# Taming concurrency with cooperation

CUSO Winter School, 2018

Dimitri Racordon

# Model everything!

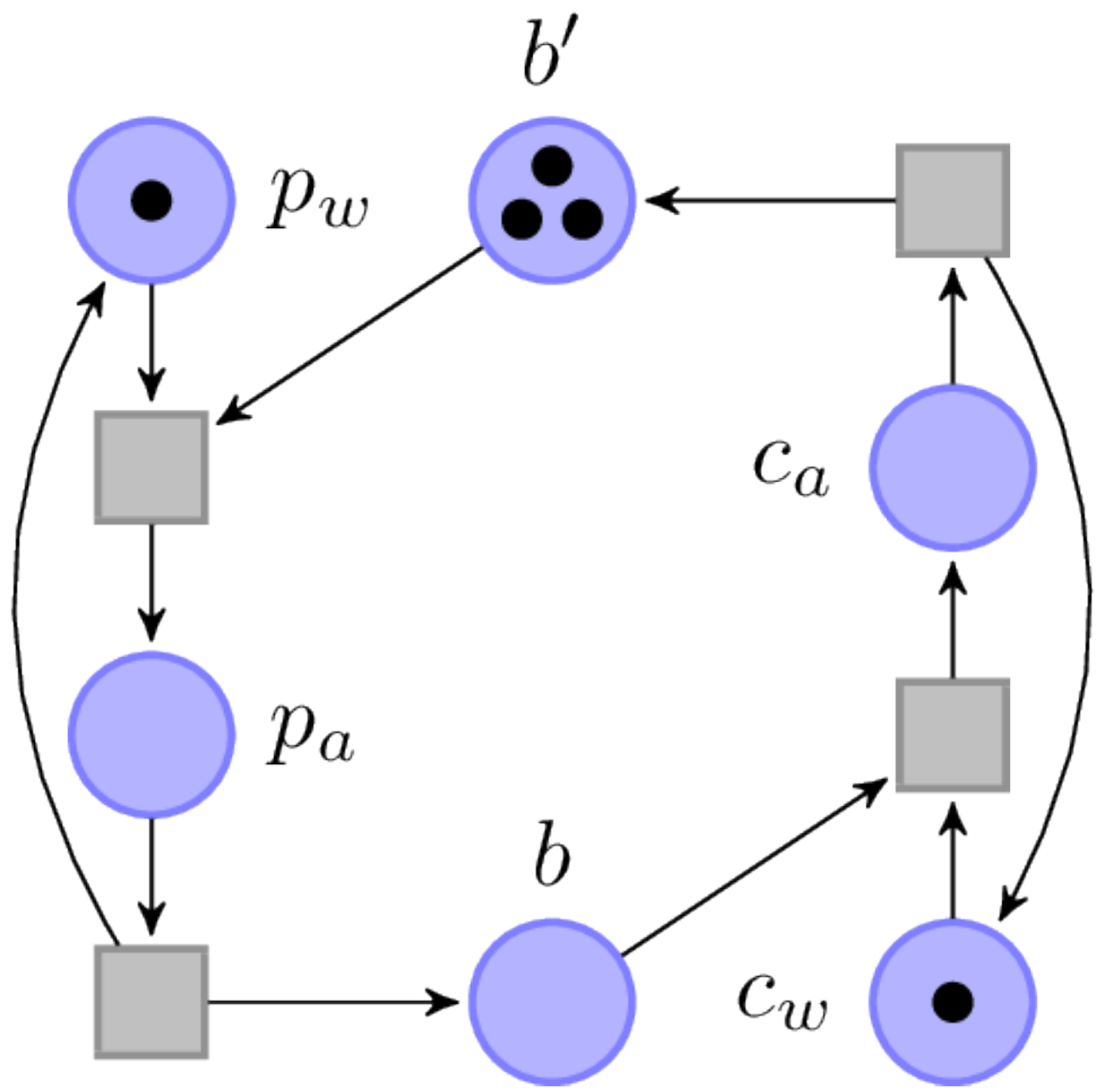
*Hans Vangheluwe*

# Model everything ...

*Hans Vangheluwe*

# ... with Petri nets!

*Didier Buchs*



To the implementation!

# The problem with preemption

```
buf = []

def produce():
  while True:
    if len(buf) < 3:
      buf.append(Unit())

def consume():
  while True:
    if buf:
      print(buf.pop())
```

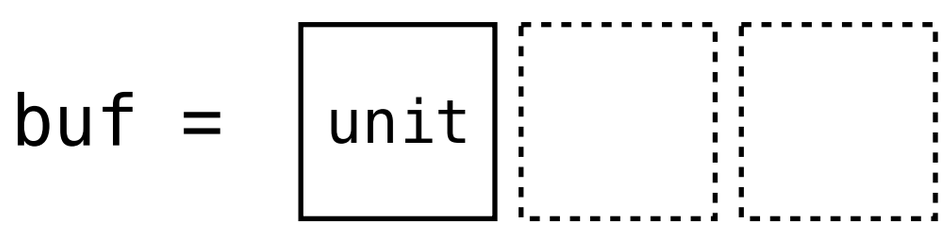
# The problem with preemption

Thread 1

```
def consume():
  while True:
    if buf:
      print(buf.pop())
```

Thread 2

```
def consume():
  while True:
    if buf:
      print(buf.pop())
```



# The problem with preemption

Thread 1

```
def consume():
  while True:
    if buf:
      print(buf.pop())
```

