persistent tuple space and stream

seki@ruby-lang.org

author of dRuby, Rinda, ERB and Drip

Actor

- おもしろそう

日本先行発売

- 次は英語で初刷を

ちゃんとした会社員

- わりと大きな会社
- プロの無職

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

気が弱い

- OSS開発者のみなさんは提案を断れますか？

たくさんの声援

- Rindaは永続化されてないから使えないよね
- SPOF ww
- TupleSpaceが永続化されたらウハウハですよ
- これで分散ハッシュ書けるからやってよ

そうは思わないけど断れない

- 実装して評価してもらおう

そうは思わないけど断れない

- 実装して評価してもらおう
  - 実装した

そうは思わないけど断れない

- 実装して評価してもらおう
  - 実装した
    - RubyKaigiで言いふらした

そうは思わないけど断れない

- 実装して評価してもらおう
  - 実装した
    - RubyKaigiで言いふらした
      - だれも使わない...ちっ

そうは思わないけど断れない

- 実装して評価してもらおう
  - 実装した
    - RubyKaigiで言いふらした
      - だれも使わない...ちっ
        - 自分で評価するか... ◀ いまここ

PTupleSpaceの反省

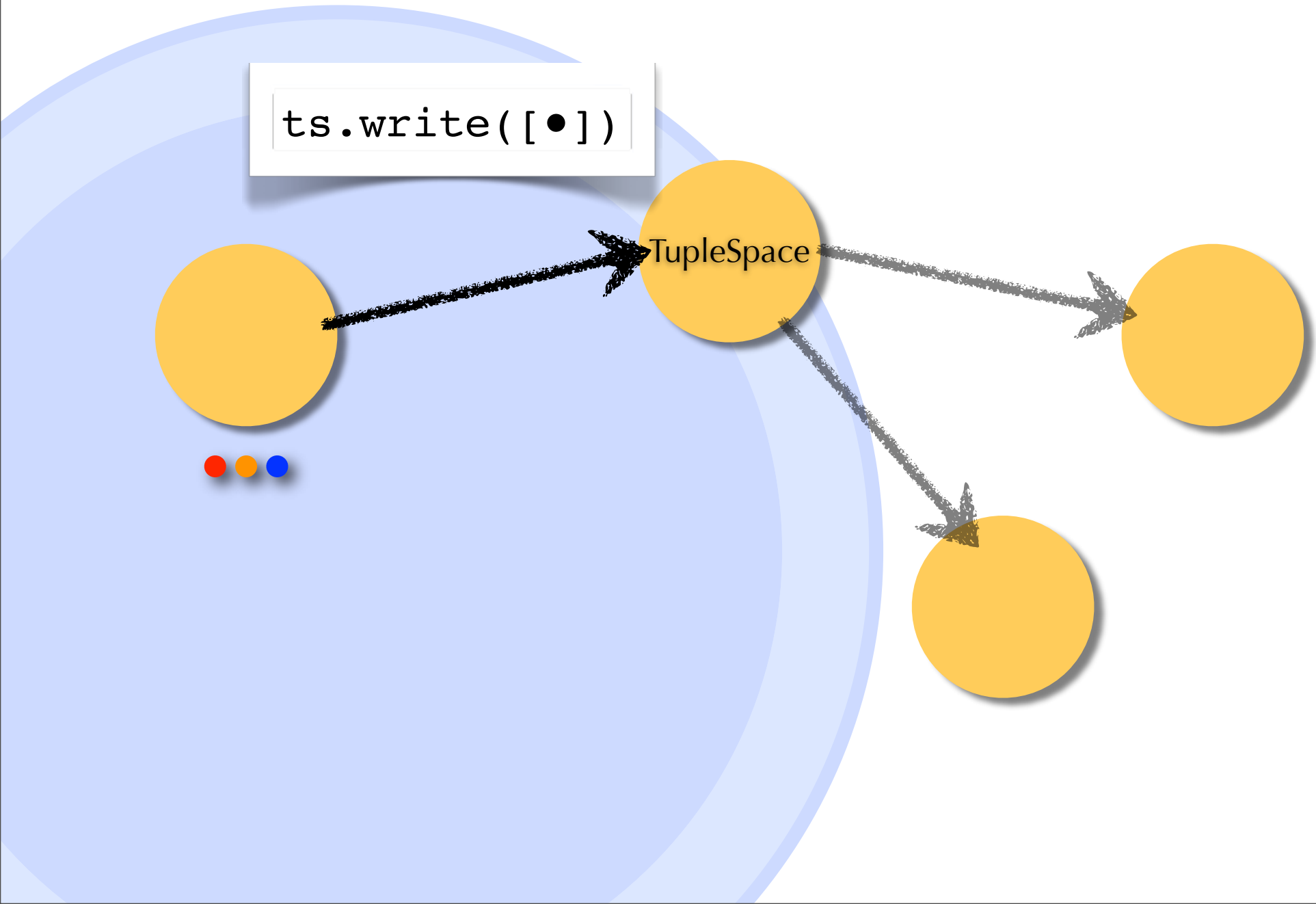
PTupleSpace

- PTupleSpaceてなに
  - LindaとRinda
  - MoreRinda
- PTupleSpaceはどうなの？
  - 協調
  - ストレージ

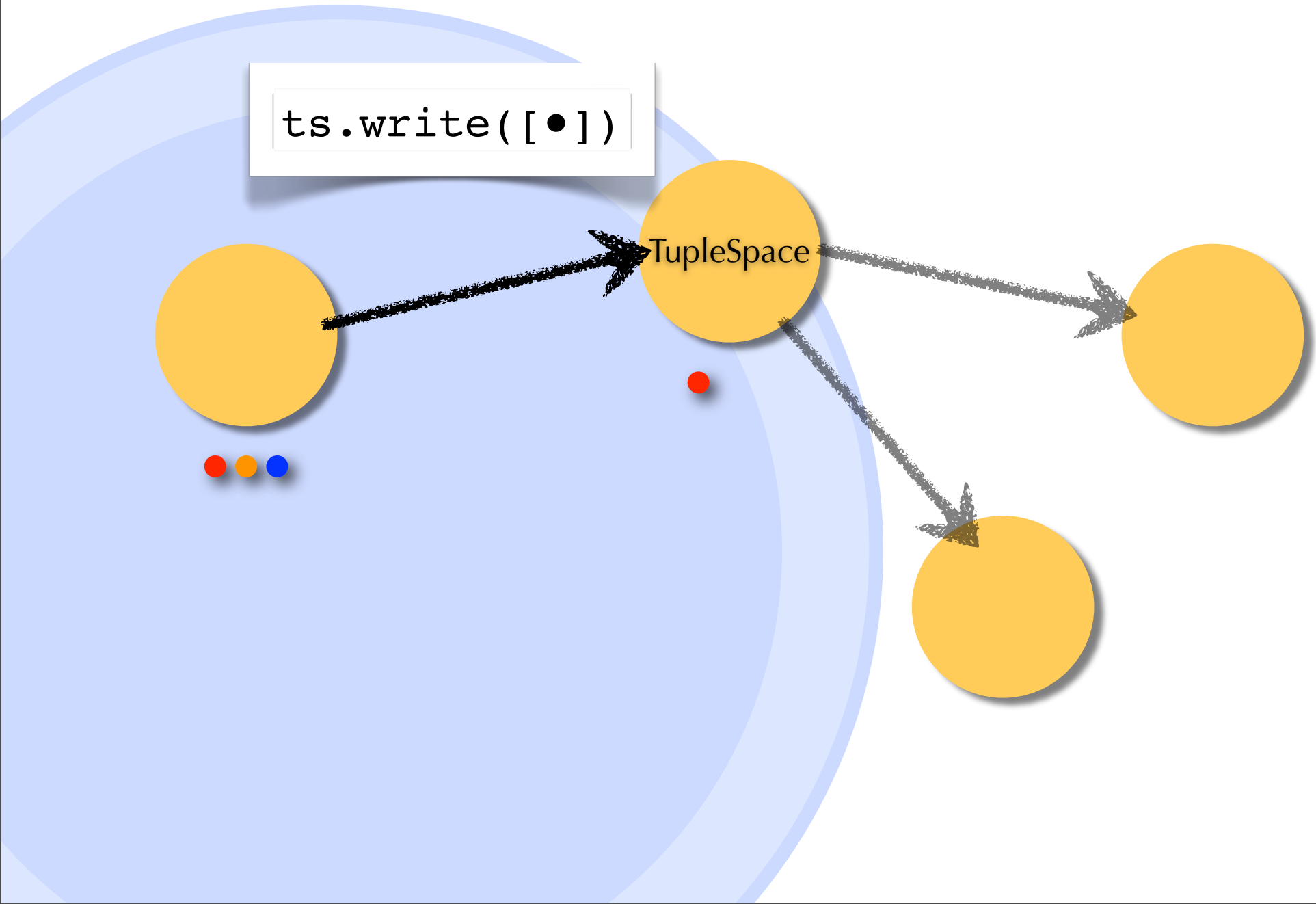
並列処理糊言語Linda

- タプルは複数のオブジェクトの組
- 「タプルの交換」を使ってプロセス群が協調する仕組み
- タプルを置いたり読んだり取り出す場がタプルスペース
  - 基本となる三つの操作

