Advanced Process Mining

Summer Semester 2020

Lecture XII: Stream Based Process Mining

dr.ir. Sebastiaan J. van Zelst

Chair of Process and Data Science
RWTH AACHEN UNIVERSITY

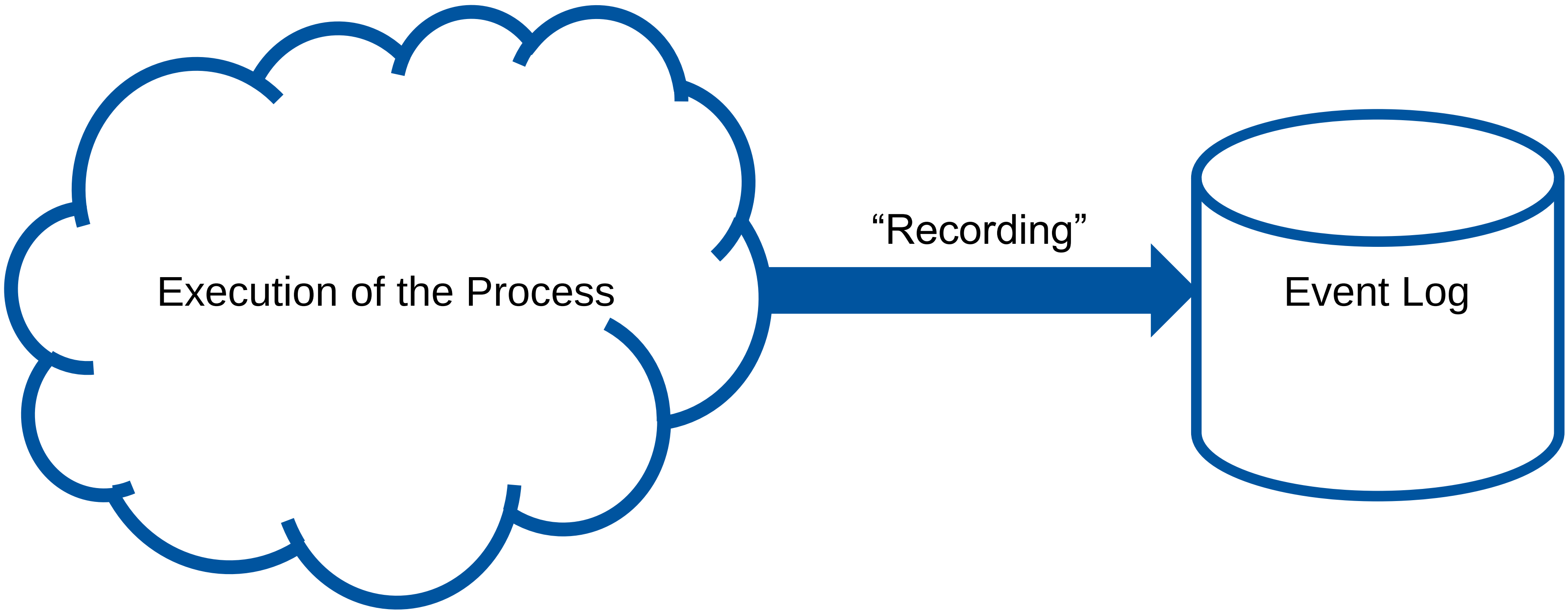
- Motivation
- Streaming Data (Theory)
- Stream Based Discovery
- Online Alignments