Thread 2

```
def consume():
  while True:
    if buf:
      print(buf.pop())
```

# The problem with preemption

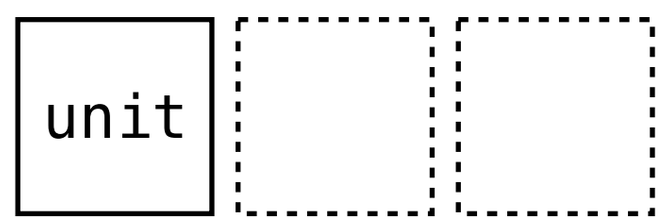
Thread 1

```
def consume():
  while True:
    if buf:
      print(buf.pop())
```

Thread 2

```
def consume():
  while True:
    if buf:
      print(buf.pop())
```

buf =



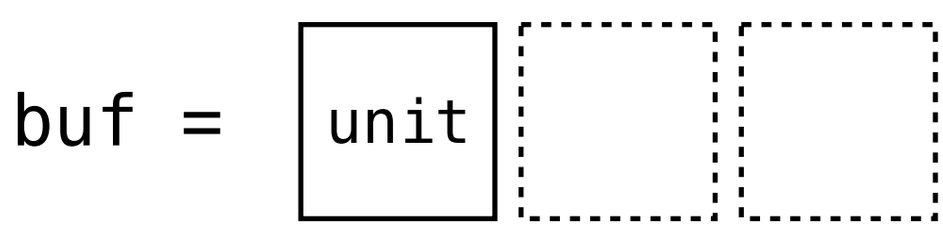
# The problem with preemption

Thread 1

```
def consume():
  while True:
    if buf:
      print(buf.pop())
```

Thread 2

```
def consume():
  while True:
    if buf:
      print(buf.pop())
```



# The problem with preemption

Thread 1

```
def consume():
  while True:
    if buf:
      print(buf.pop())
```

Thread 2

```
def consume():
  while True:
    if buf:
      print(buf.pop())
```

buf = 

# The problem with preemption

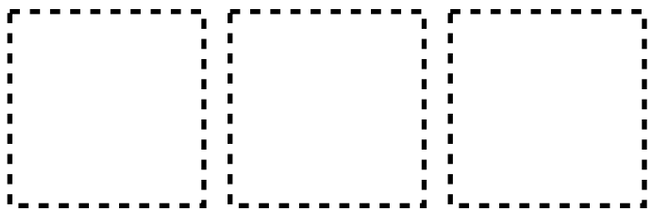
Thread 1

```
def consume():
  while True:
    if buf:
      print(buf.pop())
```

Thread 2

```
def consume():
  while True:
    if buf:
      print(buf.pop())
```

buf =



# The problem with preemption

Thread 1

```
def consume():
  while True:
    if buf:
      print(buf.pop())
```

Thread 2

```
def consume():
  while True:
    if buf:
      print(buf.pop())
```

buf =

# The problem with preemption

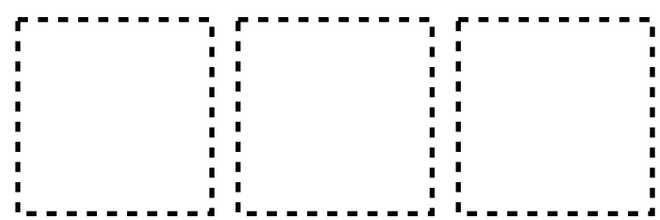
Thread 1

```
def consume():
  while True:
    if buf:
      print(buf.pop())
```

Thread 2

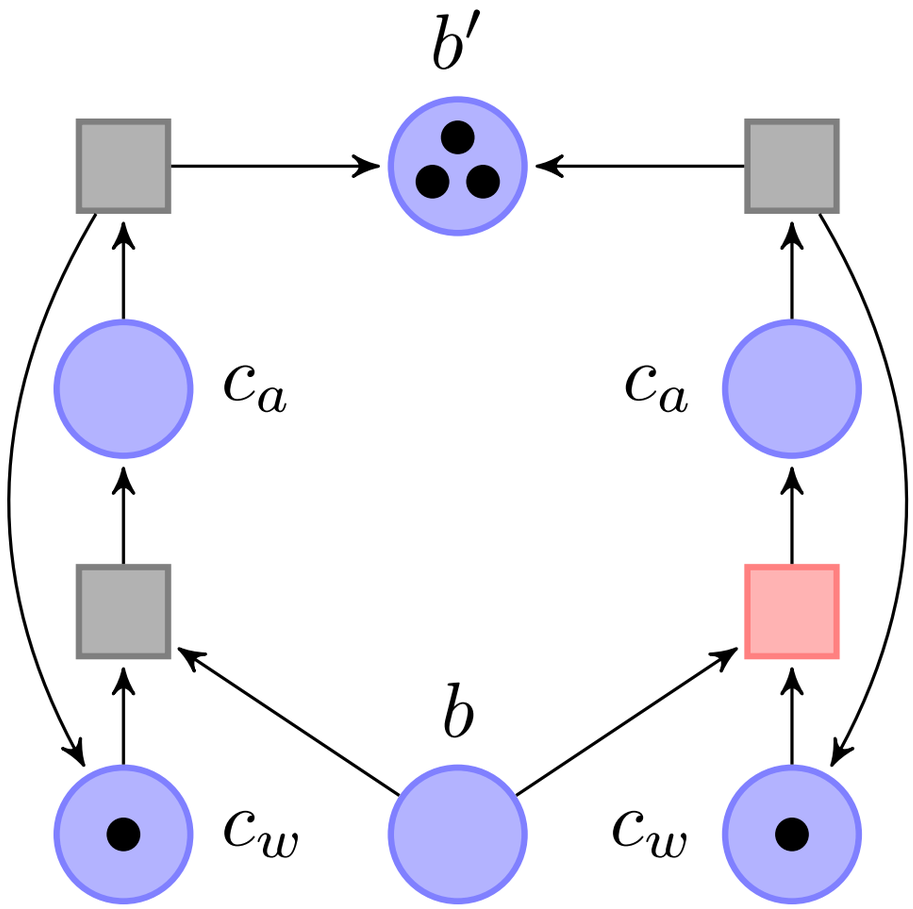
```
def consume():
  while True:
    if buf:
      print(buf.pop())
```