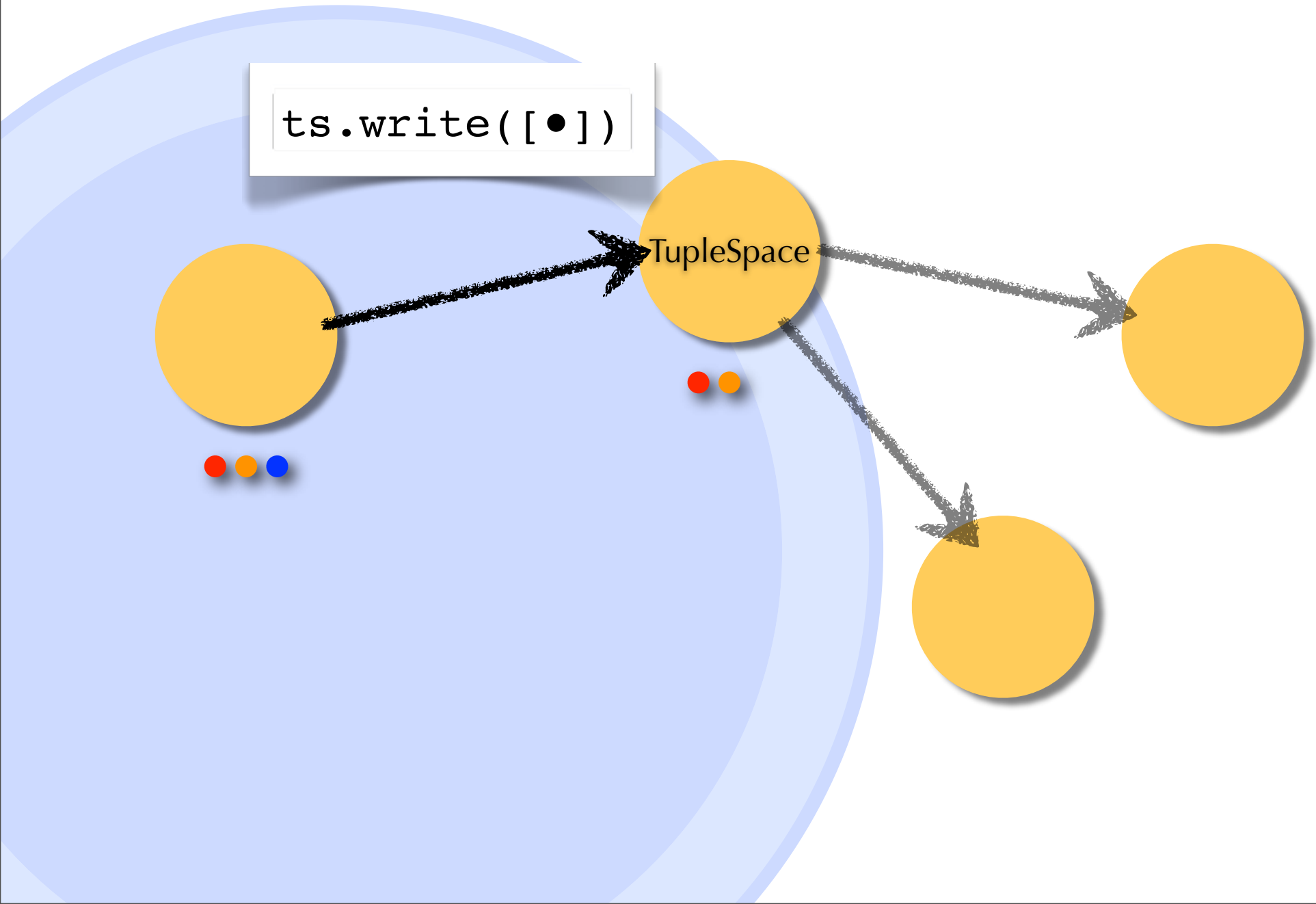
write



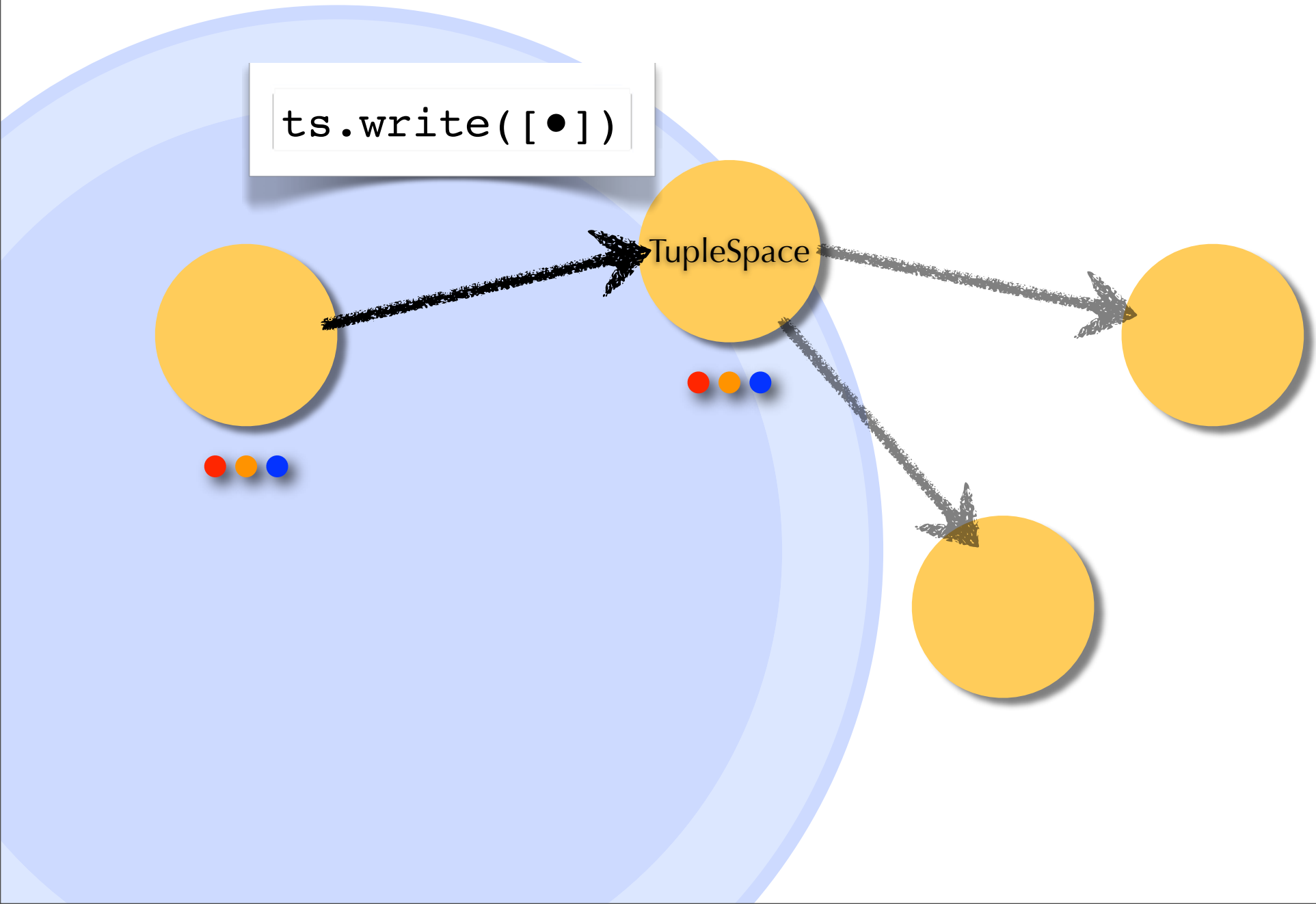
write



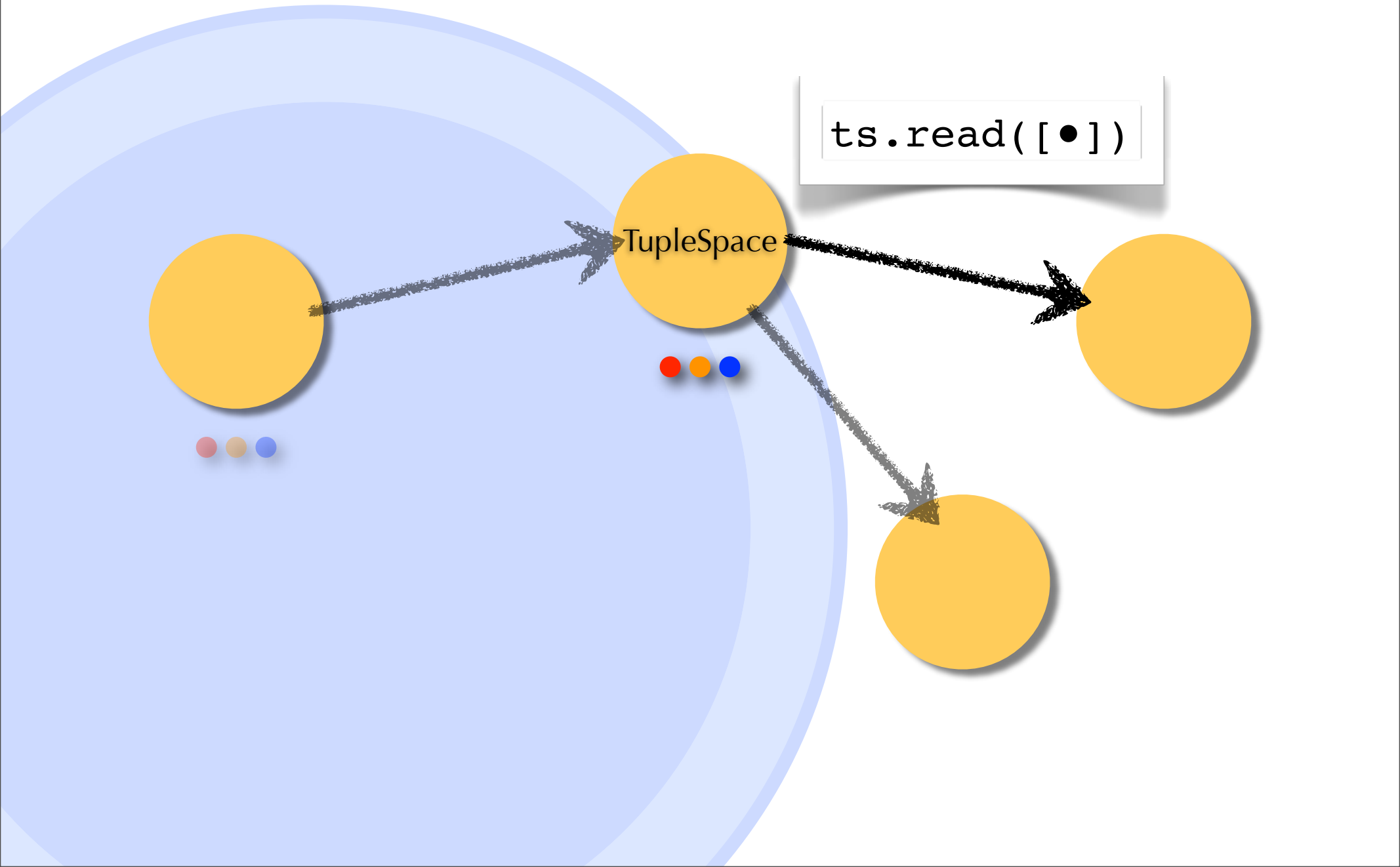
write



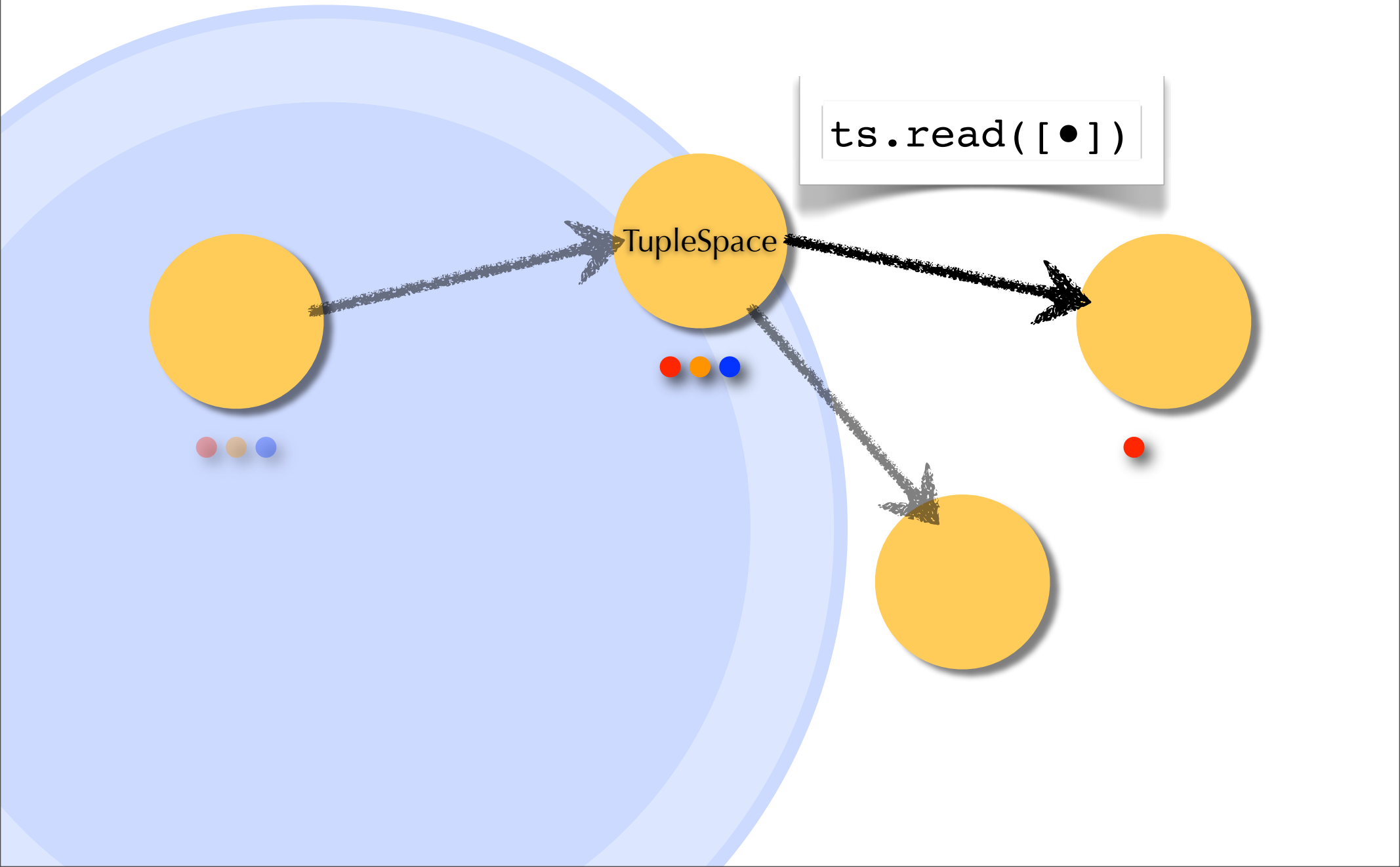
write



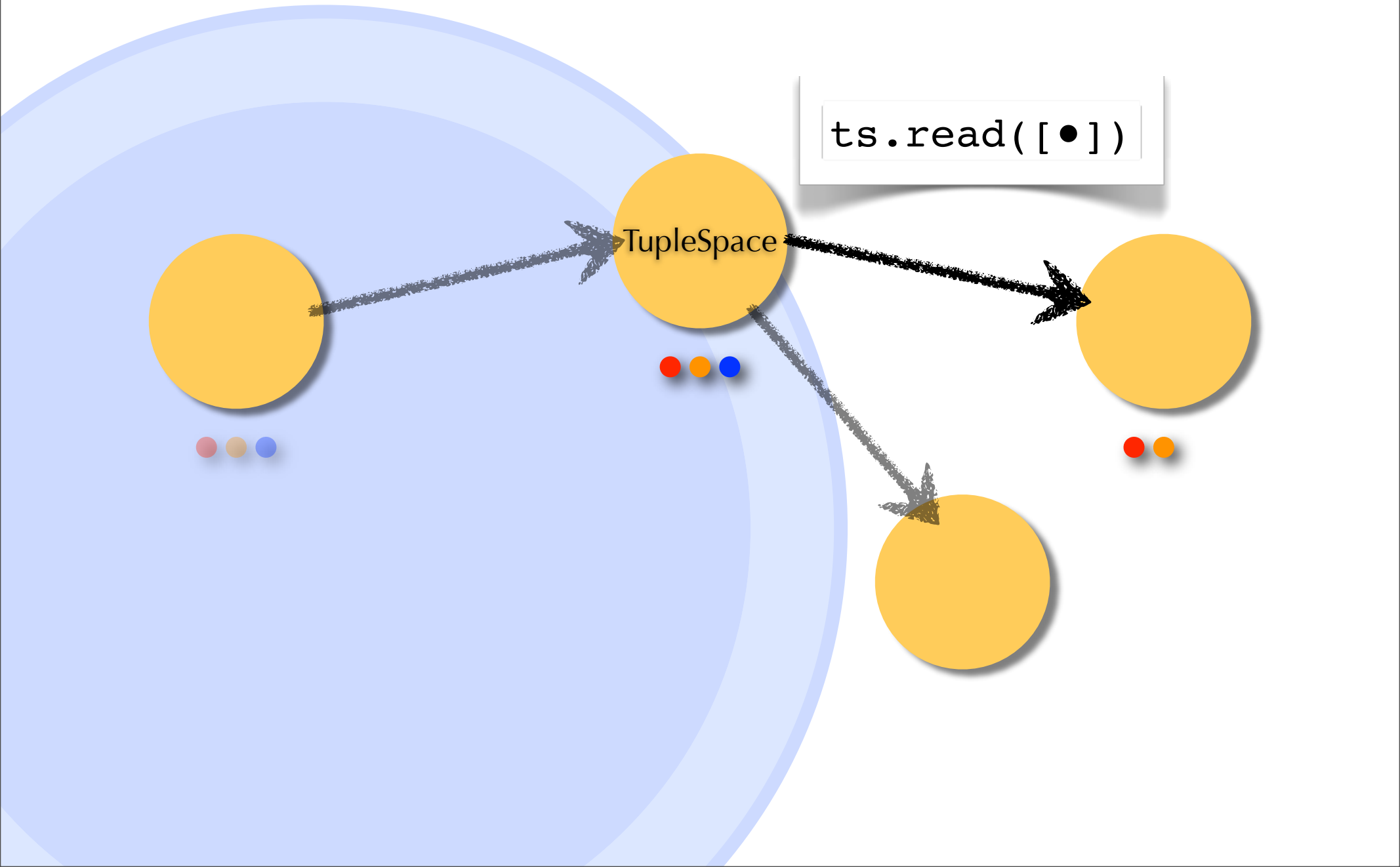
read



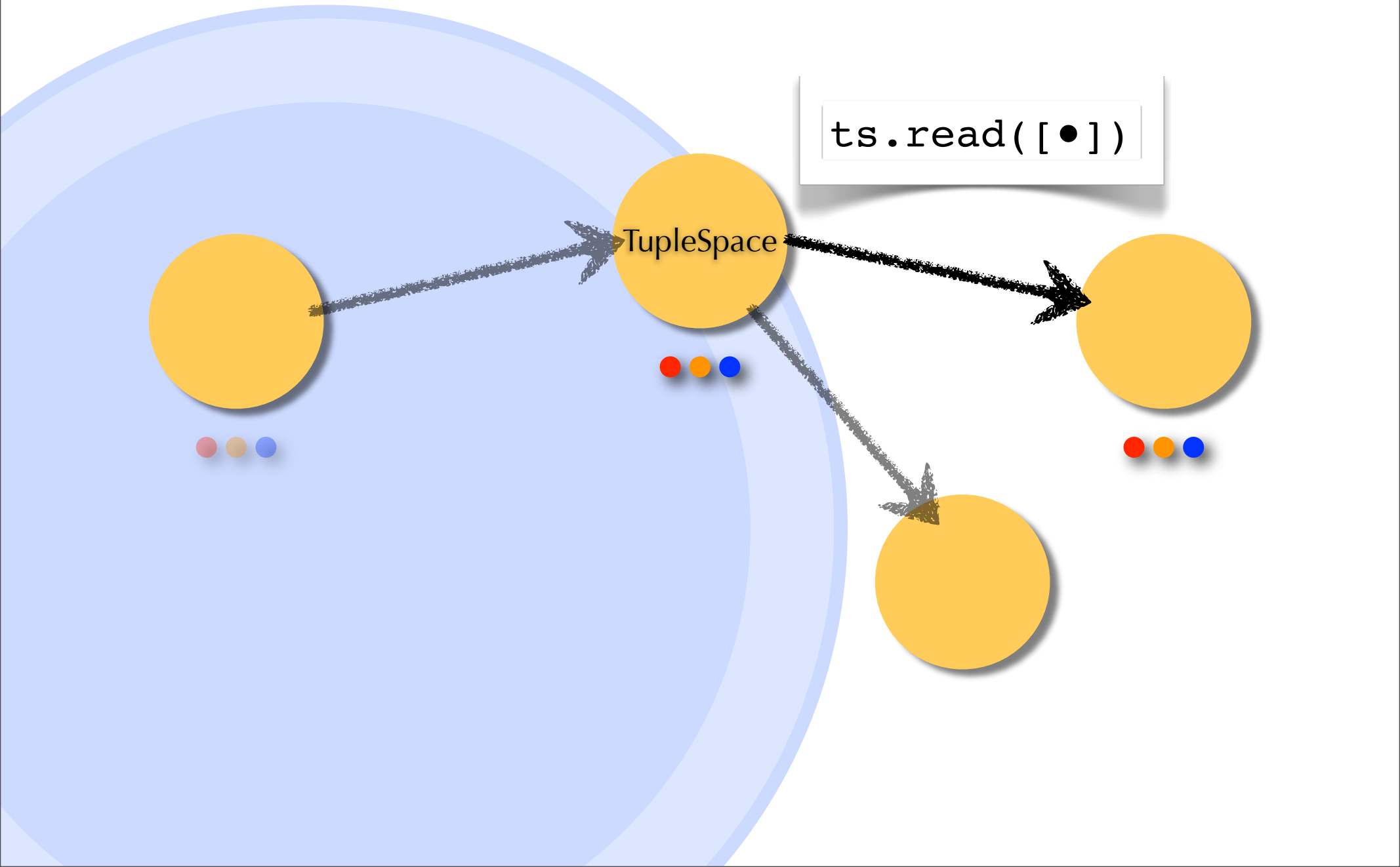
read



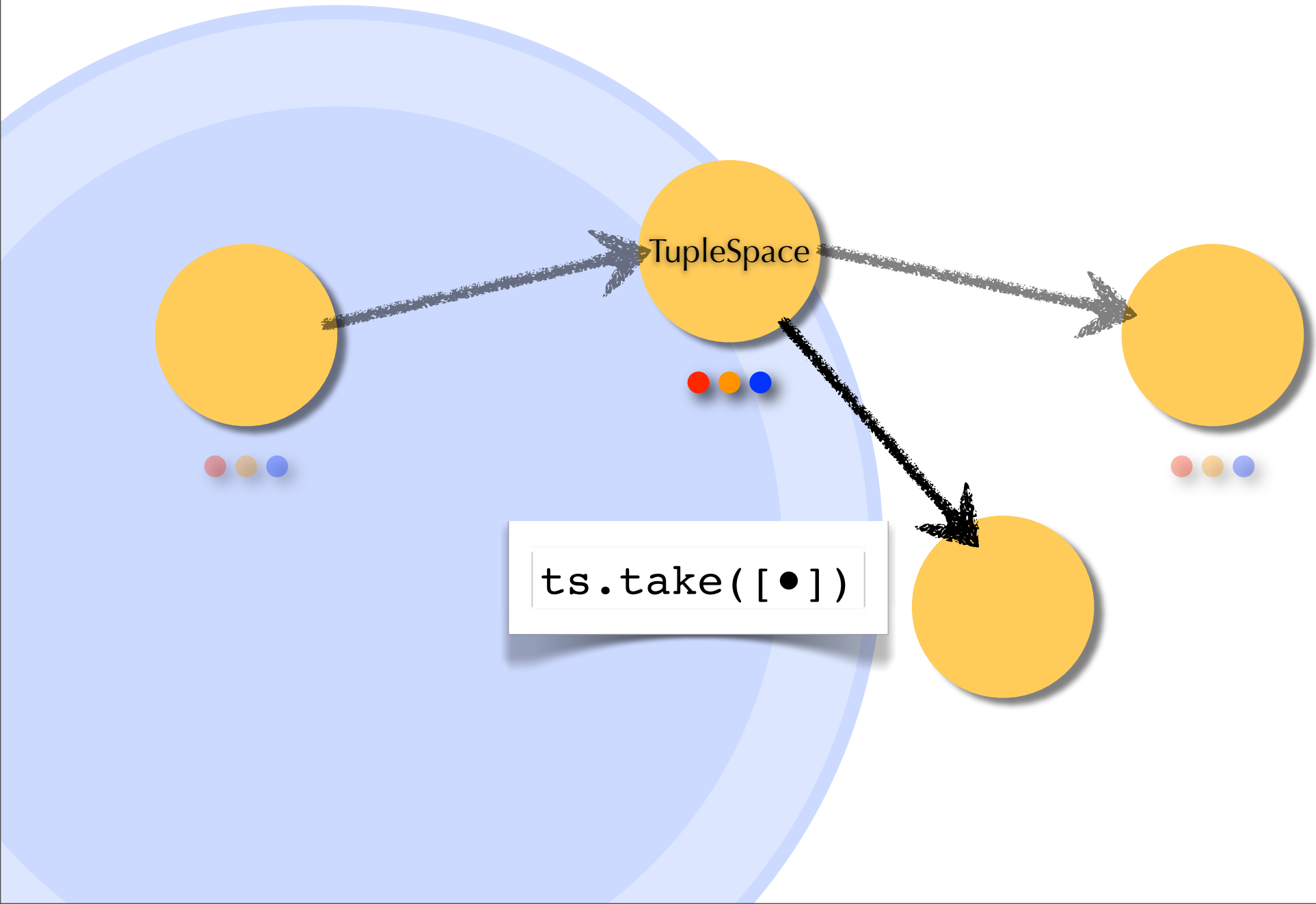
read



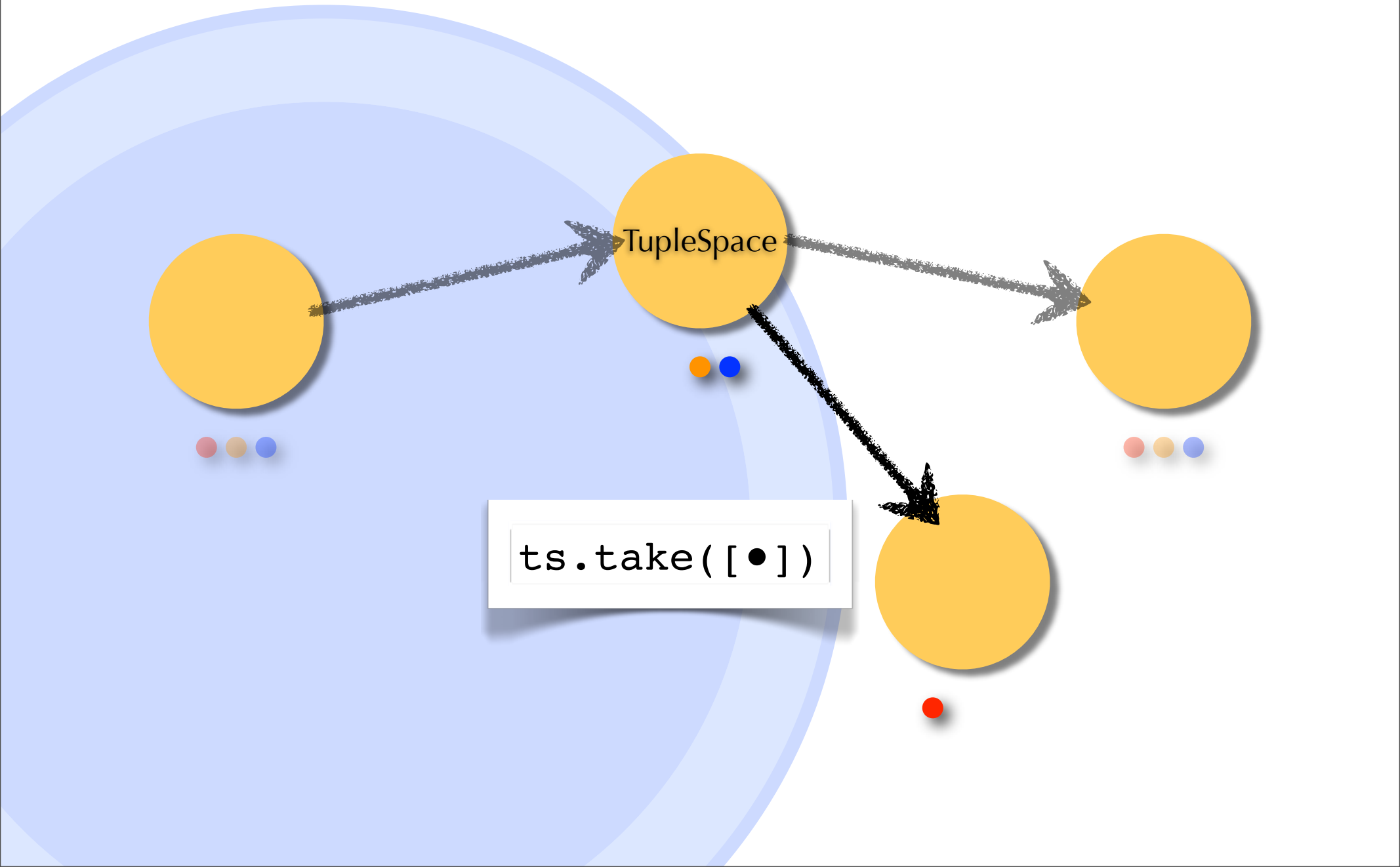
read



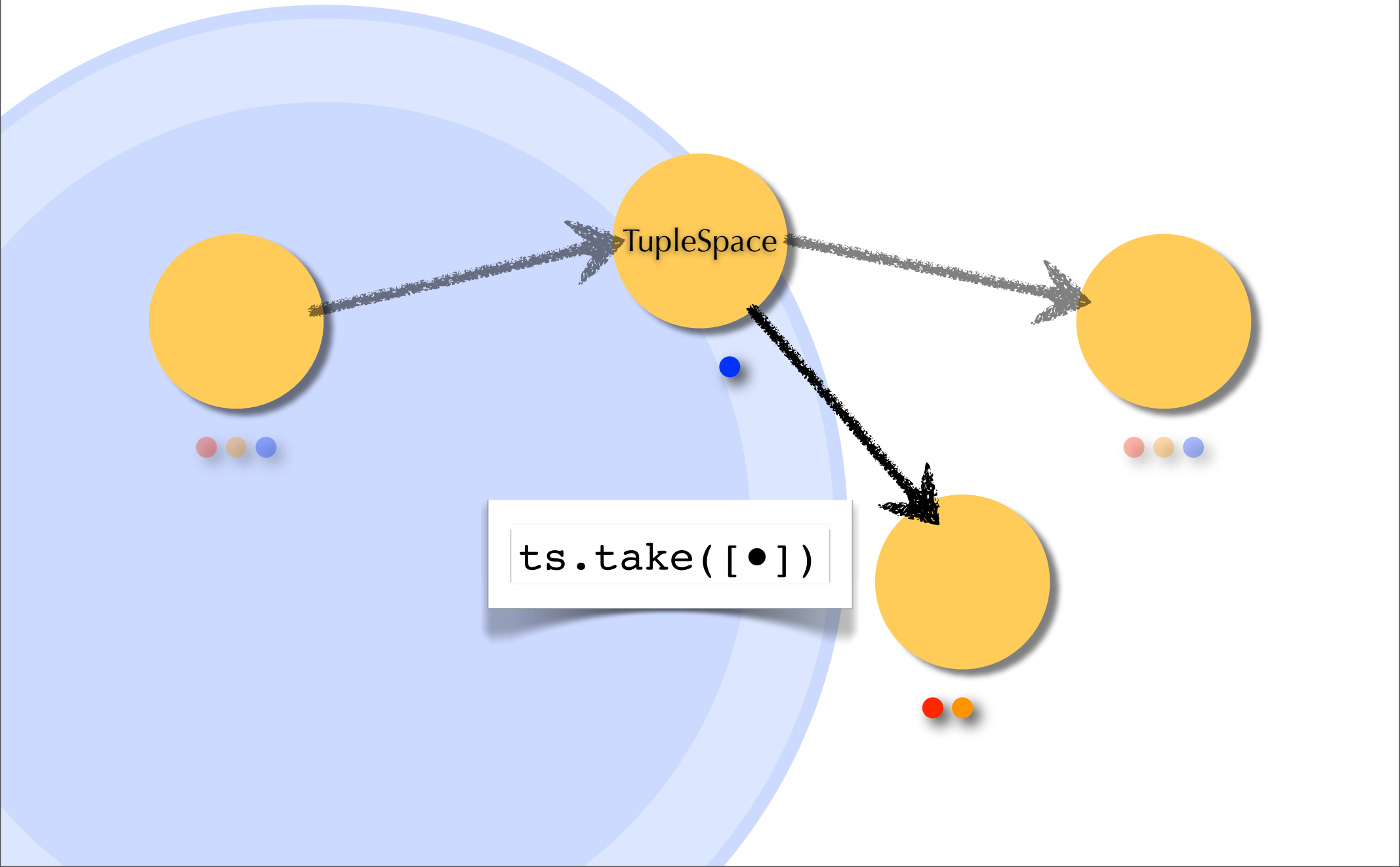
take



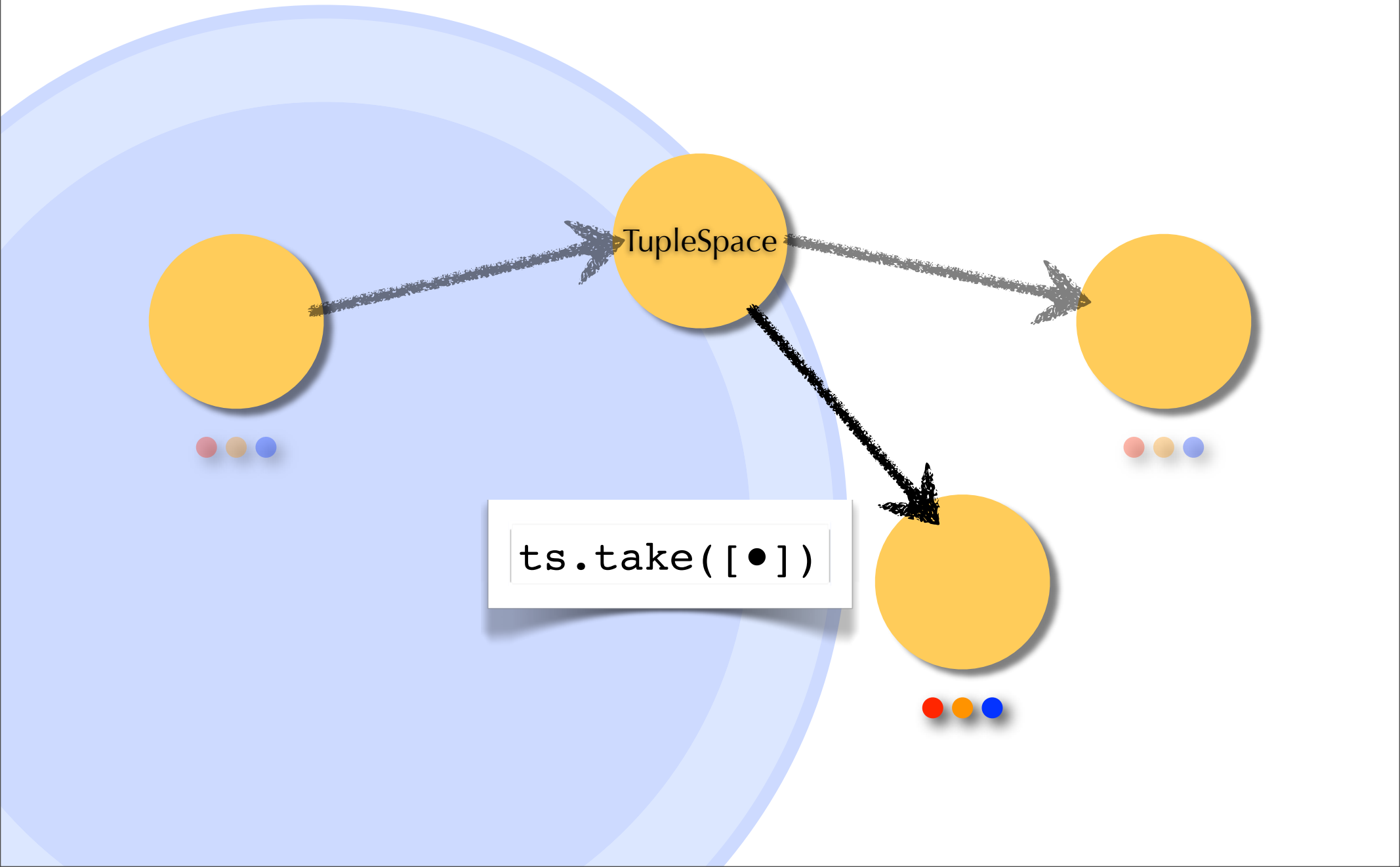
take



take



take



Rinda

- 並列処理糊言語LindaのRuby版
- Rubyで二番目に人気のあるライブラリ
  - 一番目はdRuby

PTupleSpace

- タプルを復元できるTupleSpace
  - P is for Persistent
- PTupleSpace = Rinda::TupleSpace + 永続化