https://giffiles.alphacoders.com/133/133737.gif

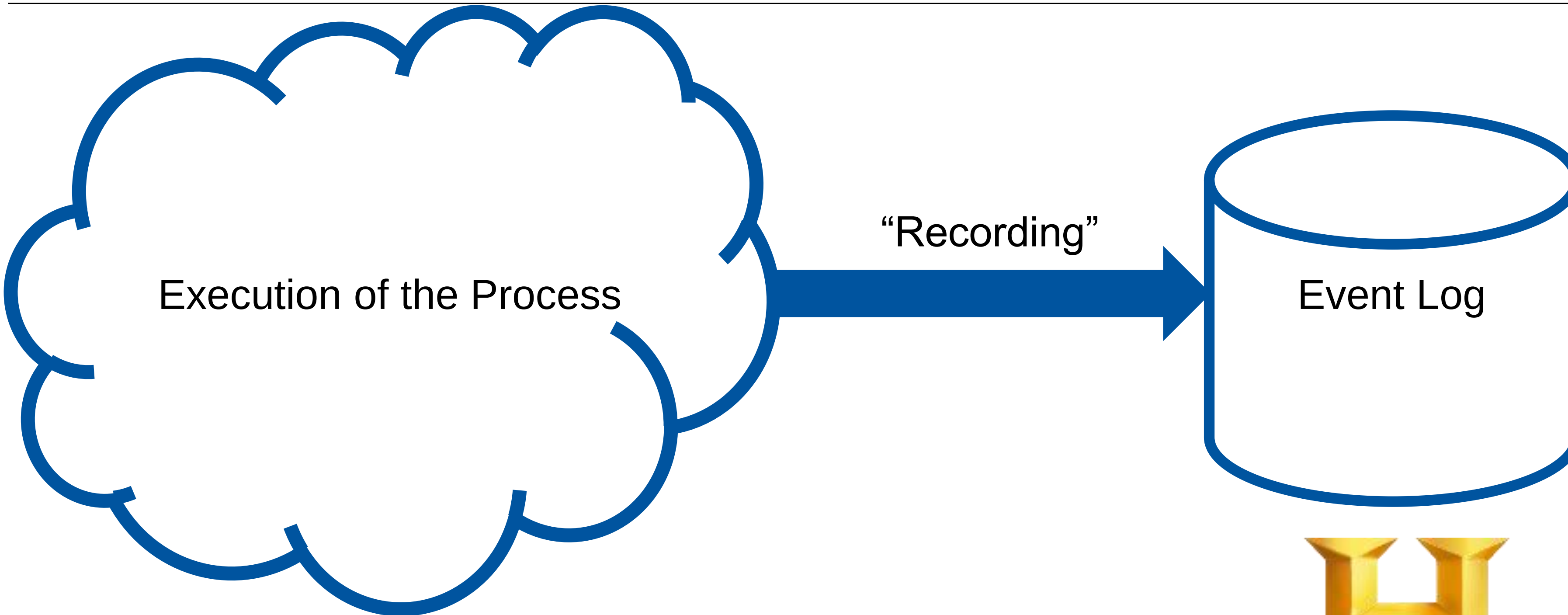
- Motivation
- Streaming Data (Theory)
- Stream Based Discovery
- Online Alignments