buf =



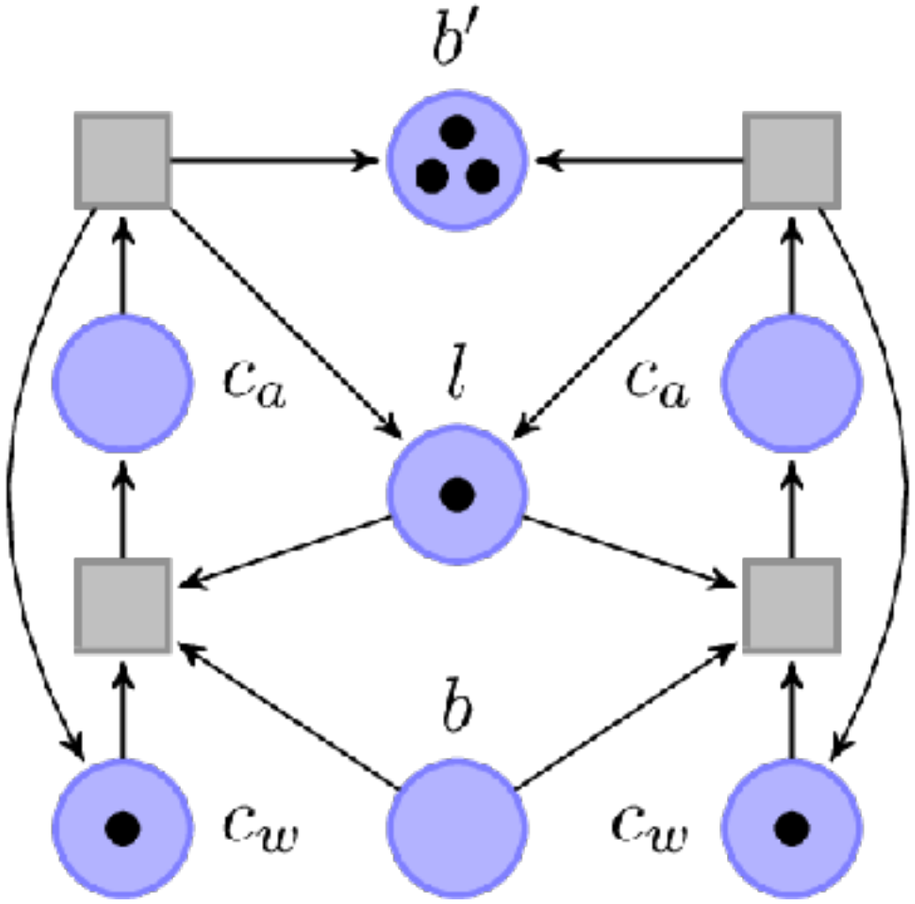
# The problem with preemption

```
def consume():
  while True:
    if buf:
      print(buf.pop())
```