- RubyKaigi2007で発表

MoreRinda

- Rindaにさらにかっこいい機能を追加する
  - PTupleSpace
  - rinda_eval - LindaのevalをRubyで実現
- github.com/seki/MoreRinda

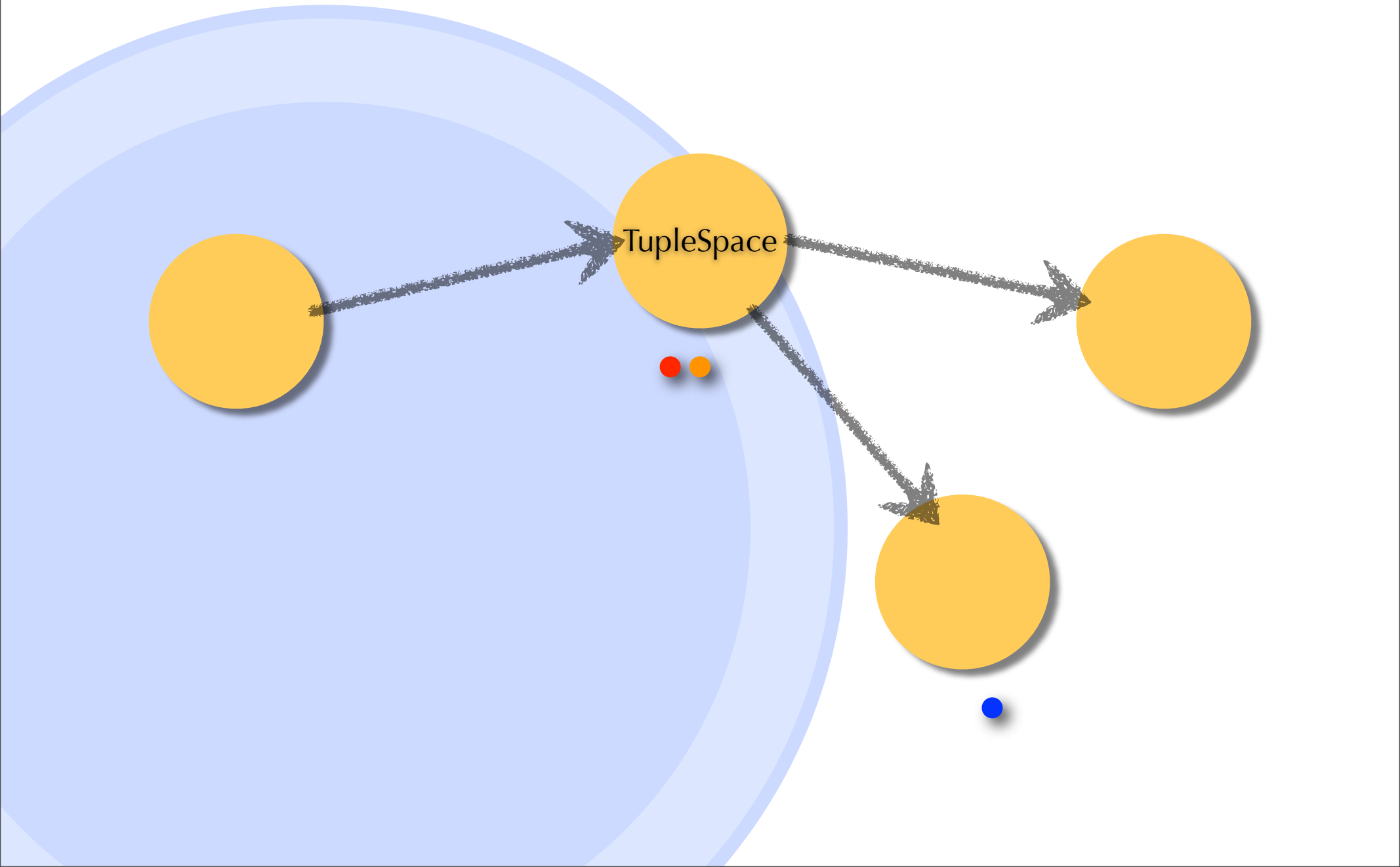
永続化できたら...

- 障害がおきても協調が再現できるかも
- ストレージのように使えるかも

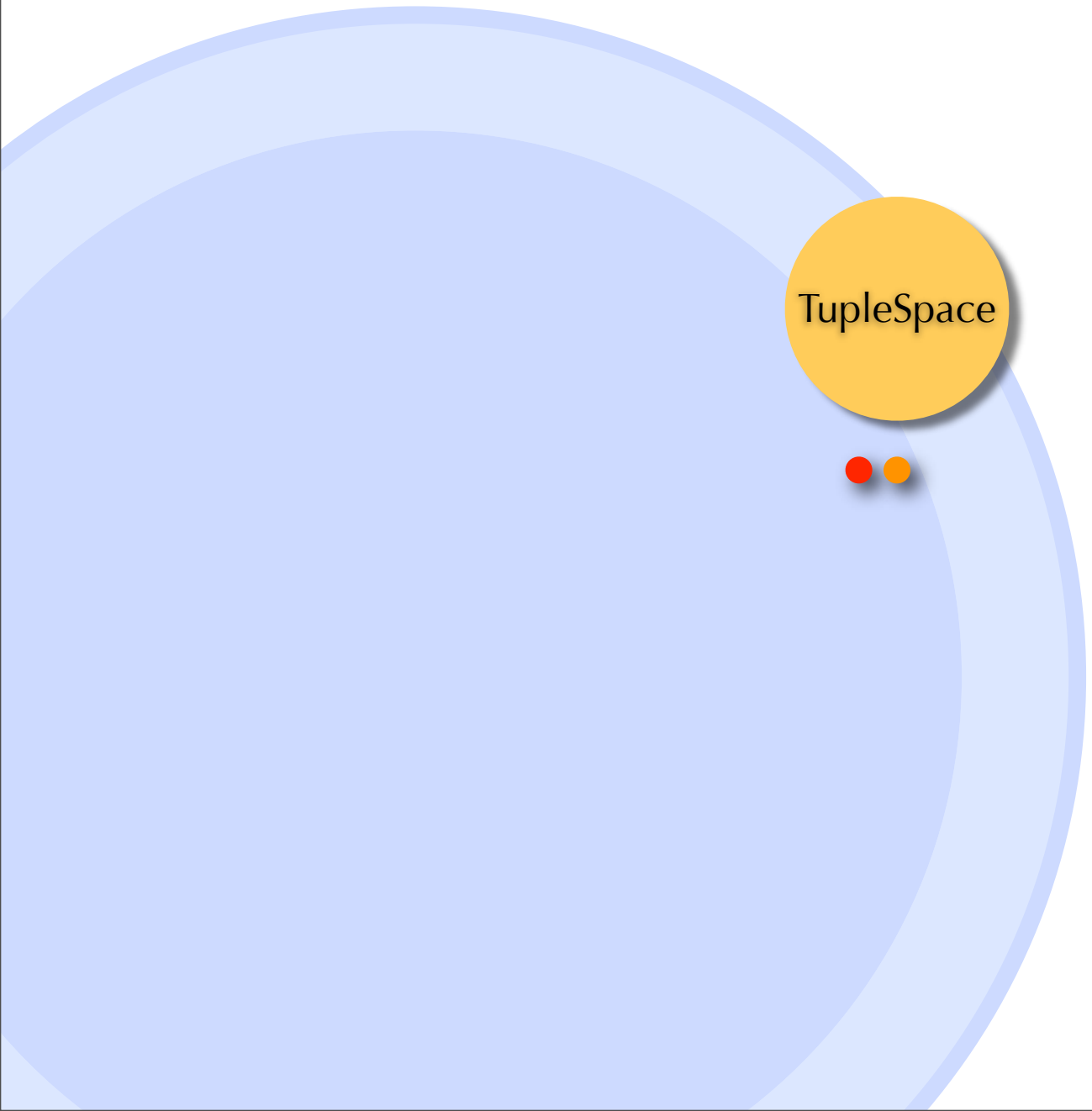
PTupleSpaceの制約

- TupleSpace内のタプルの状態だけを復元
  - 外にあるタプルは復元できない
- takeの途中でクラッシュするとタプルを失う

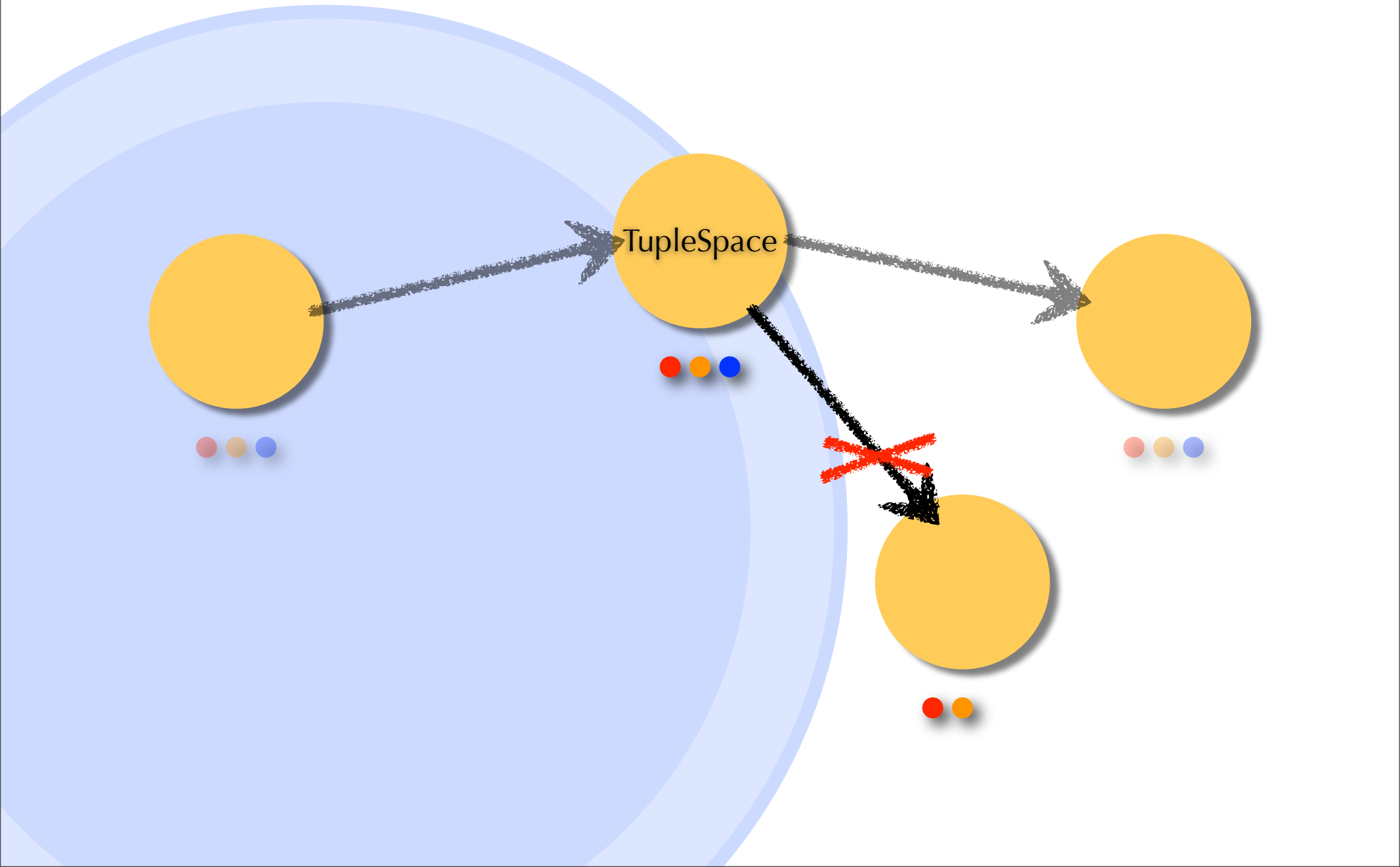
TupleSpaceの中だけ



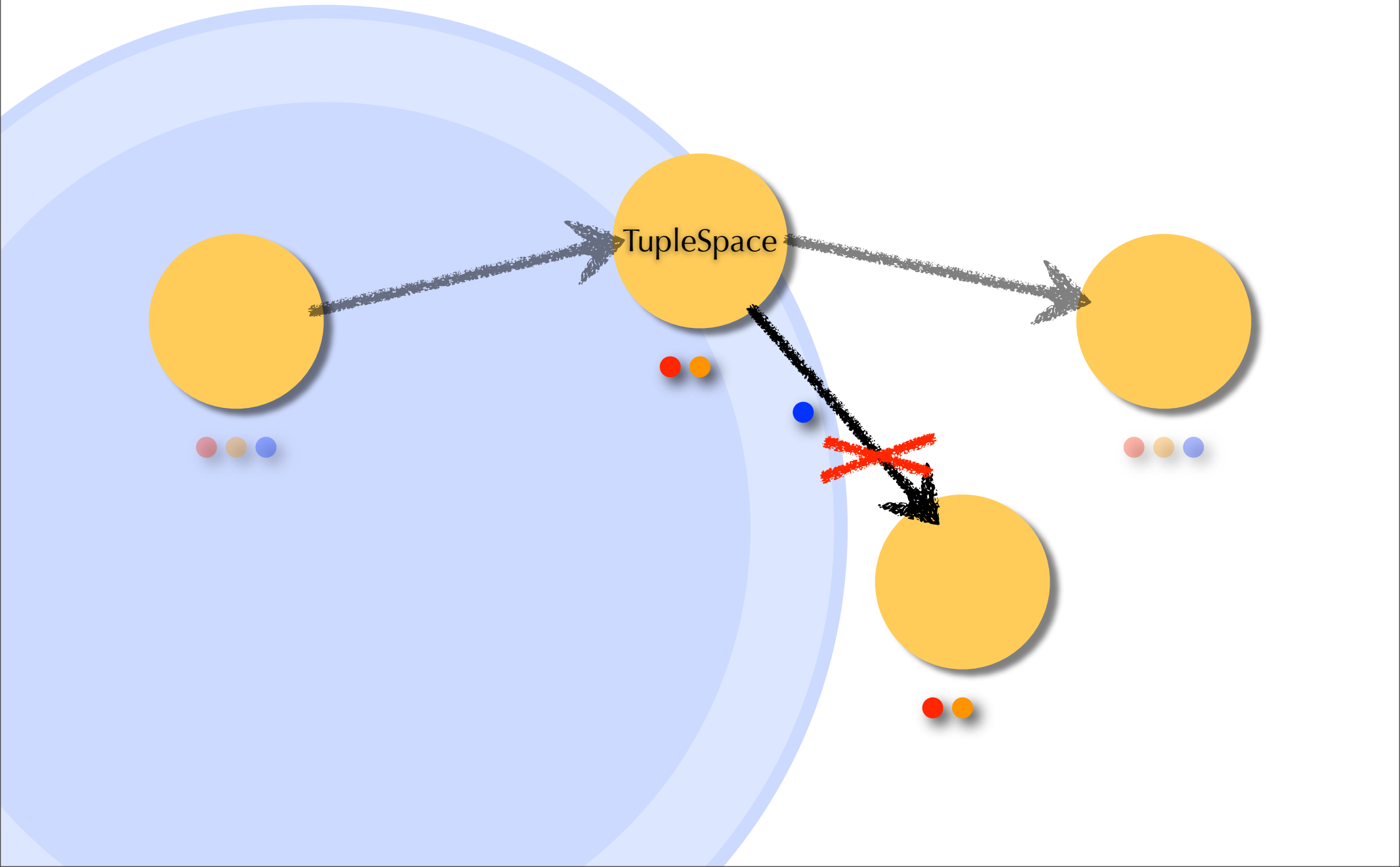
TupleSpaceの中だけ



takeでのクラッシュ



takeでのクラッシュ



takeでのクラッシュを考える

- クラッシュが削除する前か後かわからない
  - リトライすべきか判断できない
  - 協調の様子を復元するのは難しい

協調の点ではちょっと問題

- takeの問題があるので使えないケースが多い
  - タプルの紛失が問題にならなければ...

