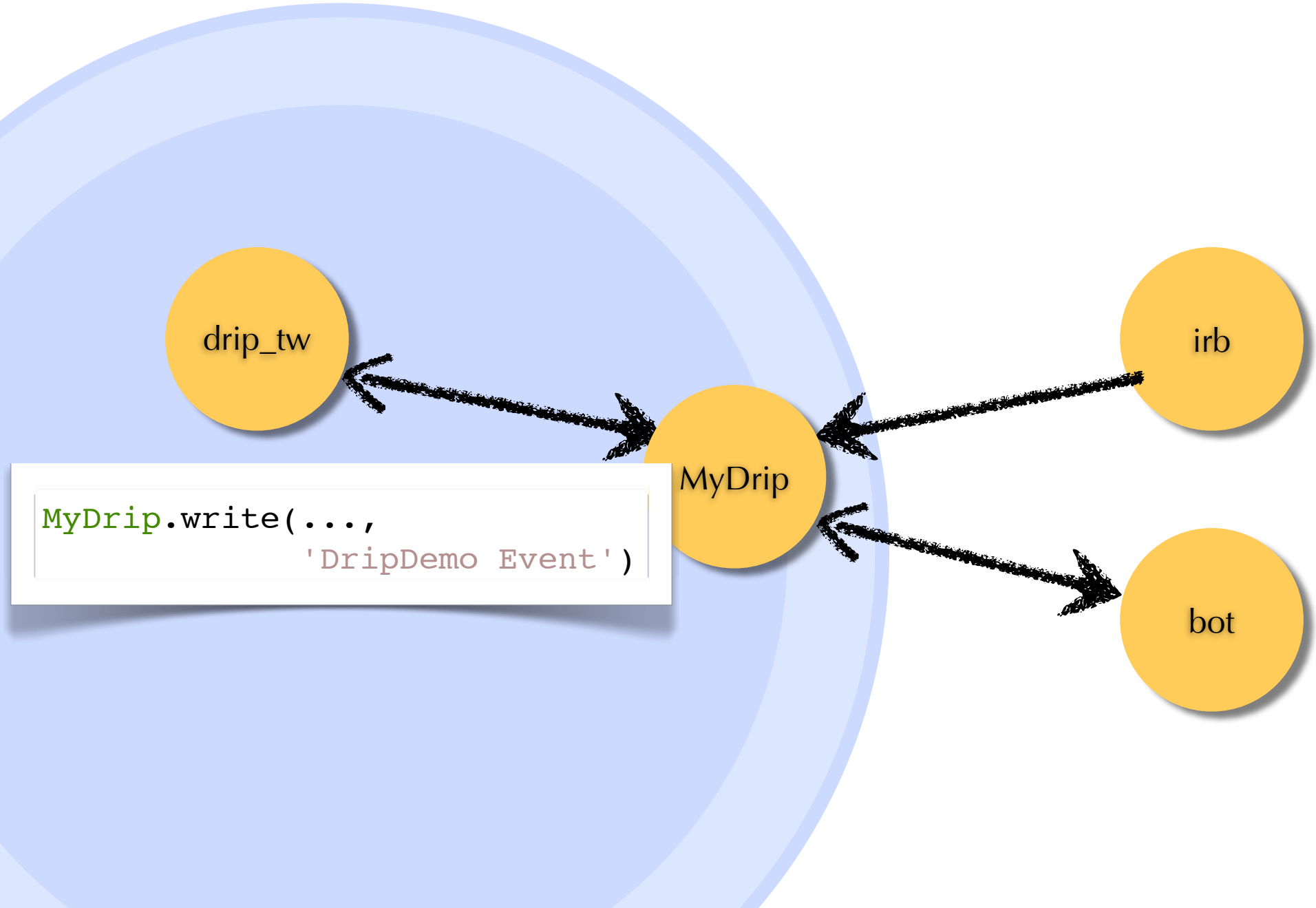
sample/drip_tw.rb



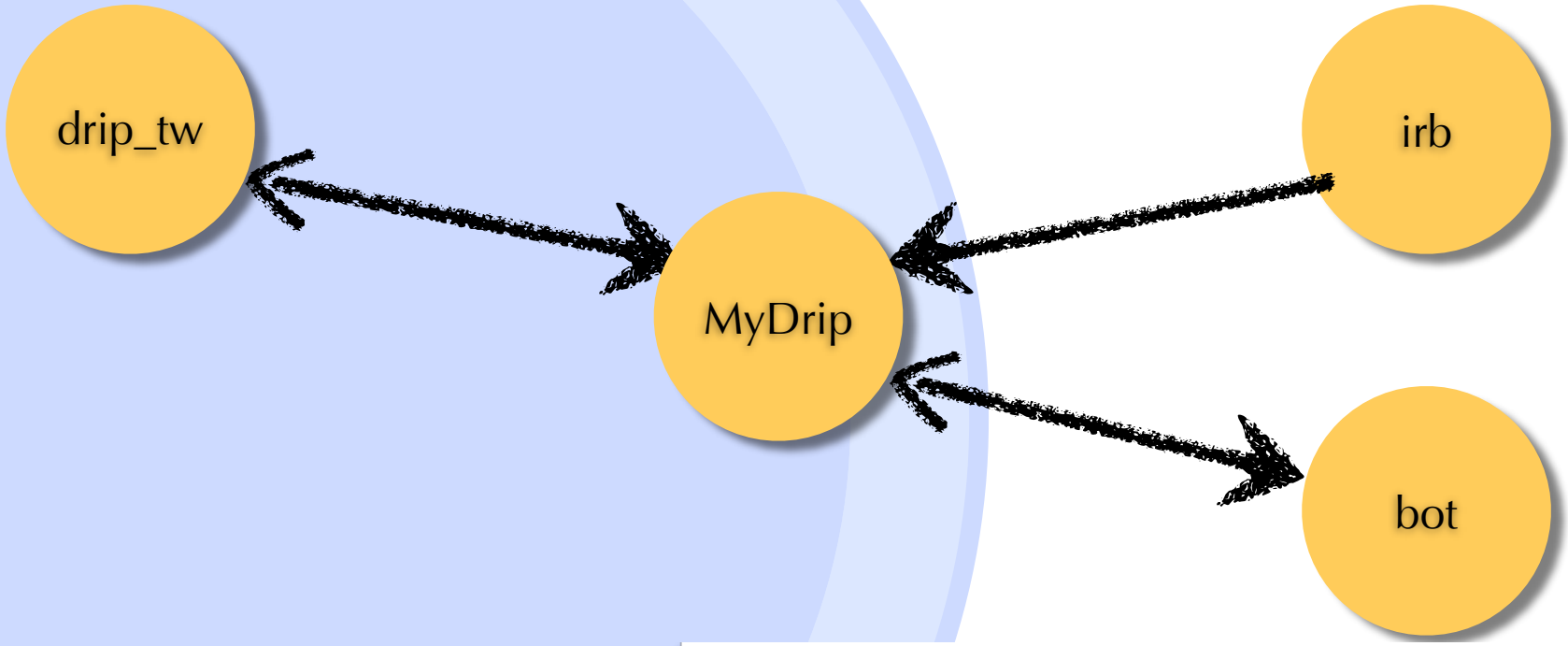
sample/drip_tw.rb



sample/drip_tw.rb