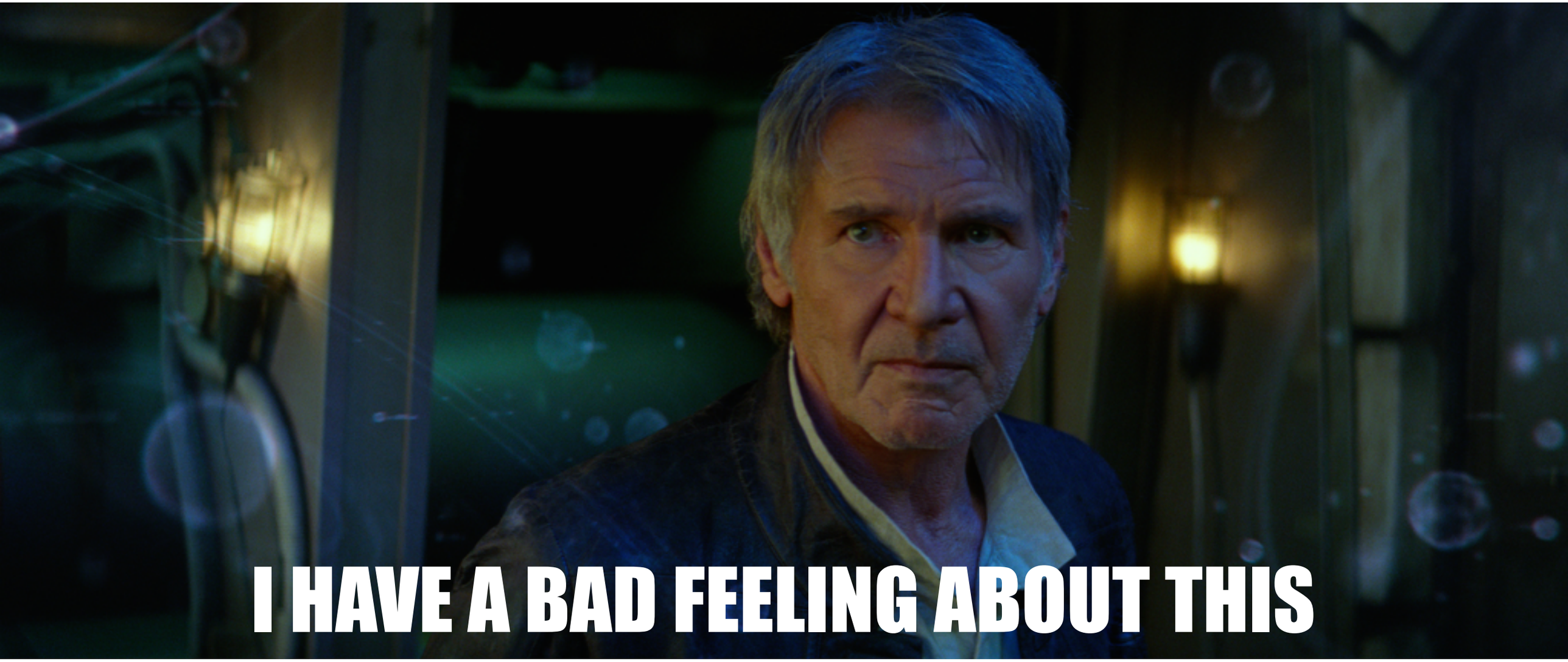


# The problem with preemption

```
def consume():
  while True:
    take_lock()
    if buf:
      print(buf.pop())
    release_lock()
```