ストレージとしてはどうか

- 永続化されるならストレージにしたいよ
- KVSとしてTupleSpaceを使ったらいいじゃん
  - という提案をしばしばいただく...

KVSとしてはどうか

- TupleSpaceは辞書でなくバッグ
  - 同じ情報の重複を許す
  - 用途に合っていれば手軽に使える
    - 用途は多くないと思う
    - せめて順序があれば...

Hashを模倣するには

- 一つのキーに一つの値
- 重複させないために全体のロックが必要
  - ロック用タプルをtake
  - 旧い値をtake
  - 新しい値をwrite
  - ロック用タプルをwrite

ストレージとしてどうか

- Hashになるのは効率悪い
- Bagならわりといける

PTupleSpaceのまとめ

- そんなに甘くなかった!
- 単純なTupleSpaceの永続化では並列処理の糊としてイケてない
- ほしいストレージがBagなら使える
  - Hashの代わりにはなるのは難しい

しばらくお待ちください

- ✓ +08:00
- ✓ 日本先行発売
- ✓ PTupleSpaceは本当に存在したことをアピール
- ✓ TupleSpaceの永続化では足りないよ
- ✓ 次が本編 / Drip

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

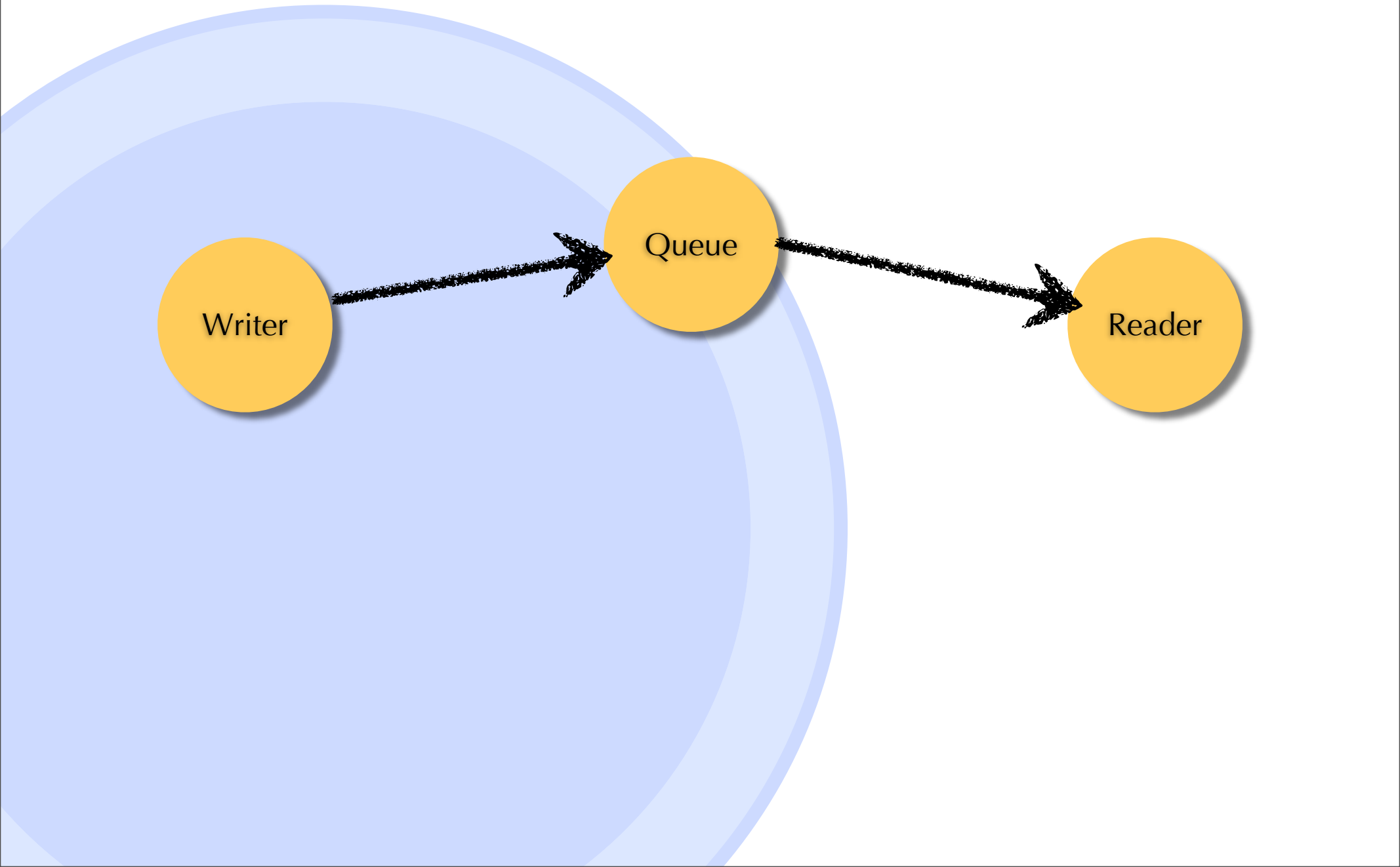
身近な例で説明

- Queueとの比較 / 協調
- Hashとの比較 / ストレージ

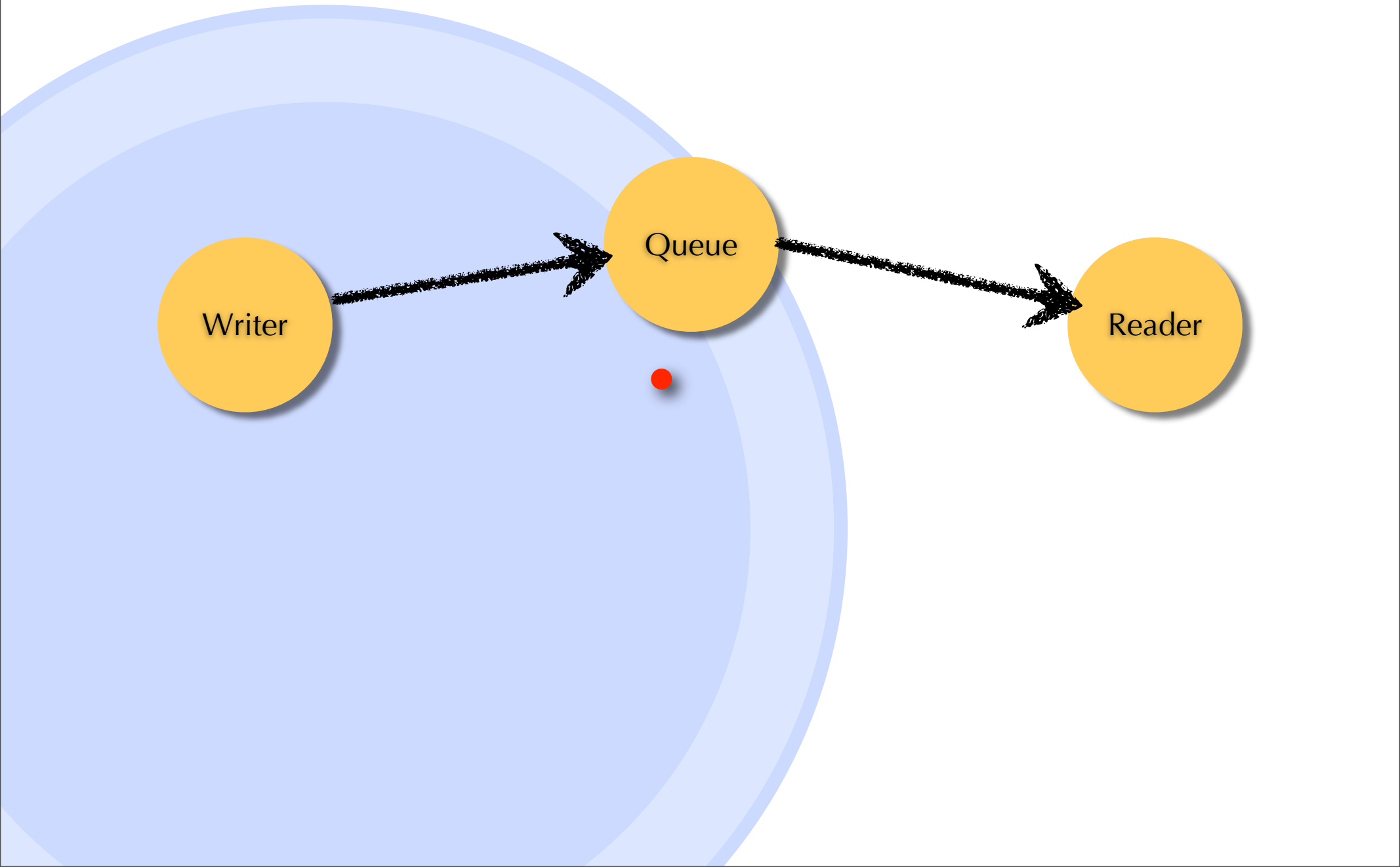
Queueとの比較

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

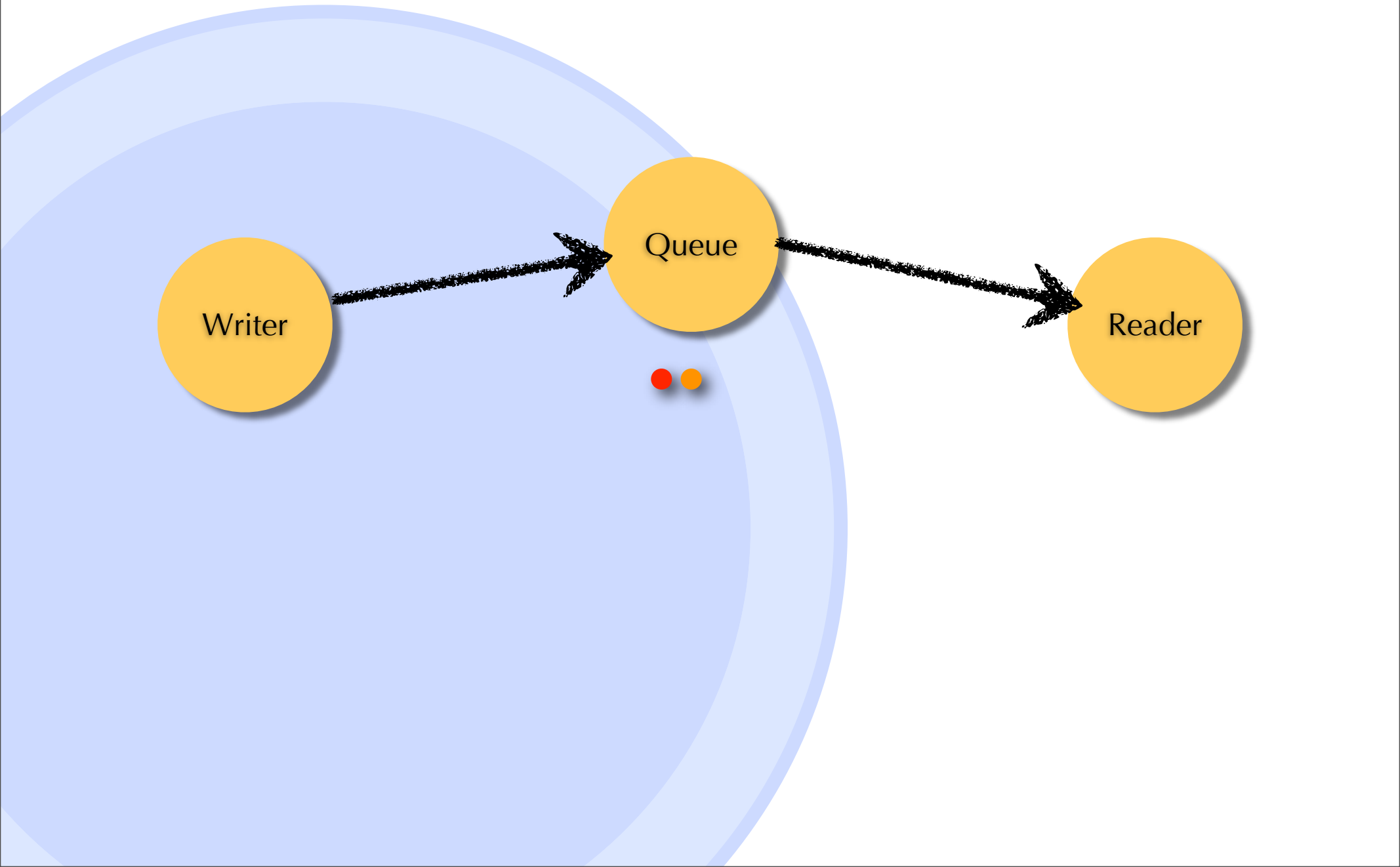
Queueだとこんな感じ



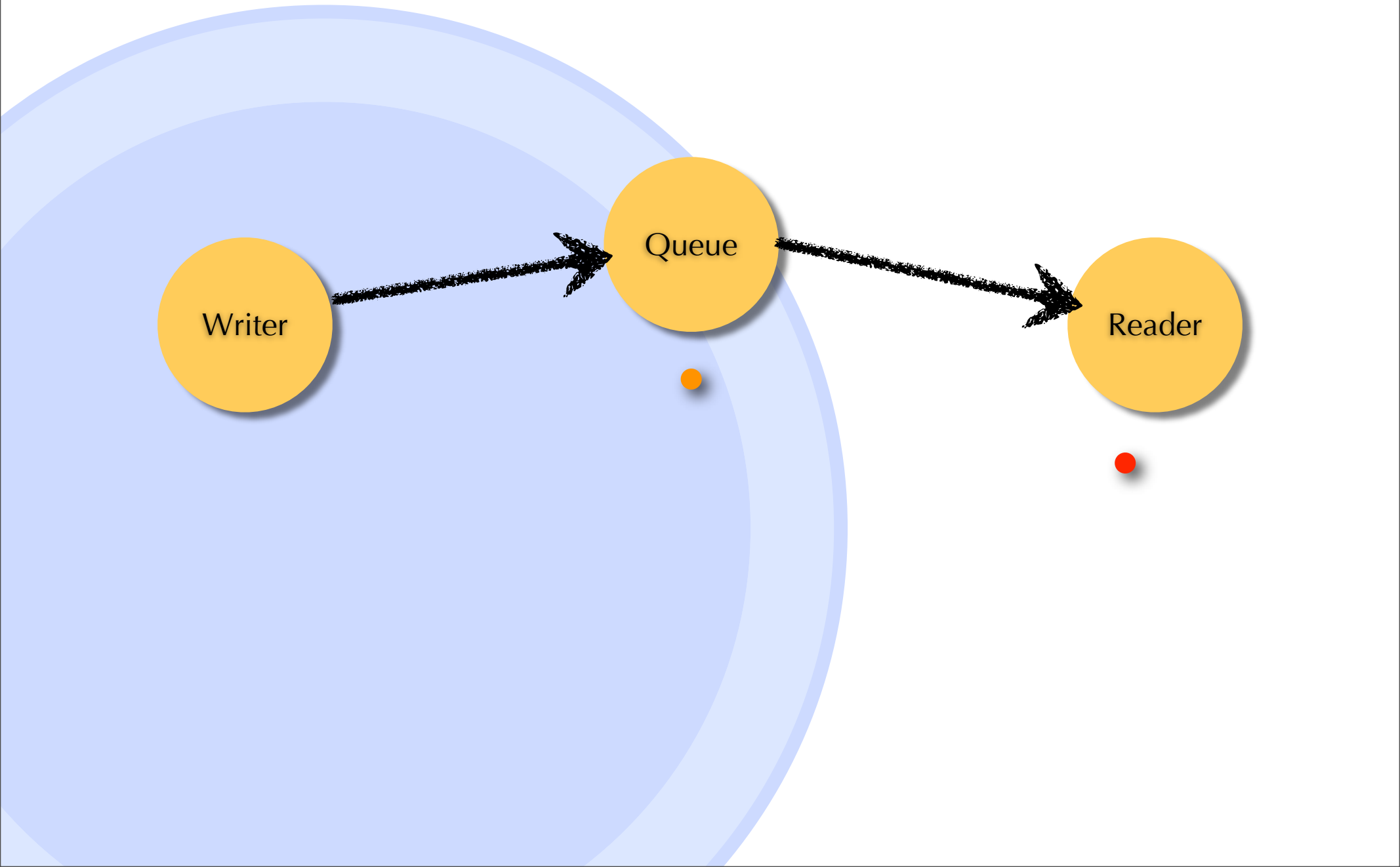
Queueだとこんな感じ



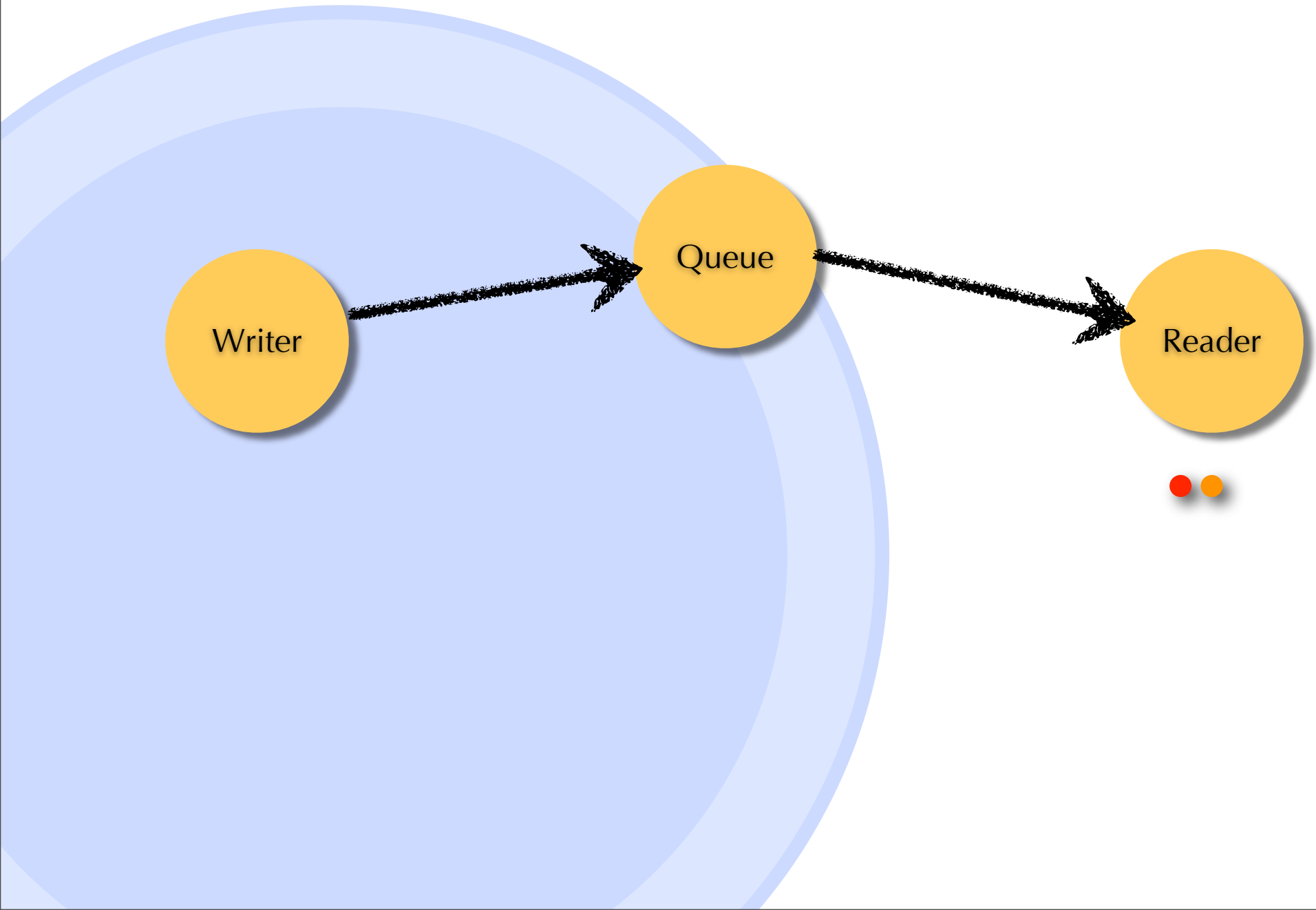
Queueだとこんな感じ