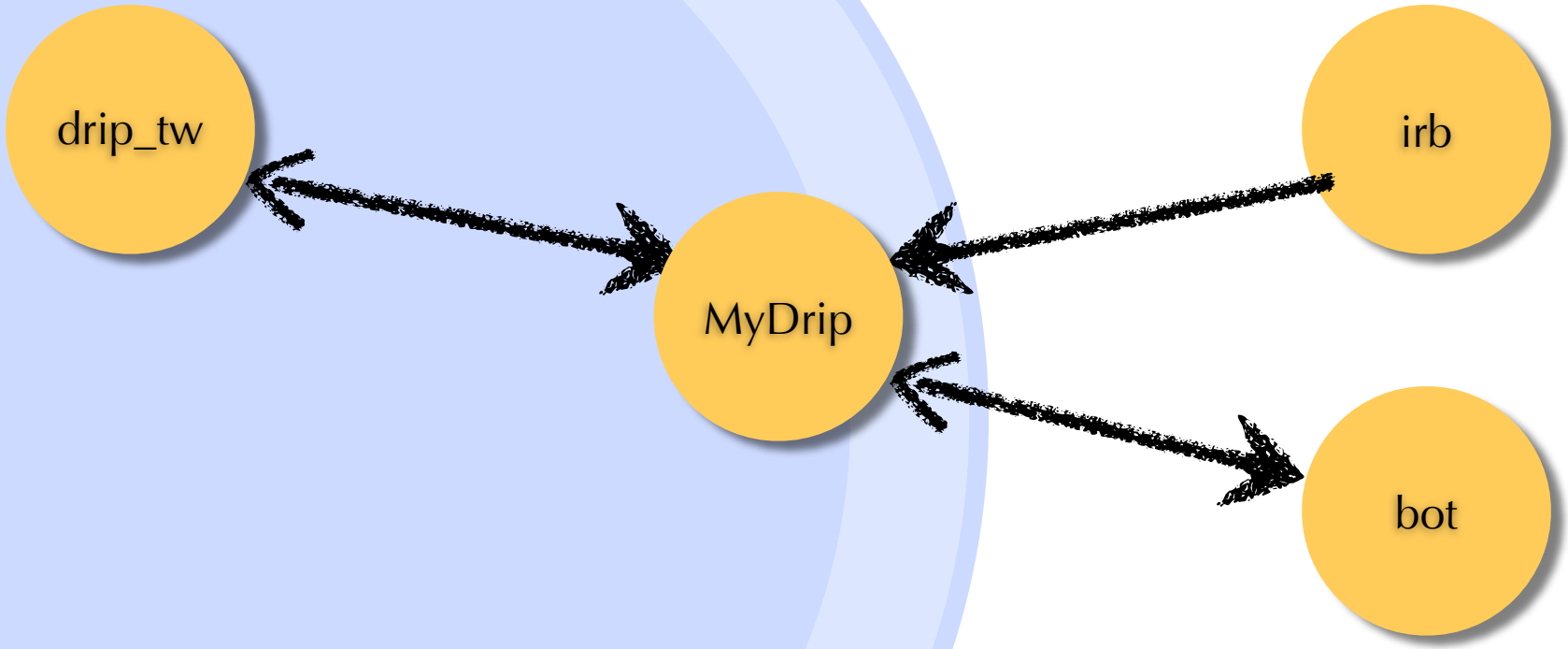


sample/drip_tw.rb




```
MyDrip.read_tag(key, 'DripDemo Event')
```

sample/drip_tw.rb




```
MyDrip.write(key, 'Bot Footprint')
```