# The problem with preemption

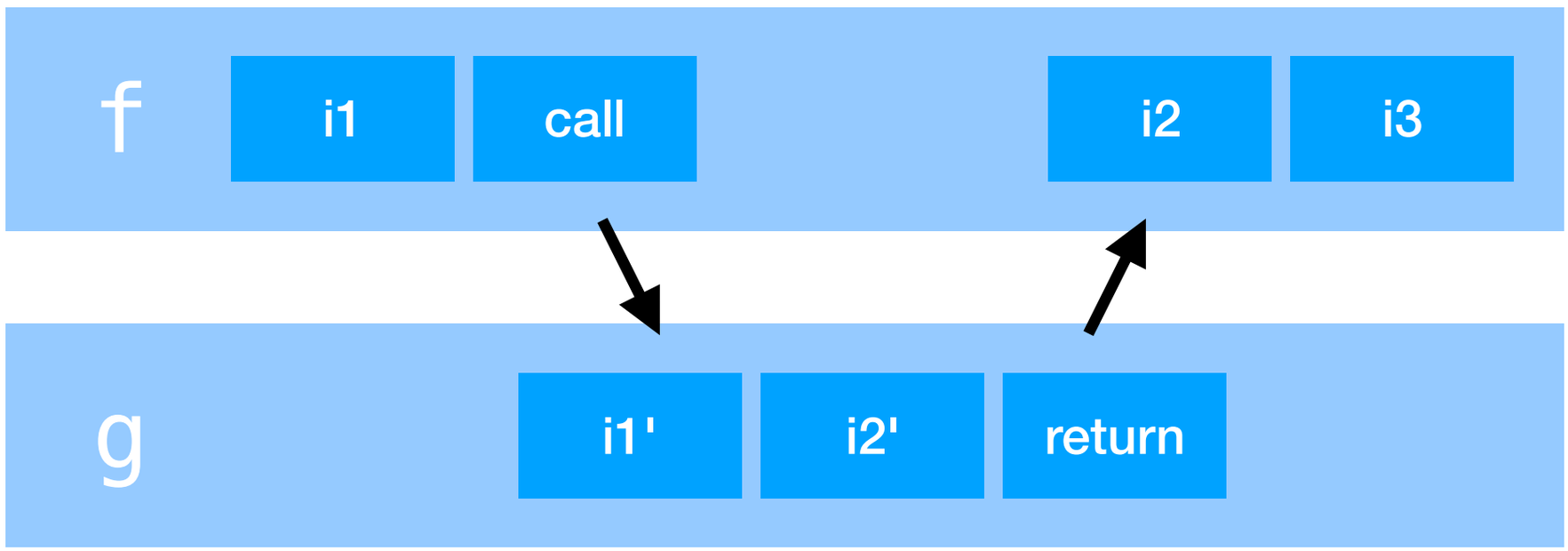


# The problem with preemption

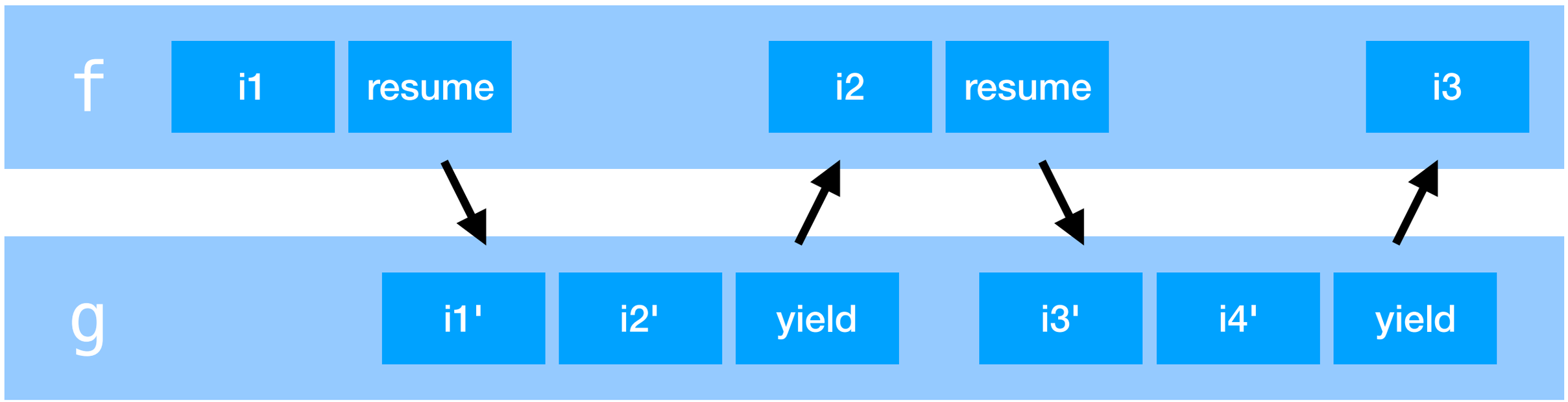
```
def consume():
  while True:
    if buf:
      print(buf.pop())
```

} Obviously atomic

# Subroutine



# Coroutine



# Back to the example

```
buf = []

async def produce():
  while True:
    if len(buf) < 3:
      buf.append(Unit())
    yield

def consume():
  while True:
    if buf:
      print(buf.pop())
    yield
```