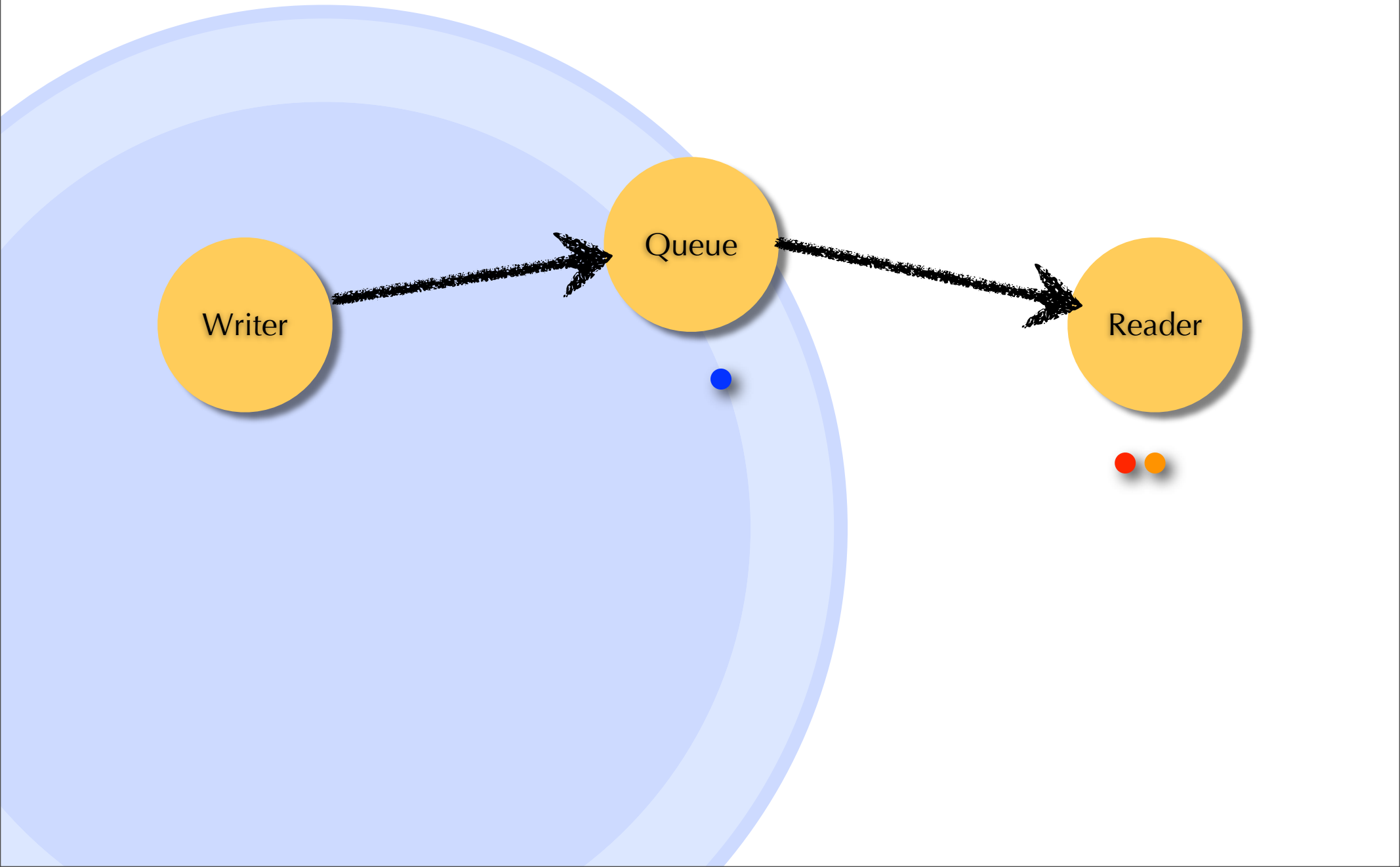
Queueだとこんな感じ



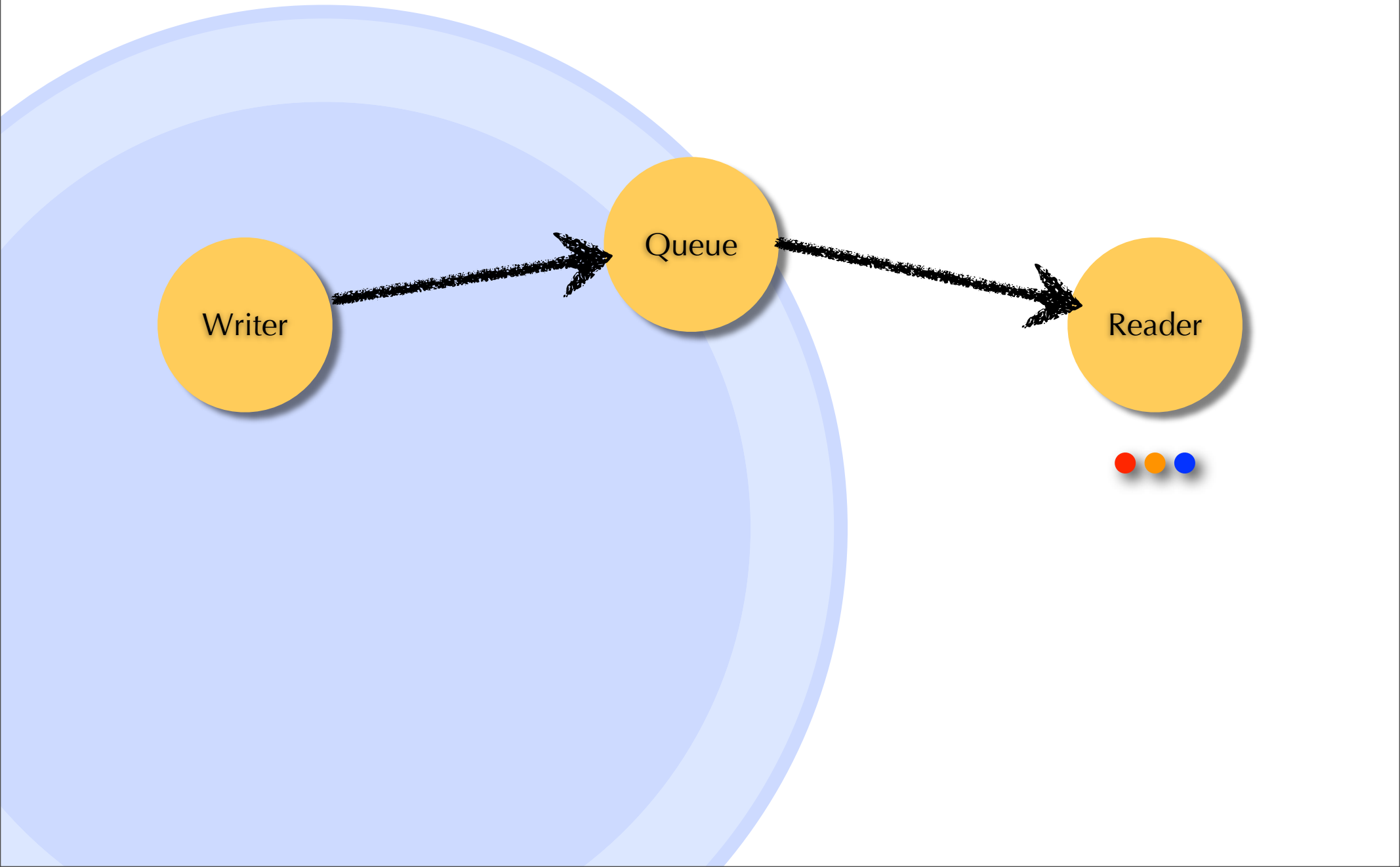
Queueだとこんな感じ



Queueだとこんな感じ



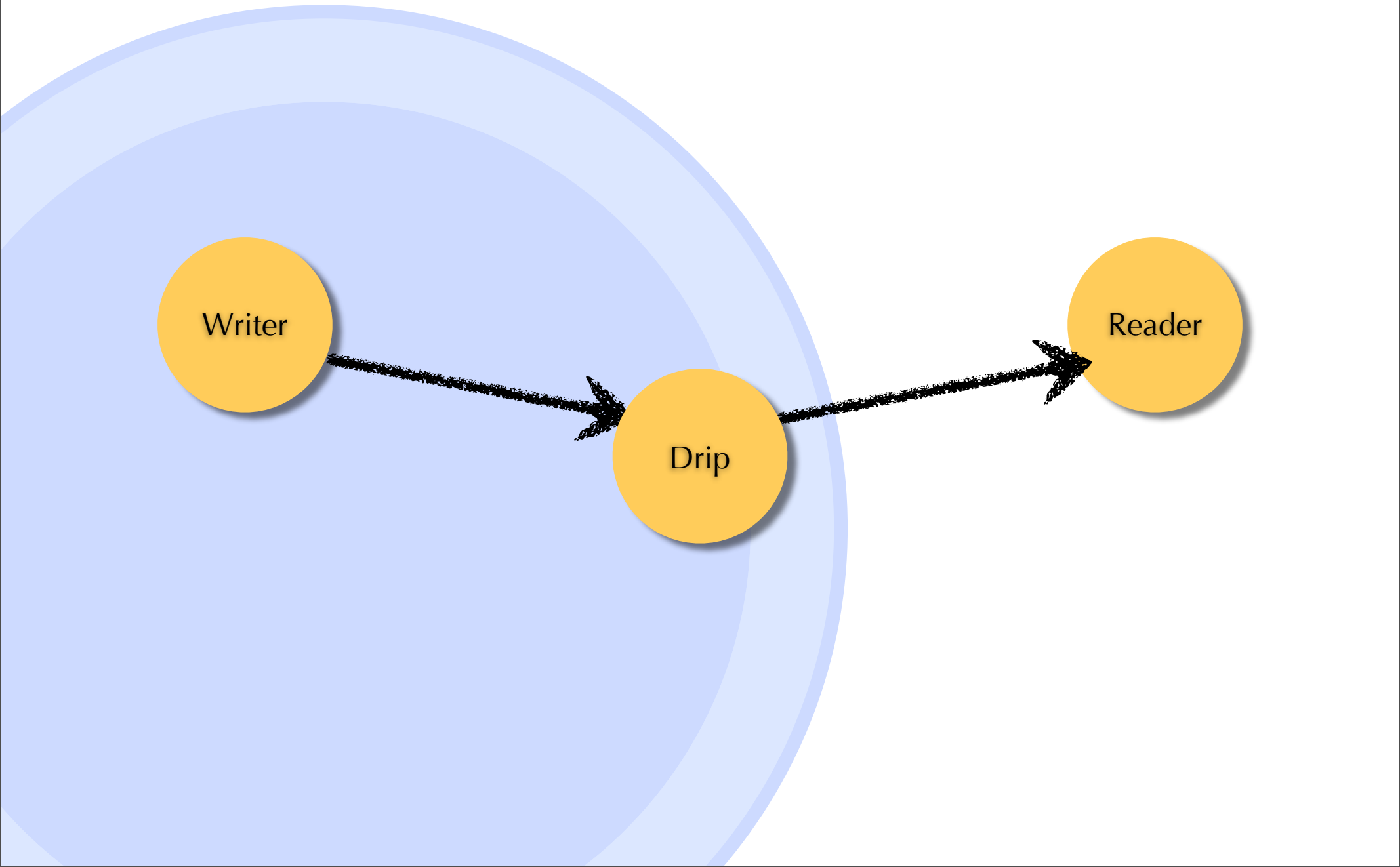
Queueだとこんな感じ



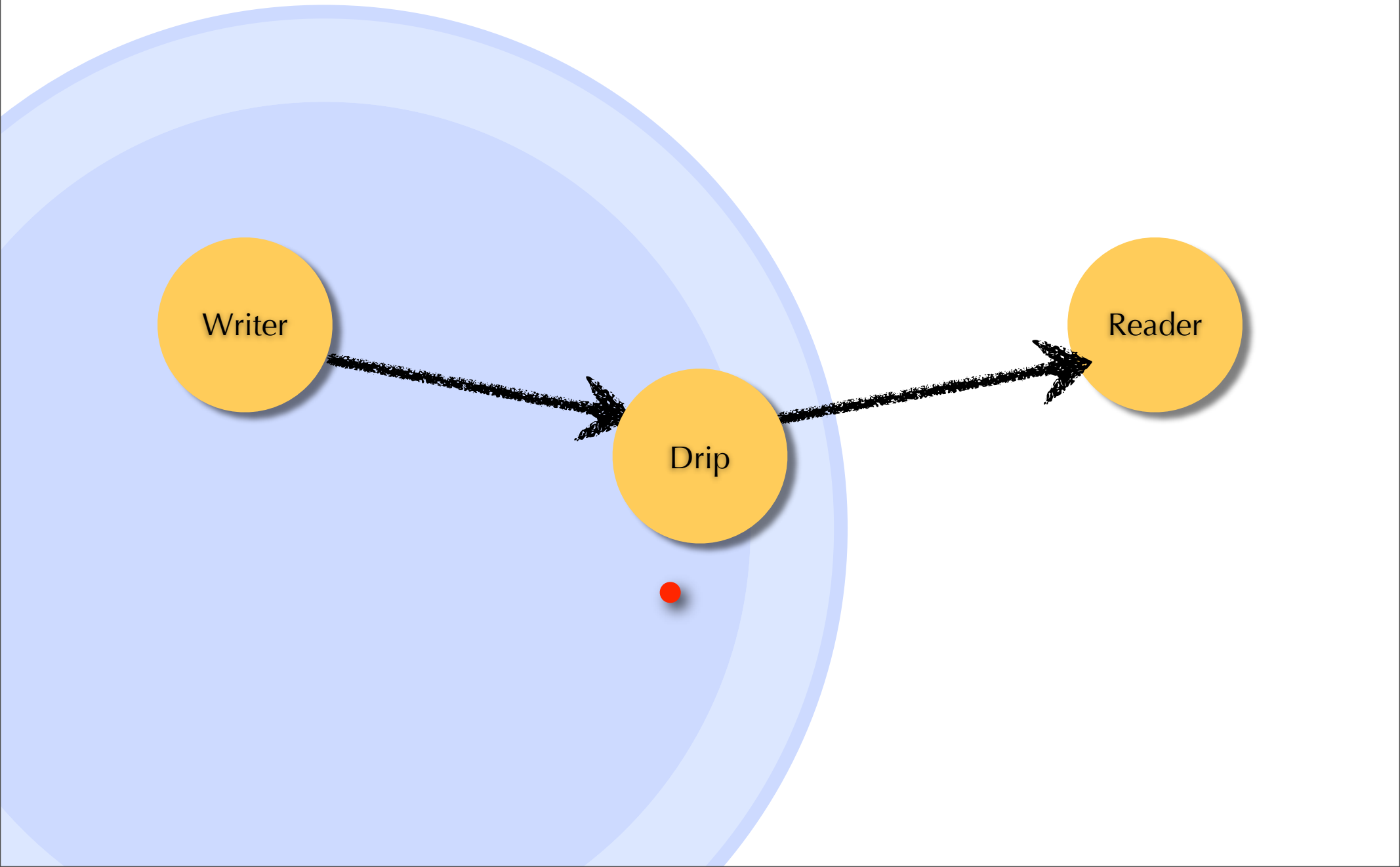
Queue#pop

- 要素は消費される

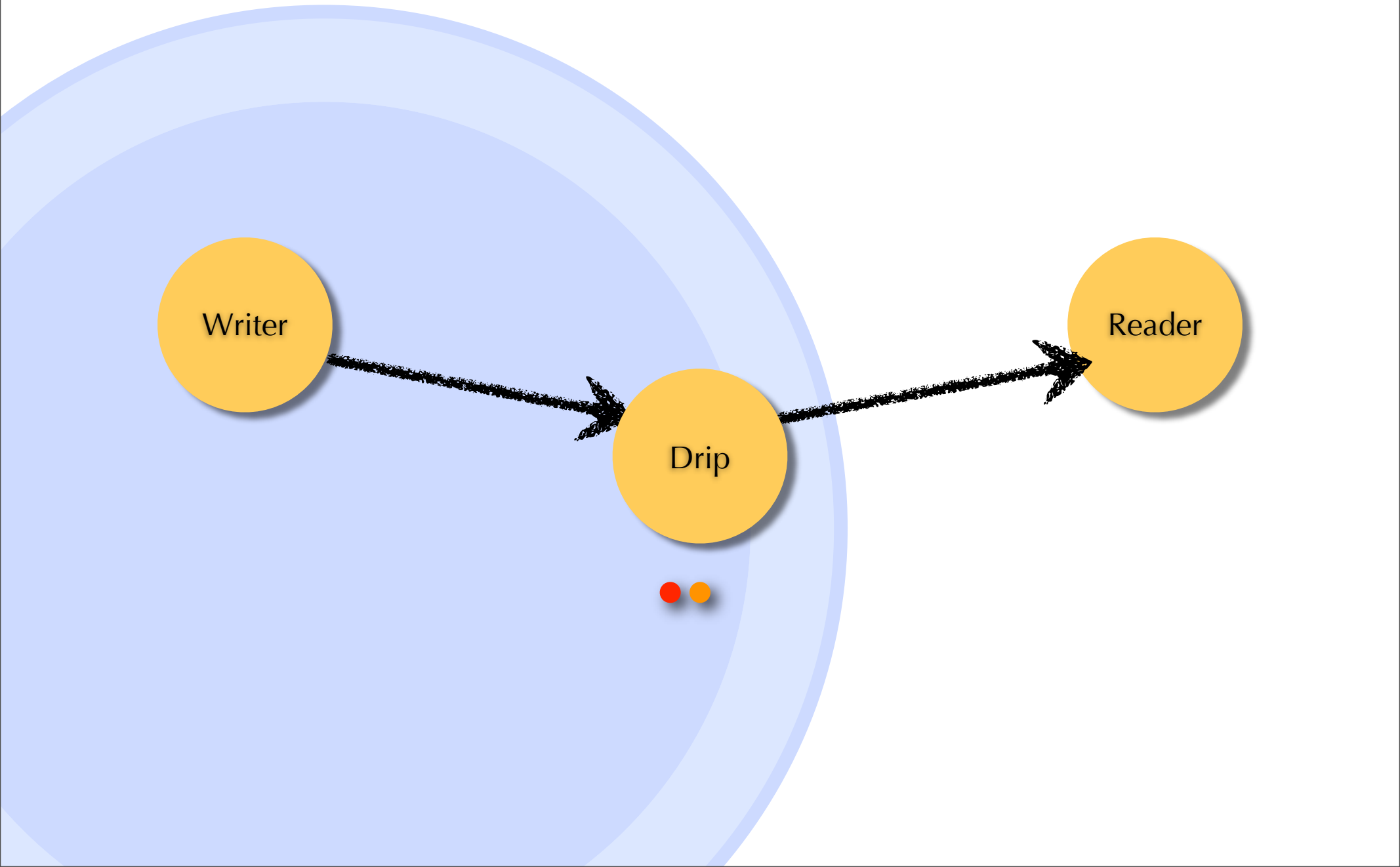
Dripだとこんな感じ



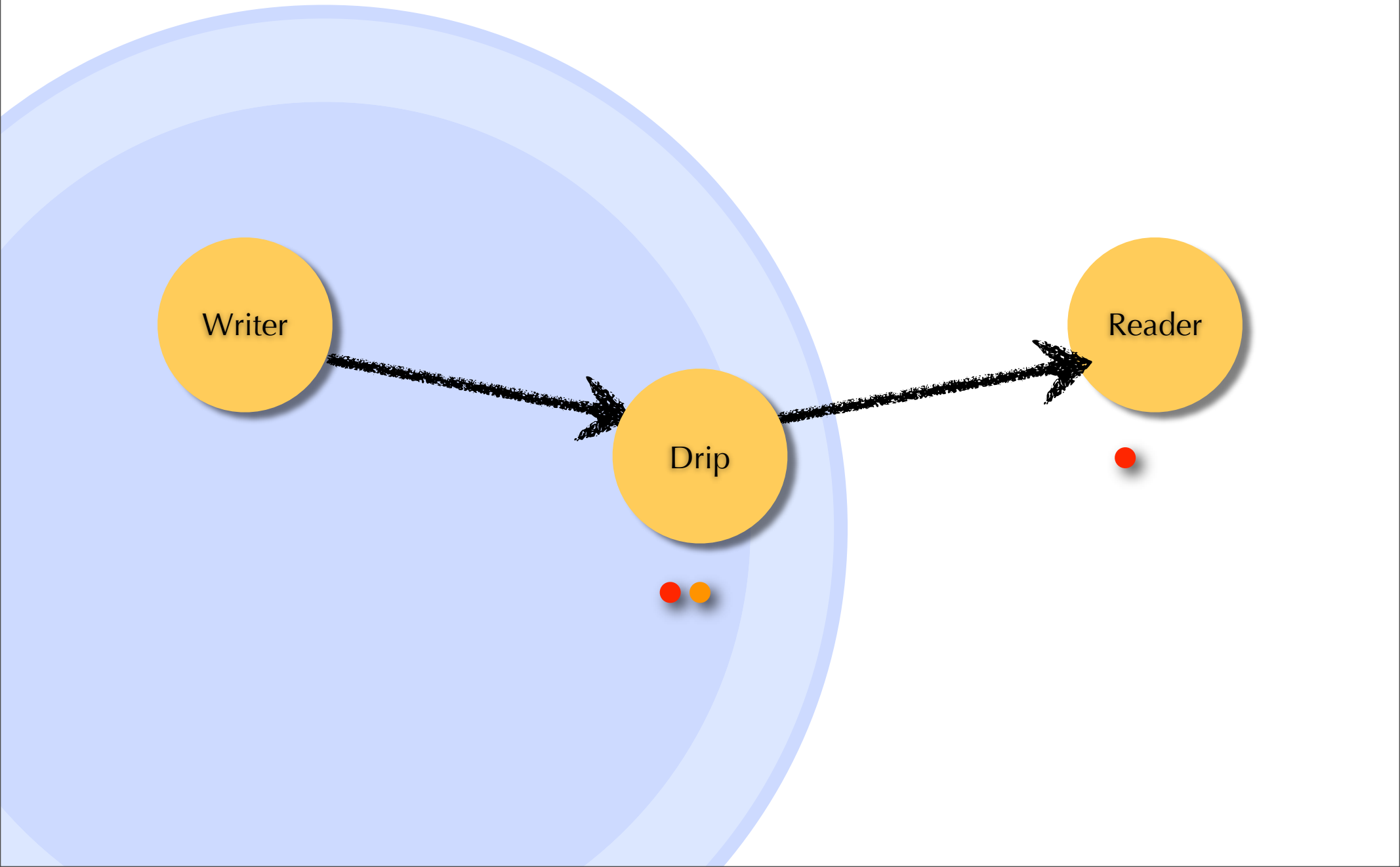
Dripだとこんな感じ



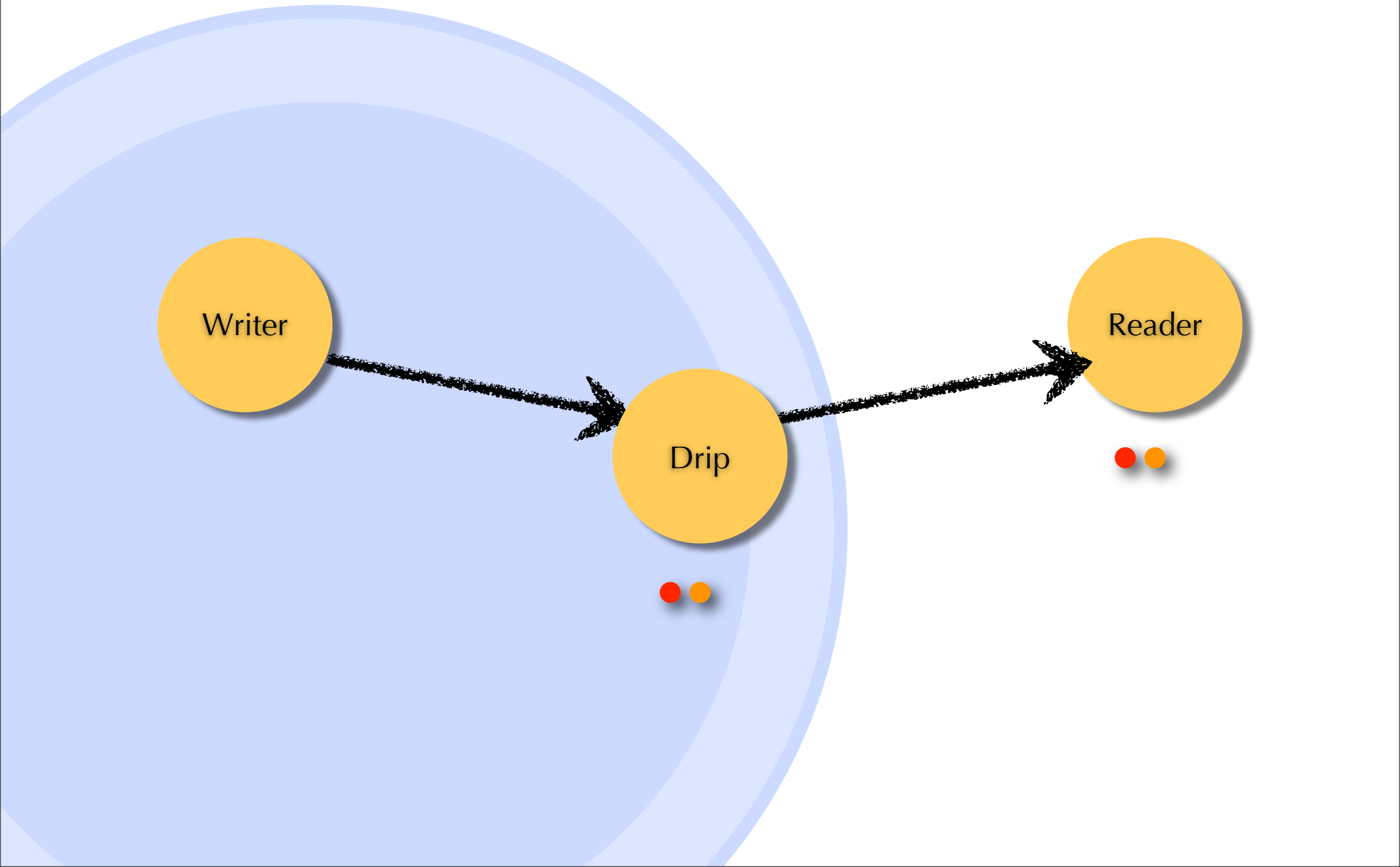
Dripだとこんな感じ



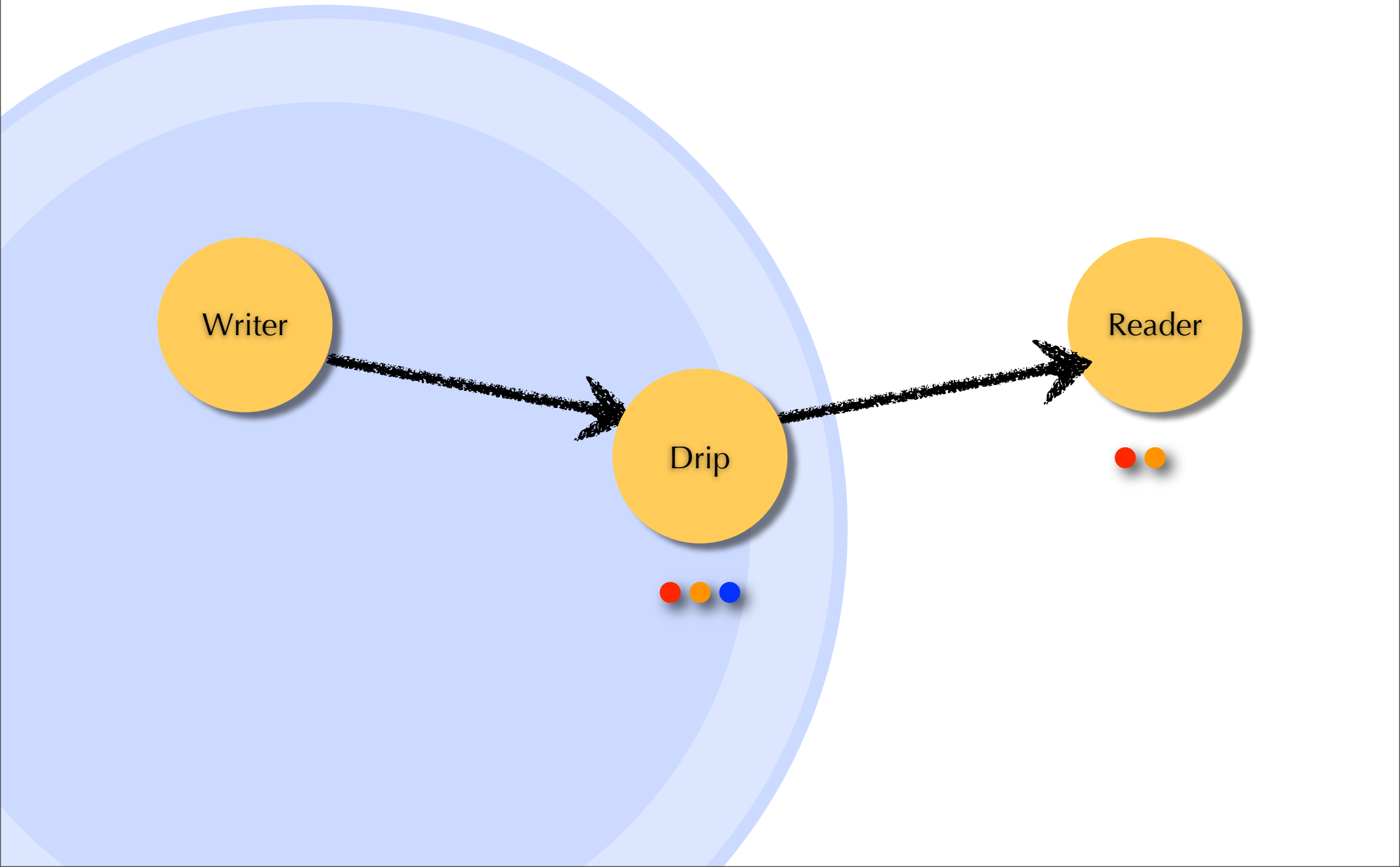
Dripだとこんな感じ



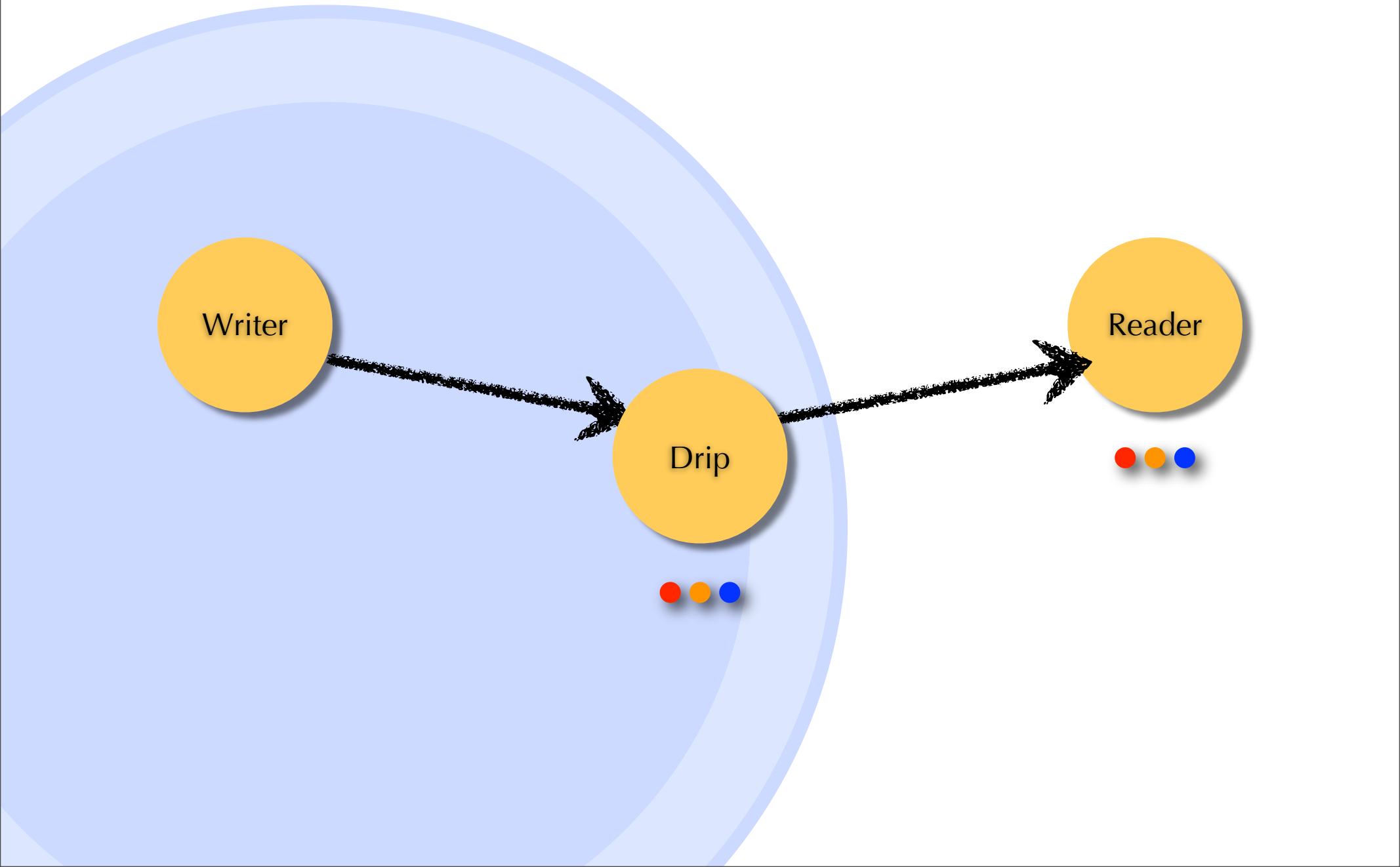
Dripだとこんな感じ



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'

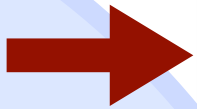
reader

```
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)

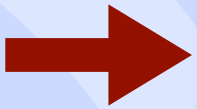
```
drip.read(0, 1)[0]
# [2, 'hello']
```



key	value
2	'hello'
3	'world'

read(2, 1)

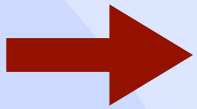
```
drip.read(2, 1)[0]
# [3, 'world']
```



key	value
2	'hello'
3	'world'

waiting...