# Back to the example

```
buf = []

async def produce():
  while True:
    if len(buf) < 3:
      buf.append(Unit())
    yield

def consume():
  while True:
    if buf:
      print(buf.pop())
    yield
```

# The code is the model!

*- many people*

# The code is <u>the</u> model!

*- me*

# What about semantics?

Statement List

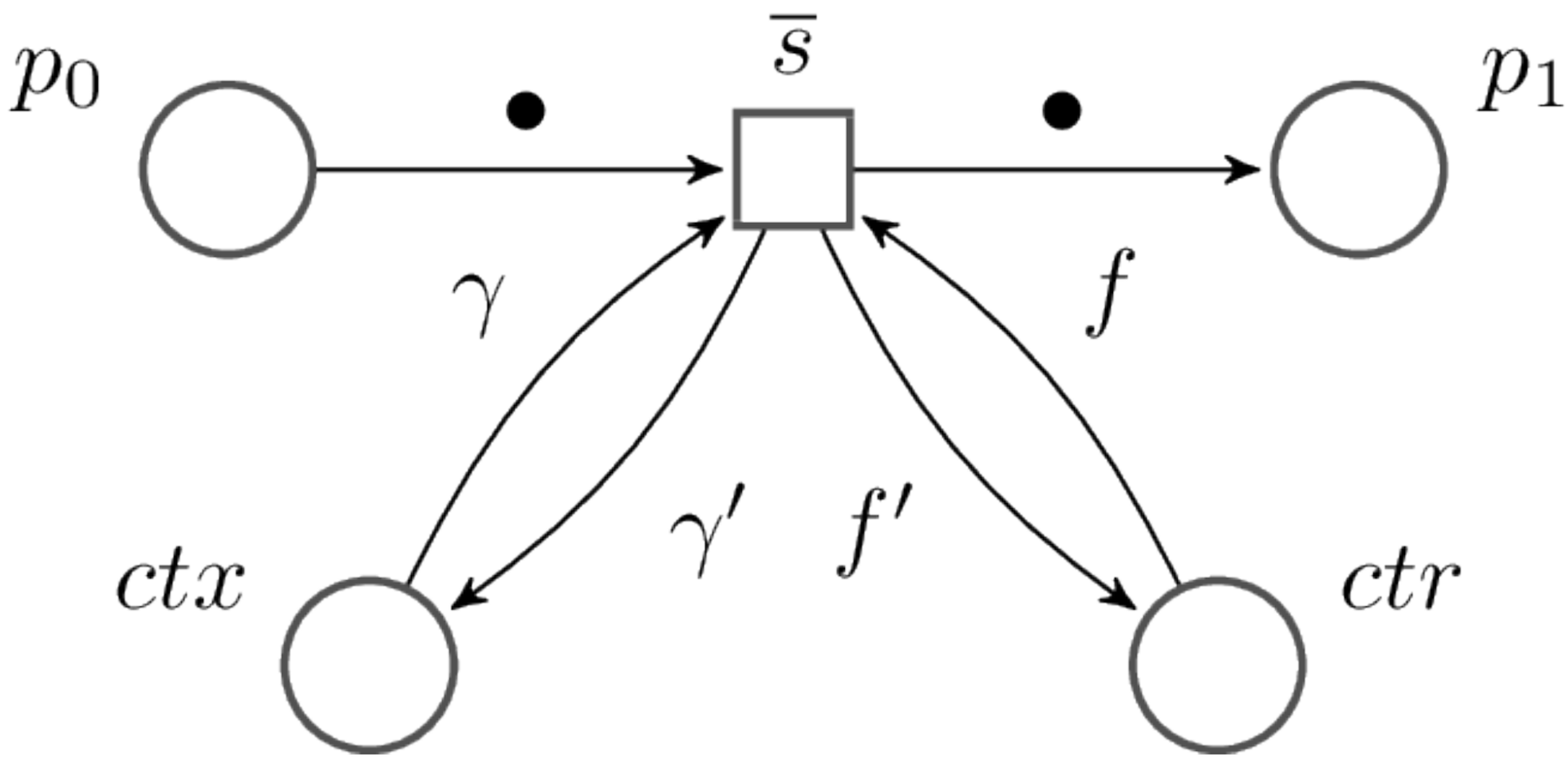
Updated Context

$$\boxed{\gamma} \vdash \boxed{\overline{s}} \Rightarrow \boxed{\overline{s}'}, \boxed{\gamma'}$$

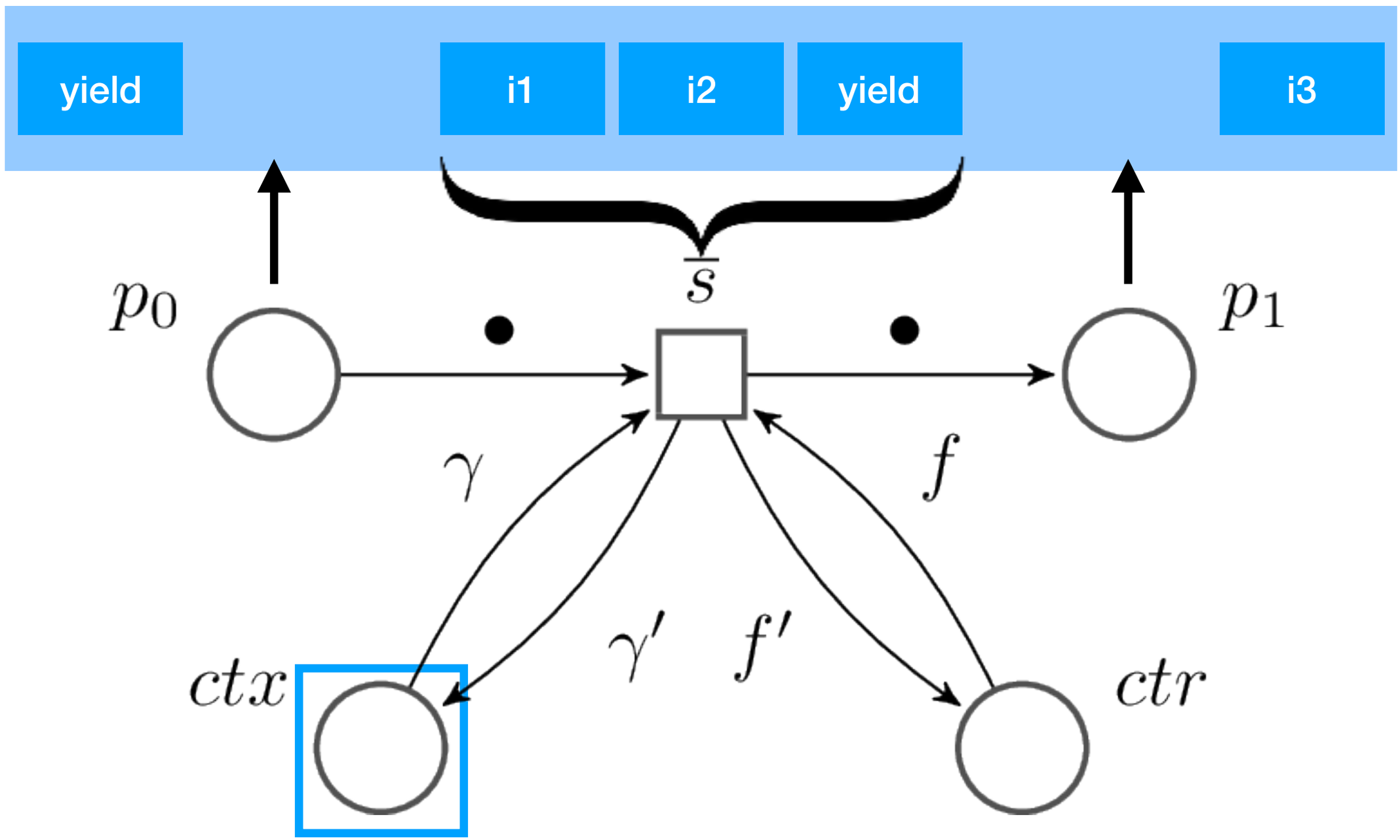
Evaluation Context

Remaining Statements

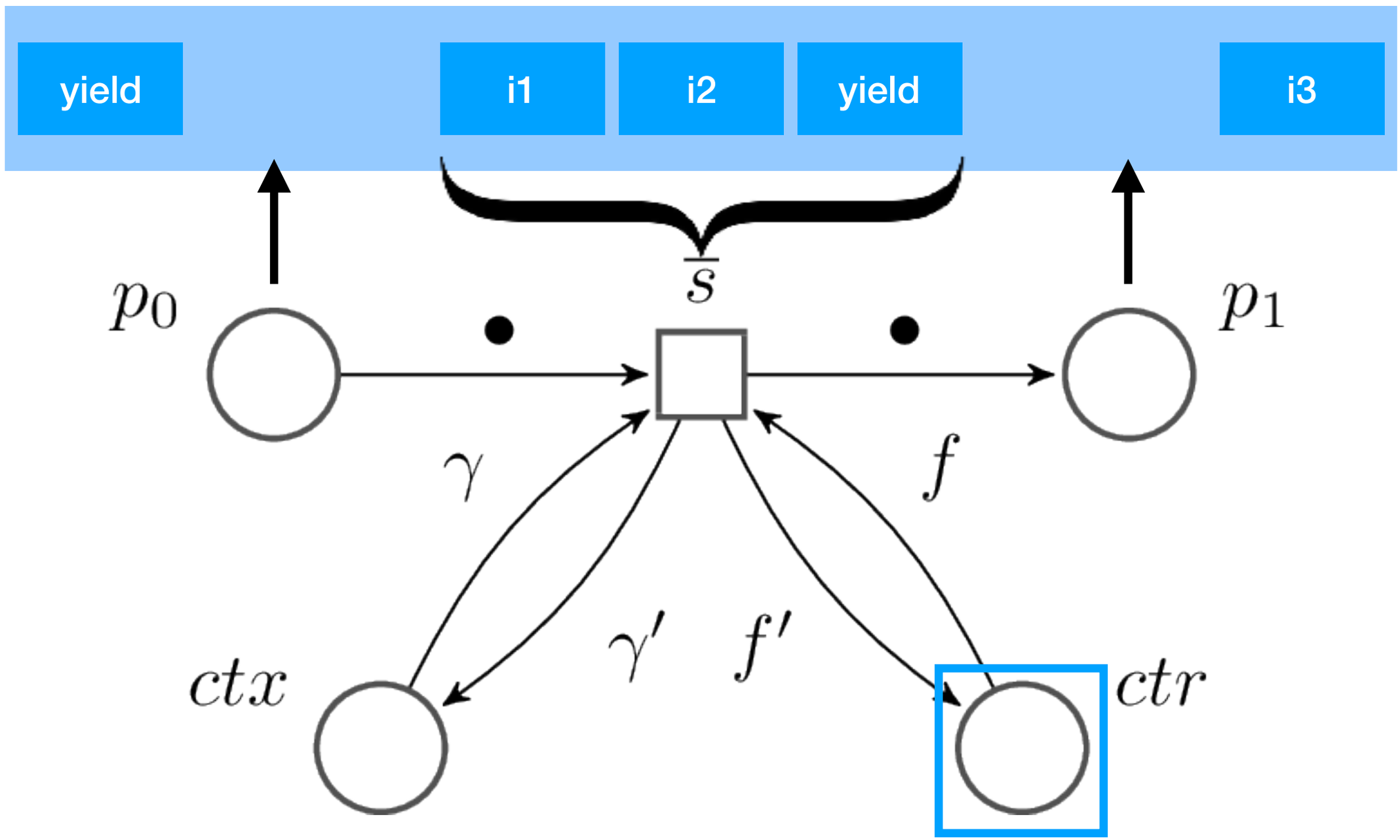
# What about semantics?