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



key	value
2	'hello'
3	'world'

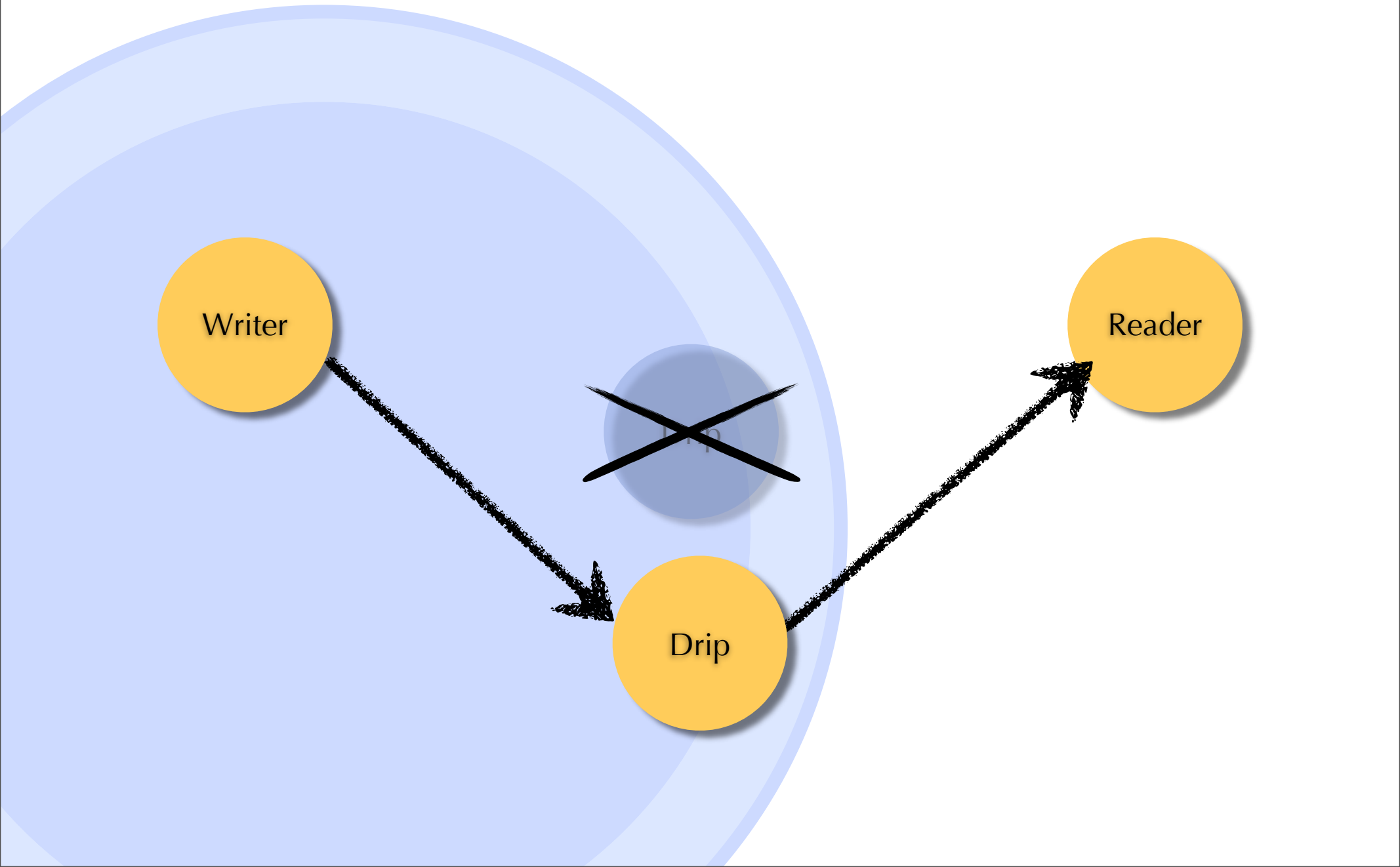
write, and read

```
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

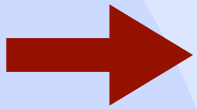


Dripを再起動

- 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]
```

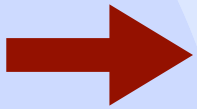


key	value
2	'hello'
3	'world'
5	'Hello, Again.'

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

- ✓ +15:00
- ✓ 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

- ✓ +20:00
- ✓ 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

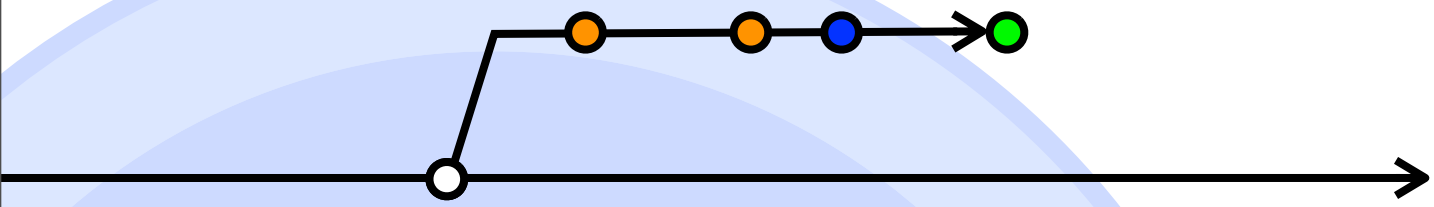
```
while ture
  ary = drip.read(key, 4, 1)
  ...
  key = ary[-1][0]
end
```

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
```

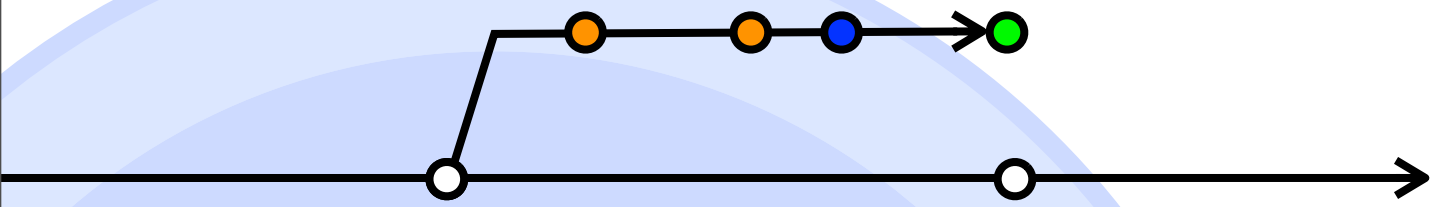
読み進める



- 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
```

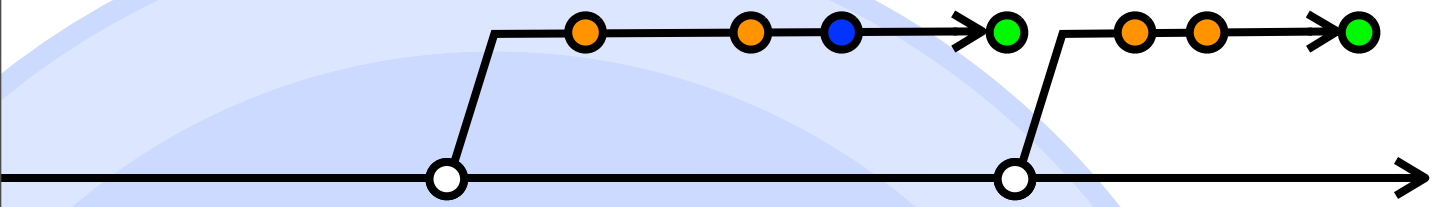
読み進める



- 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
```