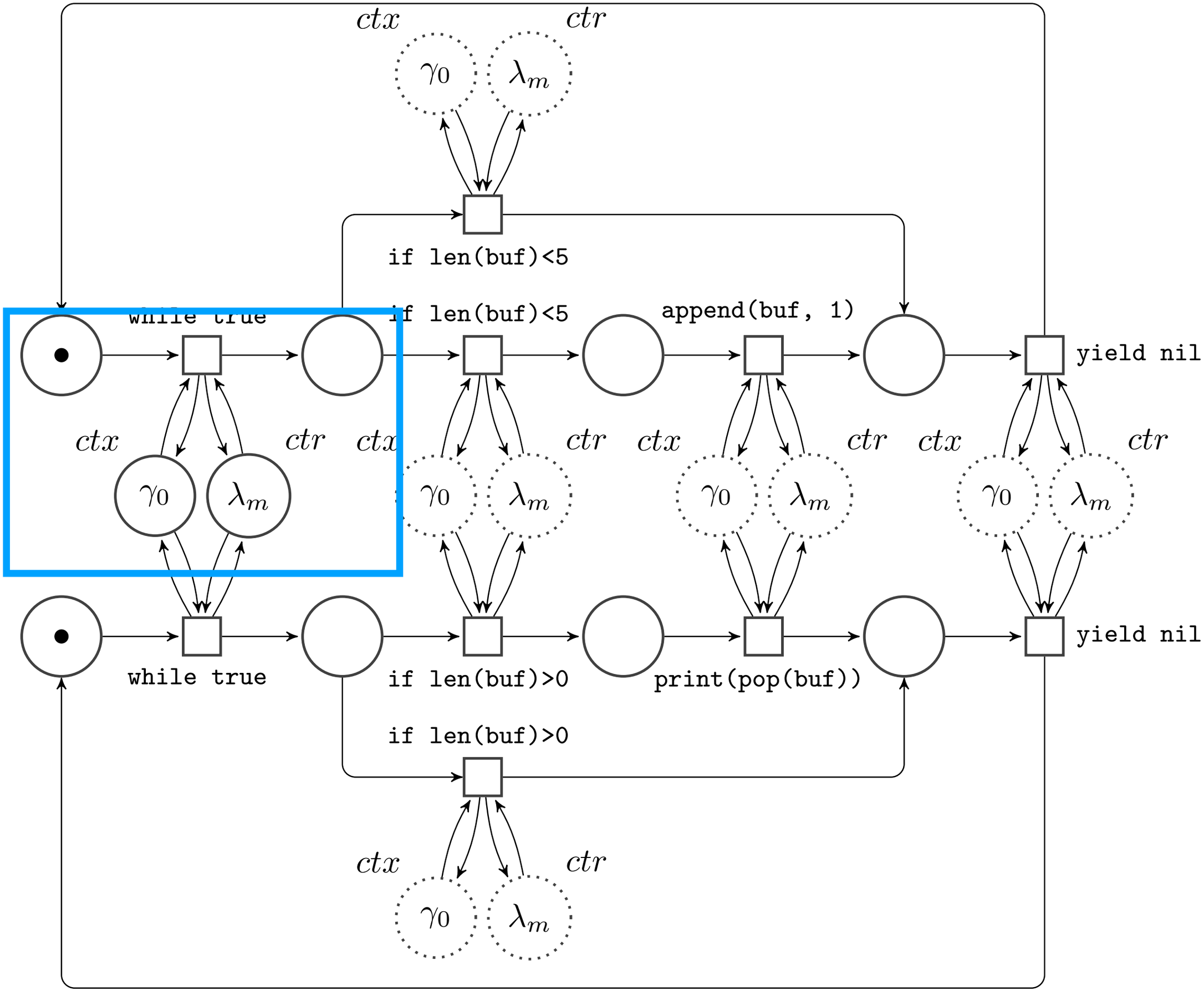


# What about semantics?



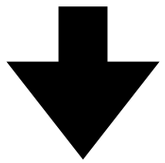
# What about semantics?




ctx
ctr
$\gamma_0$
$\lambda_m$
if len(buf)<5
while true
if len(buf)<5
append(buf, 1)
yield nil
while true
if len(buf)>0
print(pop(buf))
yield nil
if len(buf)>0

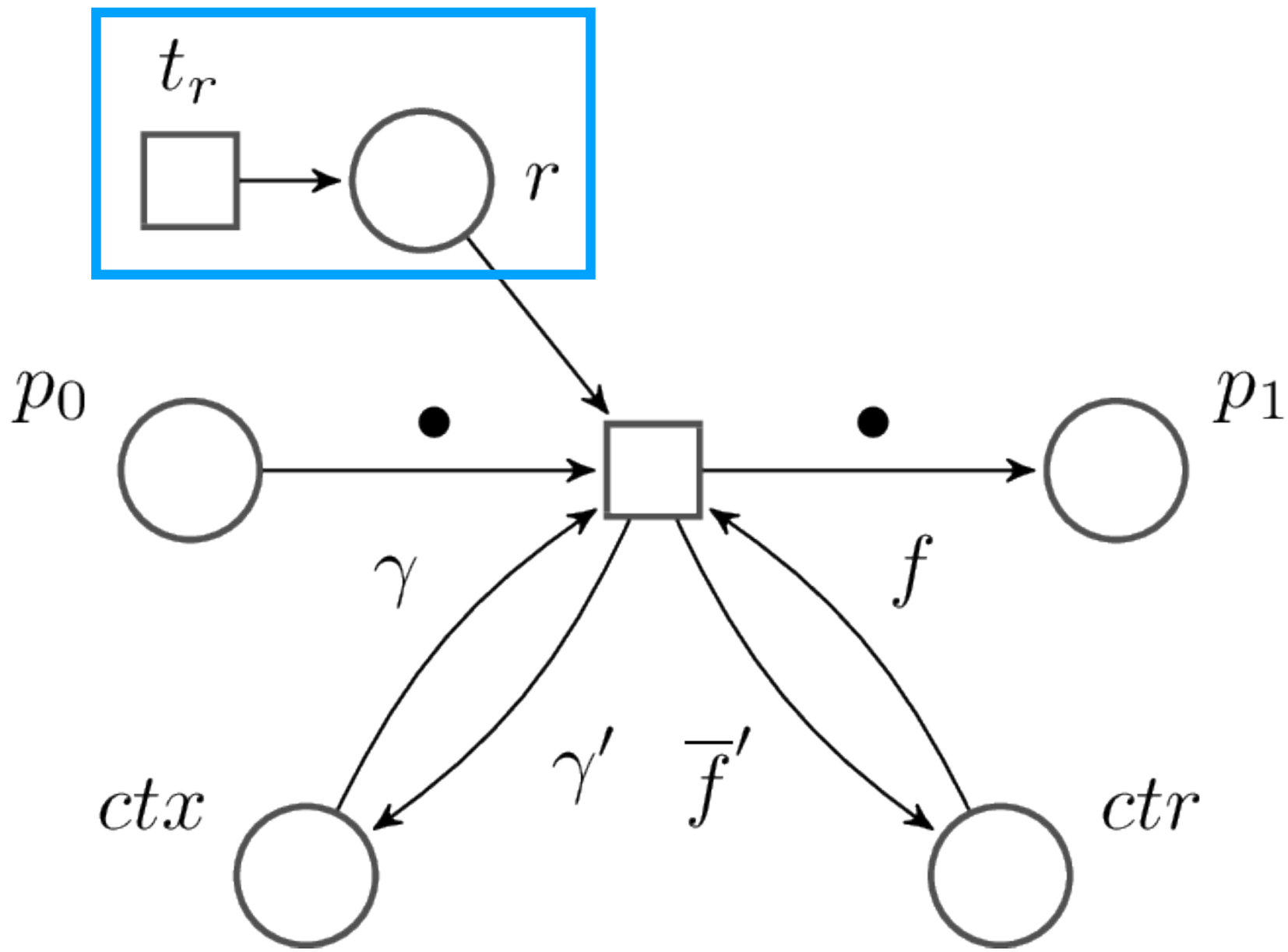
# What about true asynchronism?

```
x = open('x.txt')
y = open('y.txt')
z = open('z.txt')
```



```
x, y, z = await open_many([
  'x.txt', 'y.txt', 'z.txt',
])
```

# Refining our translation



```
print('Thanks!')
while True:
  q = await question
  q.answer()
```