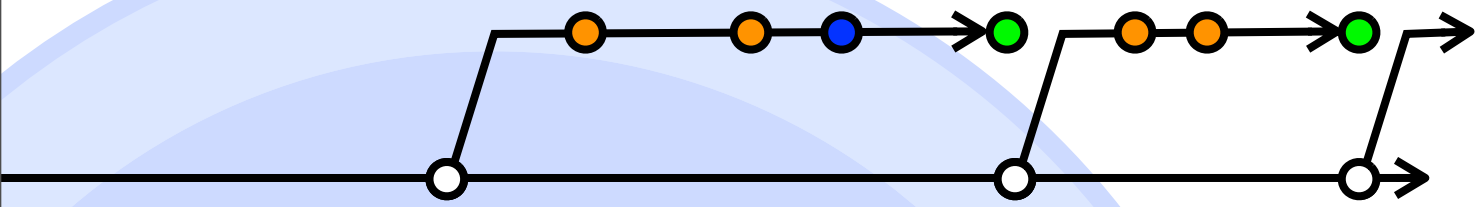
読み進める



- 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
```

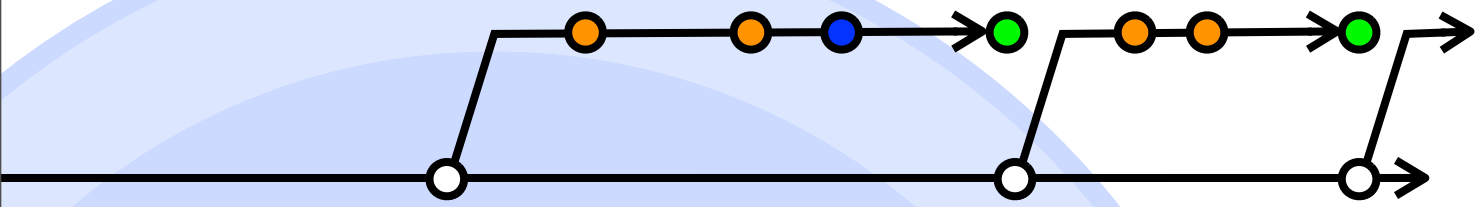
読み進める



- 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
```

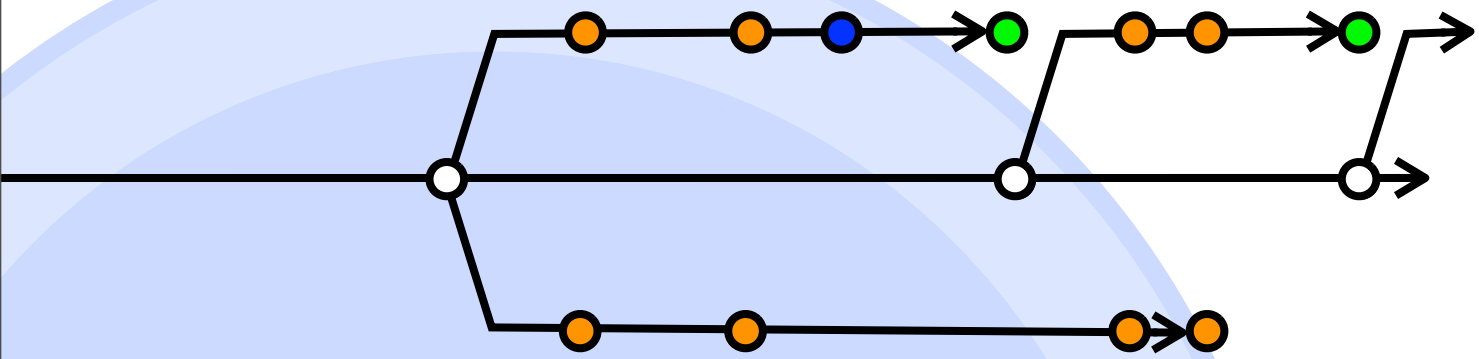
フィルタして読む



- 興味があるものだけ返す

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

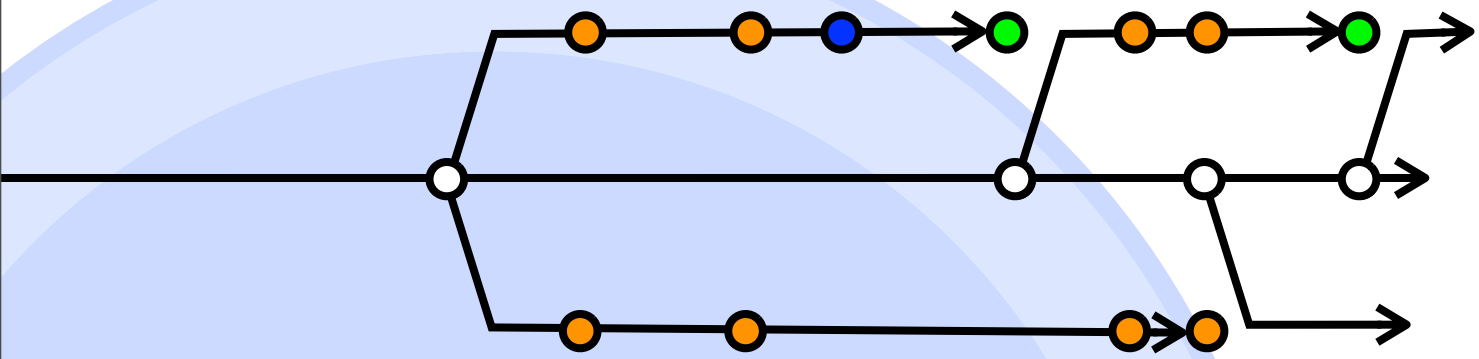
フィルタして読む



- 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

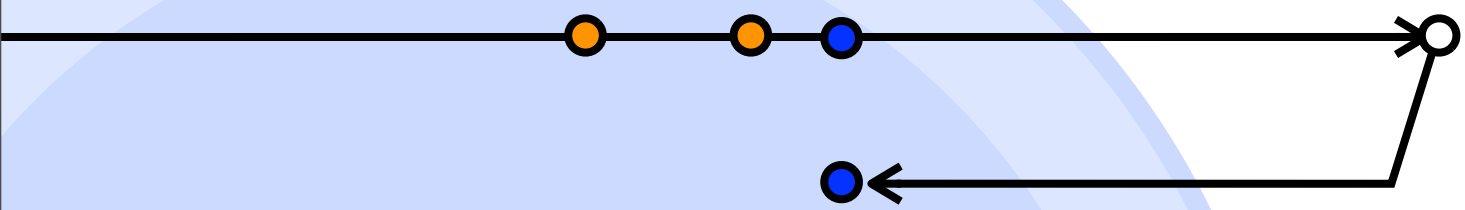
```
k, v = head(1, 'blue')[0]
read(k, 5)
```

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)
```

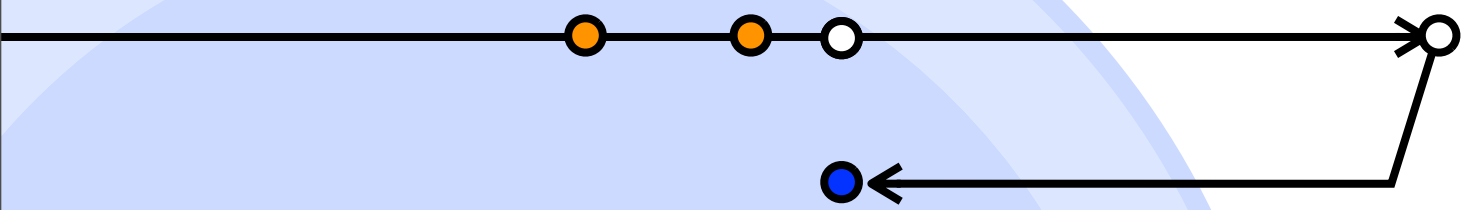
ちょっと戻ってから読む



- 最新の'blue'から5つずつください

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

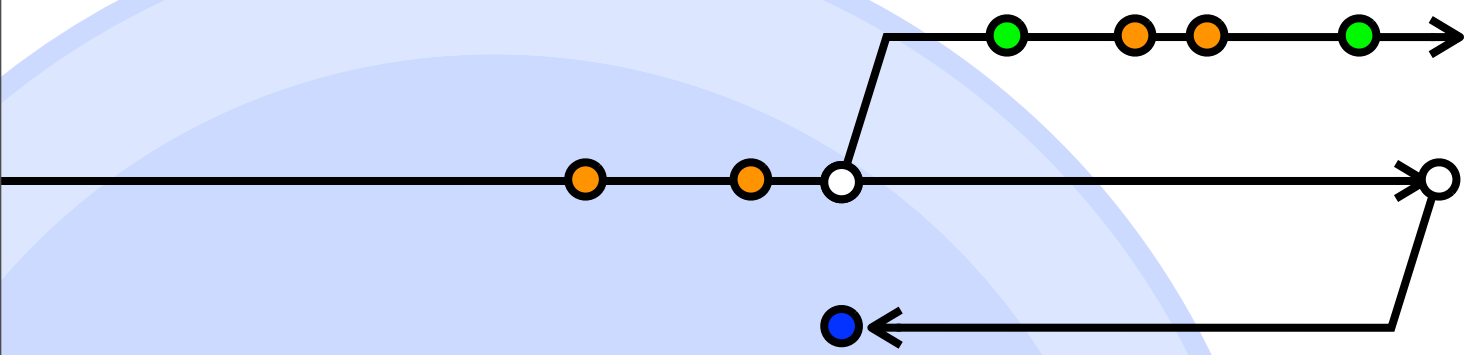
ちょっと戻ってから読む



- 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

- ✓ +25:00
- ✓ ブラウズのようすをなんとなくわかって
- ✓ 蘊蓄を言い過ぎてないか確認

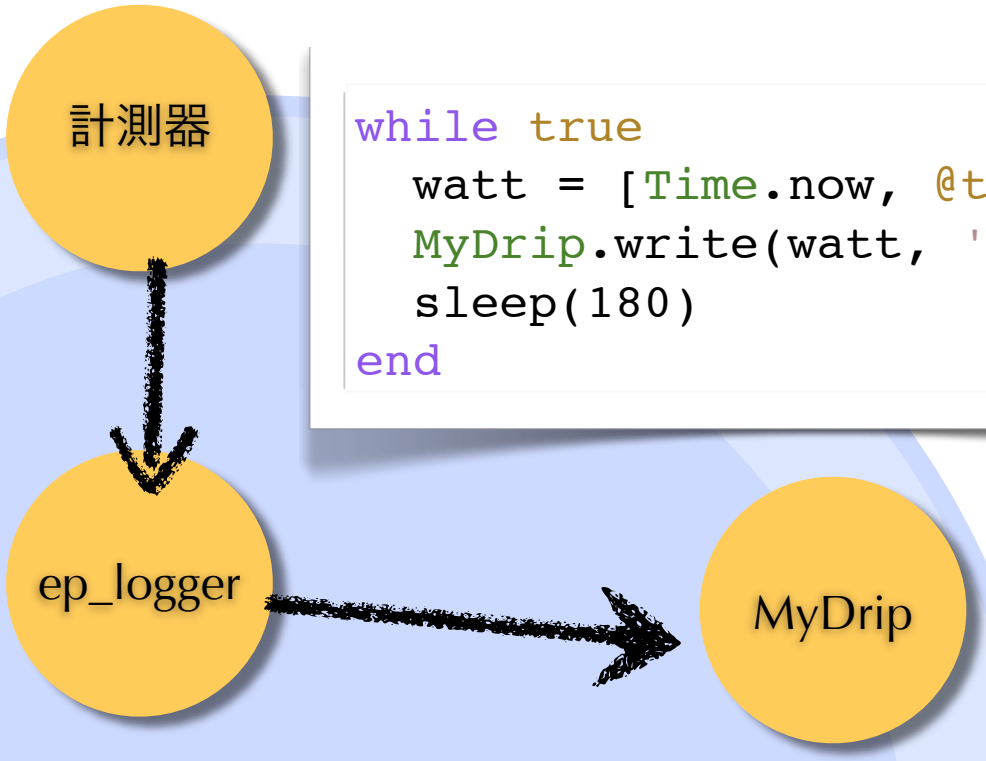
今日の話

- PTupleSpaceの反省
  - 単純な永続化では足りない
- 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

日本先行発売

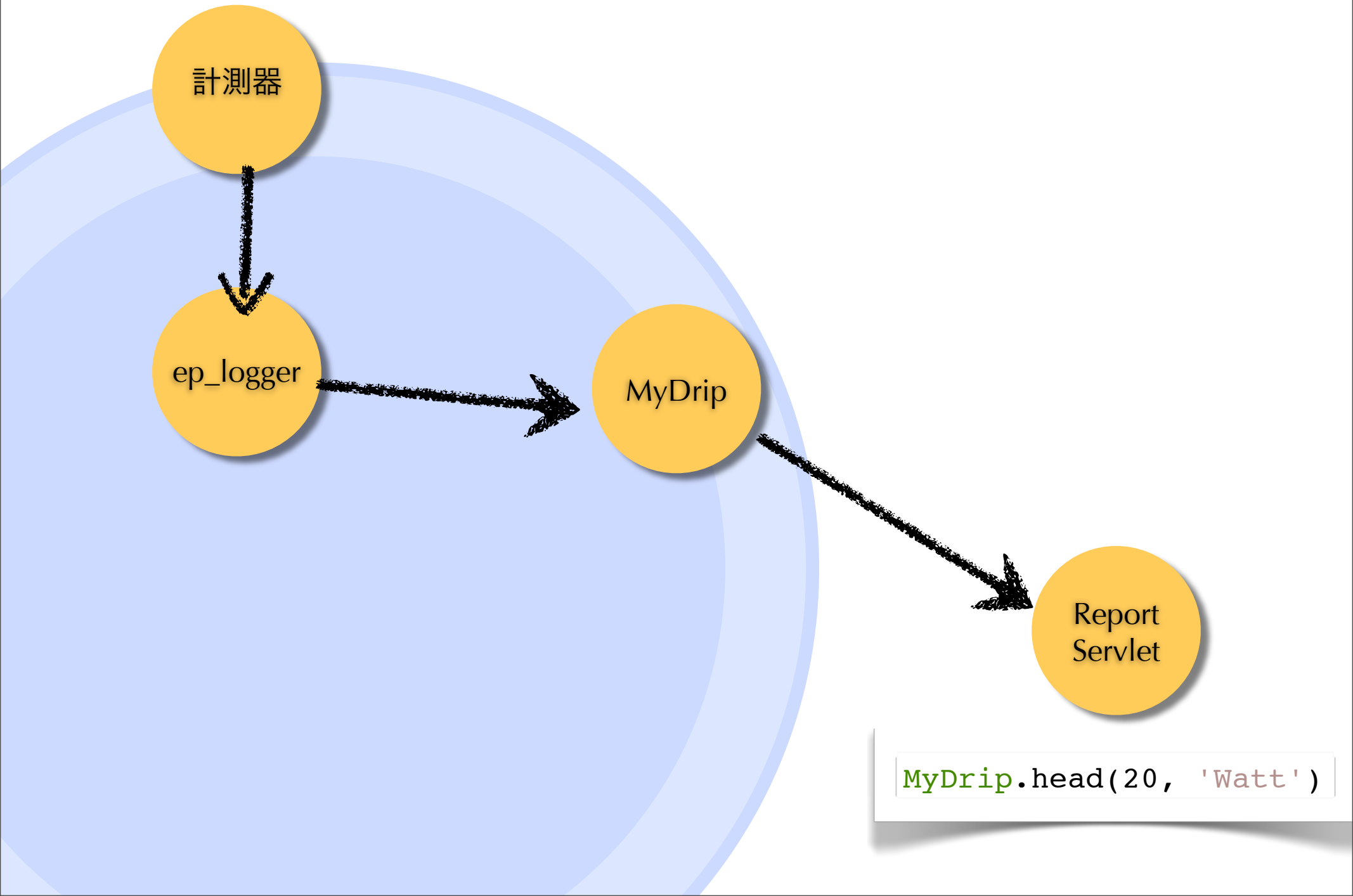
- 次は英語で初刷を

電力消費量ロガー＆レポート



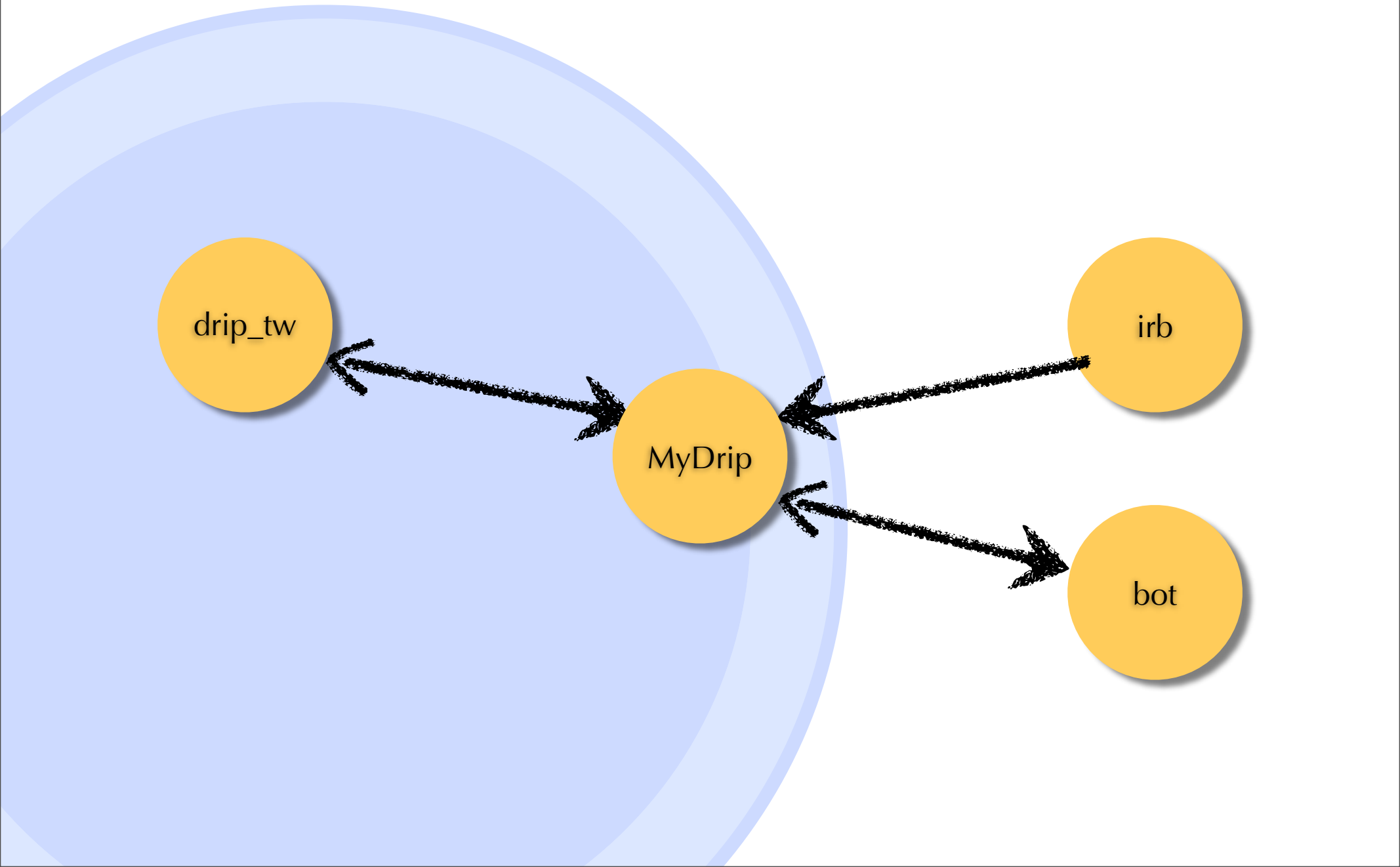

```
while true
  watt = [Time.now, @there.value]
  MyDrip.write(watt, 'Watt')
  sleep(180)
end
```

電力消費量ロガー＆レポート



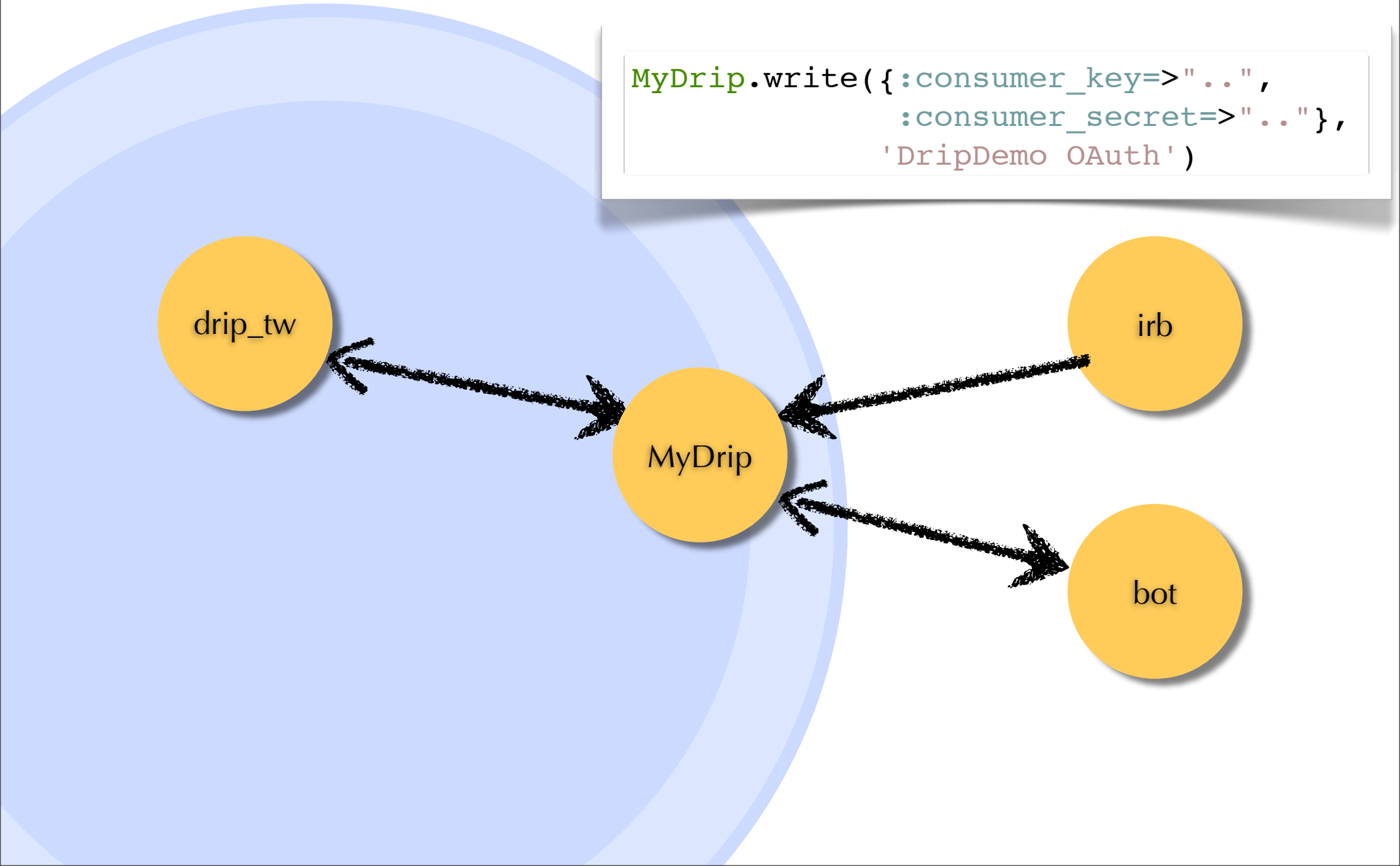

```
MyDrip.head(20, 'Watt')
```

sample/drip_tw.rb

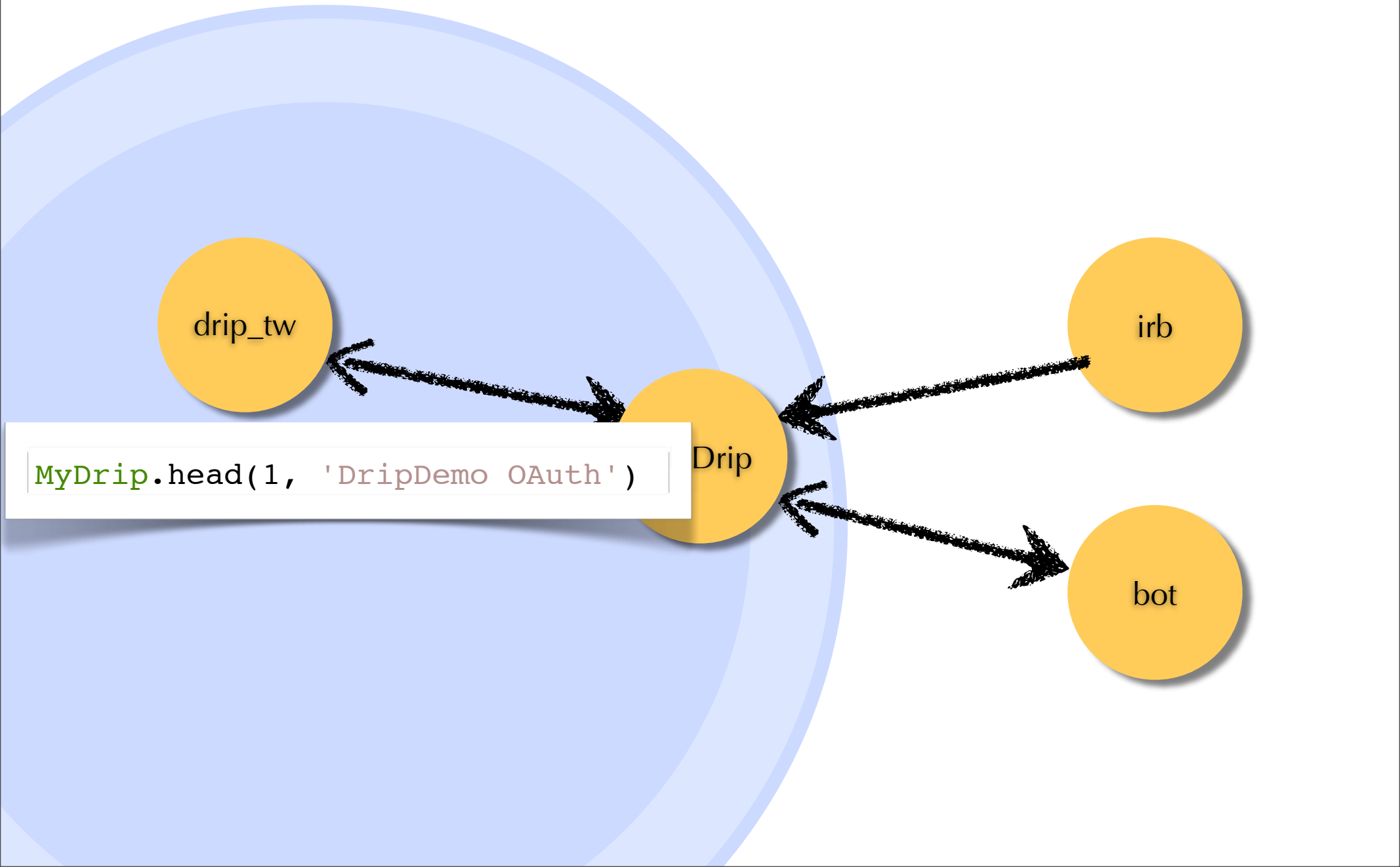


sample/drip_tw.rb

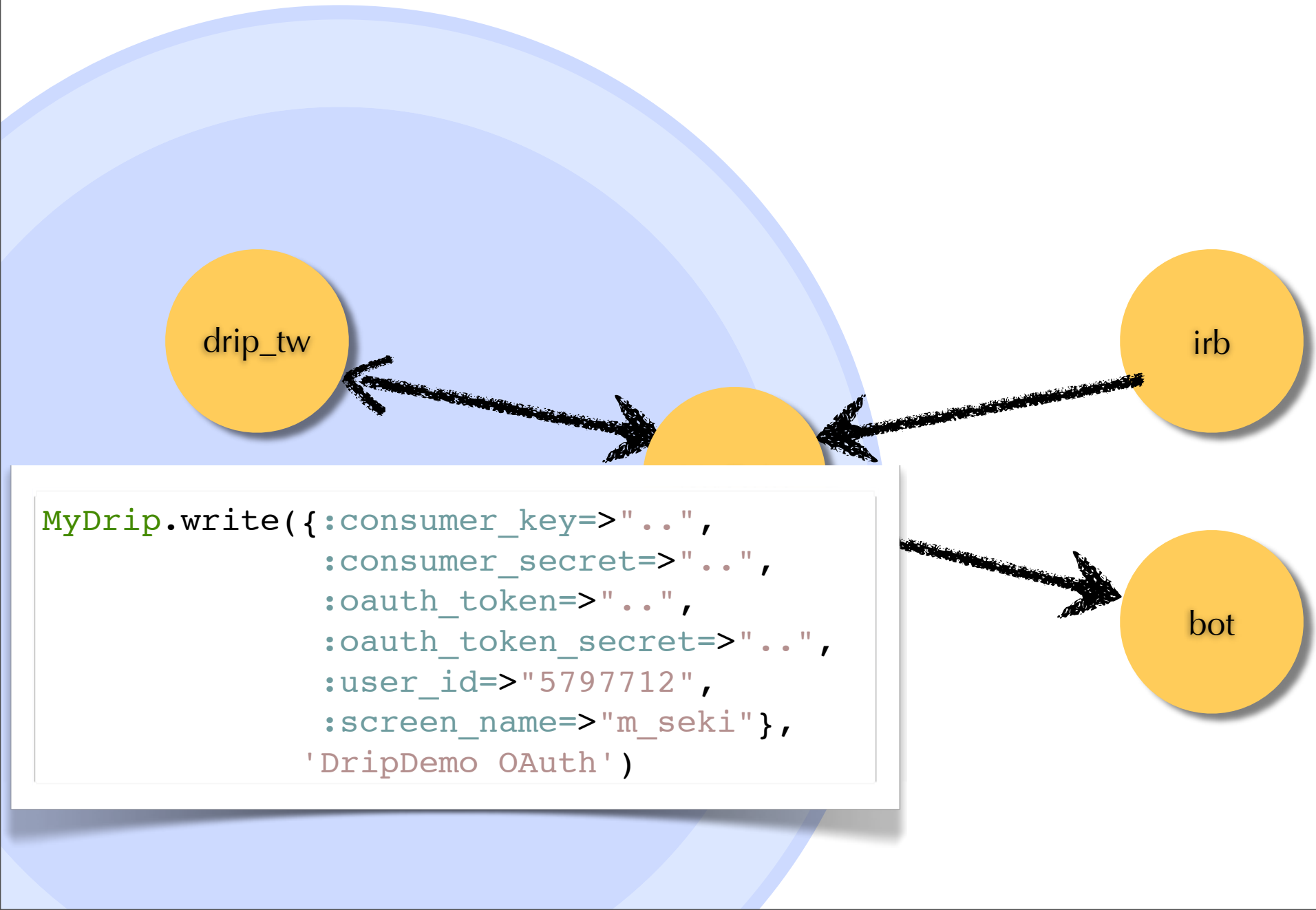
```
MyDrip.write({:consumer_key=>"..",
              :consumer_secret=>".."},
             'DripDemo OAuth')
```



sample/drip_tw.rb

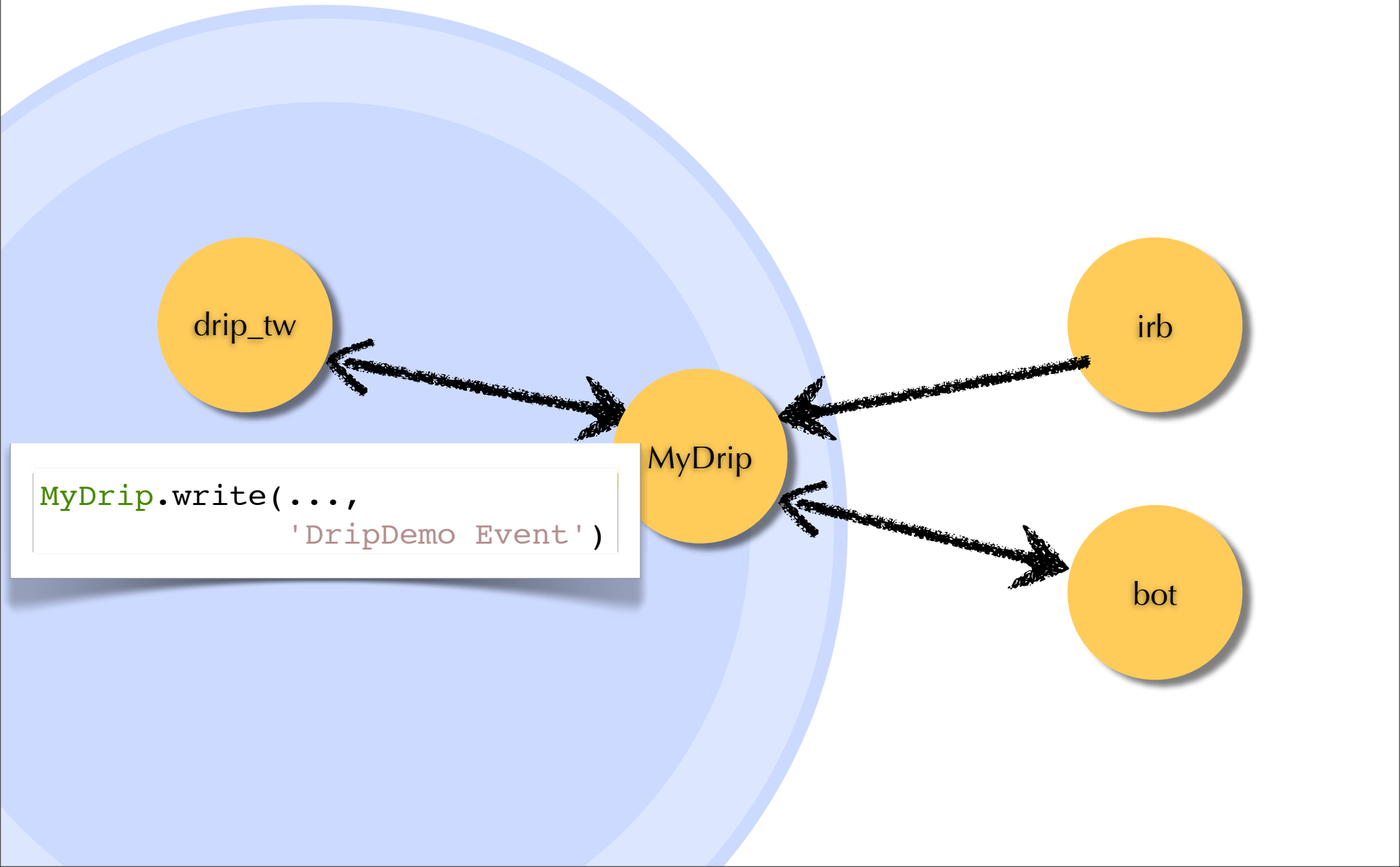


sample/drip_tw.rb

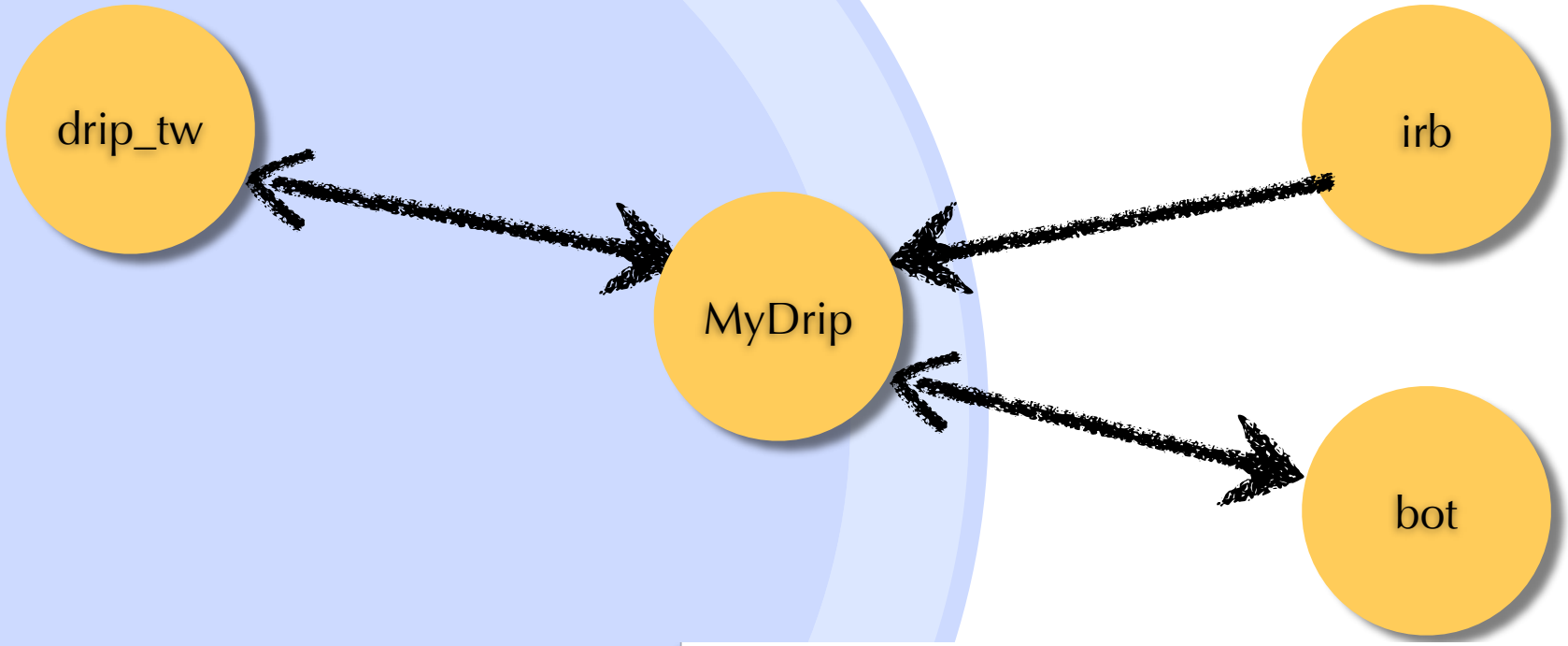



```
MyDrip.write({:consumer_key=>"..",
              :consumer_secret=>"..",
              :oauth_token=>"..",
              :oauth_token_secret=>"..",
              :user_id=>"5797712",
              :screen_name=>"m_seki"},
             'DripDemo OAuth')
```

sample/drip_tw.rb

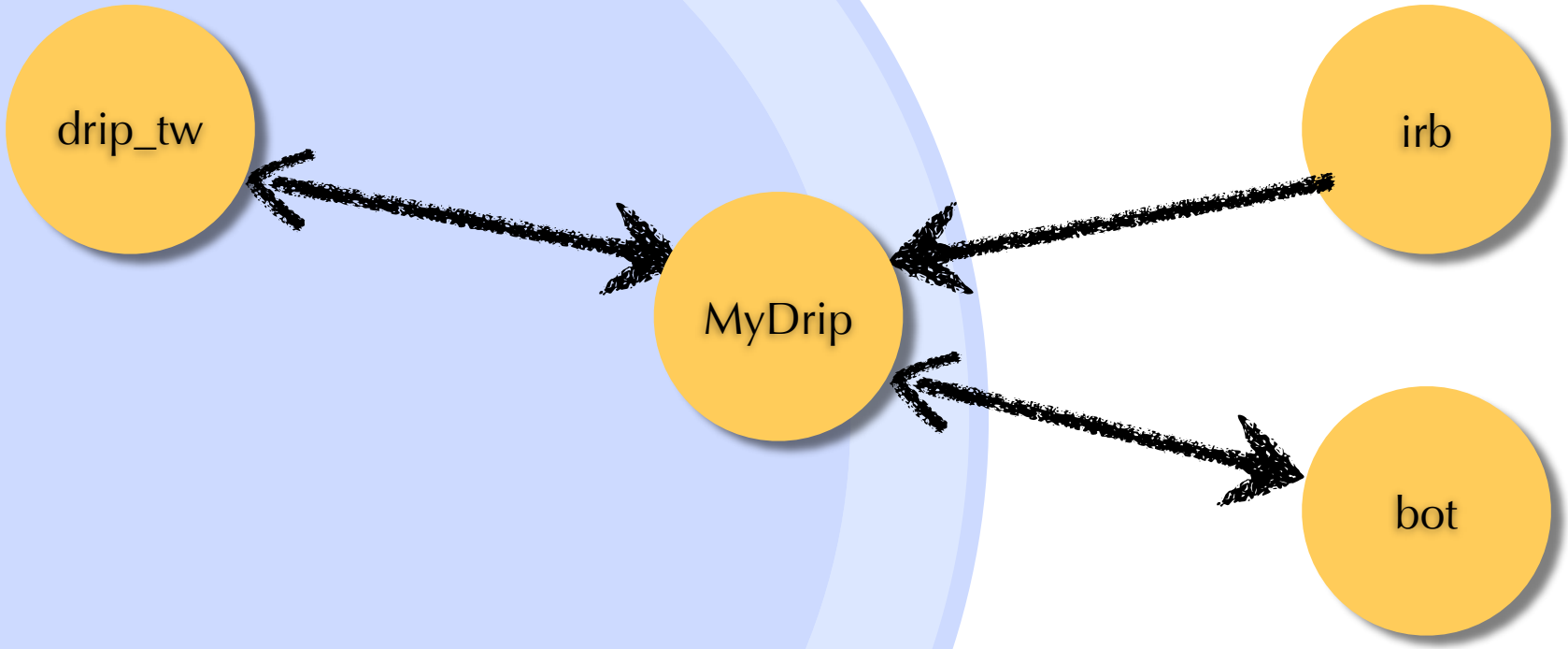


sample/drip_tw.rb




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

sample/drip_tw.rb




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