https://giffiles.alphacoders.com/133/133737.gif

Motivation



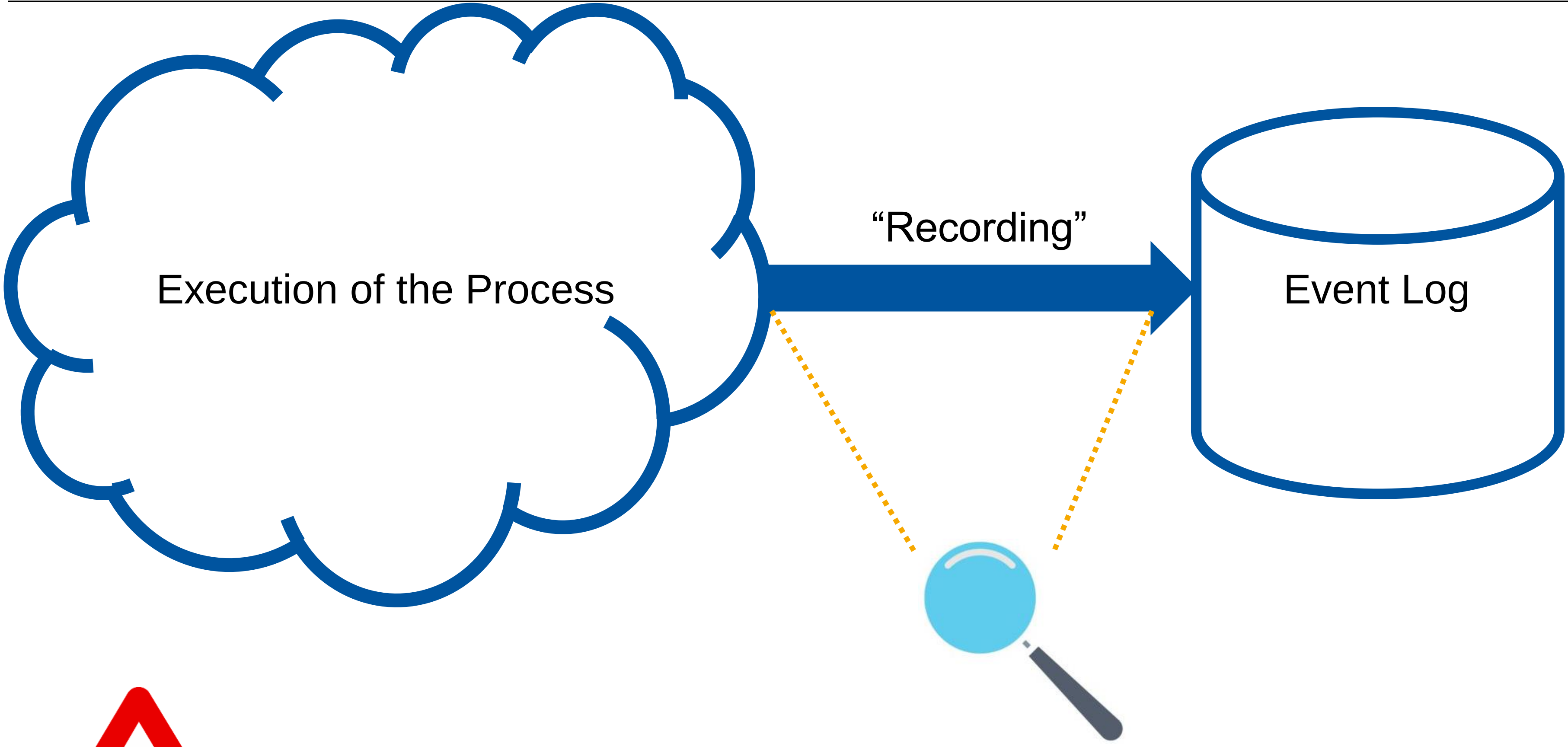
Streaming Data
Motivation



“Something went wrong X months ago!”

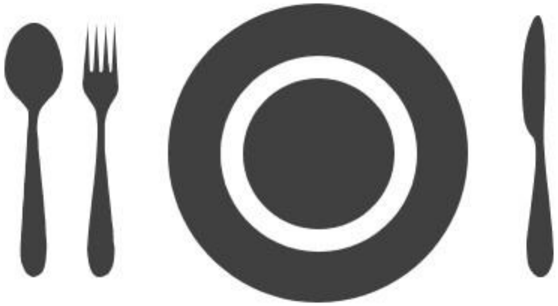
https://directtvlatam-aem65-prod.adobecqms.net/content/dam/public-sites/channels/caribbean/tt/1742.png
https://www.stadt-butzbach.de/wp-content/uploads/warning-sign-30915_1280-1.png

Motivation



"Something went wrong JUST NOW!"

https://directtvlatam-aem65-prod.adobecqms.net/content/dam/public-sites/channels/caribbean/tt/1742.png
https://www.stadt-butzbach.de/wp-content/uploads/warning-sign-30915_1280-1.png



- Motivation
- Streaming Data (Theory)
- Stream Based Discovery
- Online Alignments

https://giffiles.alphacoders.com/133/133737.gif

Counting Numbers

- Let’s assume, we observe an infinite sequence of numbers

Counting Numbers

- Let's assume, we observe an infinite sequence of numbers

5

Counting Numbers

- Let’s assume, we observe an infinite sequence of numbers

37

Counting Numbers

- Let's assume, we observe an infinite sequence of numbers

1,337

Counting Numbers

- Let's assume, we observe an infinite sequence of numbers

145,327,243

Counting Numbers

- Let’s assume, we observe an infinite sequence of numbers

16,531

Counting Numbers

- Let’s assume, we observe an infinite sequence of numbers

...

Counting Numbers

- Let's assume, we observe an infinite sequence of numbers
- **unsigned 'long'** values, i.e., 2^{64} observable numbers

Counting Numbers

- Let's assume, we observe an infinite sequence of numbers
- **unsigned 'long'** values, i.e., 2^{64} observable numbers
- maintain a counter per value (again, an unsigned long)

Counting Numbers

1. c(x) = 0 **for each** x $\in \{0, \ldots, 2^{64}\}$
2. i = 1
3. **while** true:
4. c(S(i))++ // S represents 'the stream of numbers'
5. i++

Counting Numbers

- Let's assume, we observe an infinite sequence of numbers
- **unsigned 'long'** values, i.e., 2^{64} observable numbers
- maintain a counter per value (again, an unsigned long)

2^{64} rows

number	value
0	8 byte
1	8 byte
2	8 byte
3	8 byte
4	8 byte
5	8 byte
…	…

Counting Numbers

- Let's assume, we observe an infinite sequence of numbers
- **unsigned 'long'** values, i.e., 2^{64} observable numbers
- maintain a counter per value (again, an unsigned long)

147,573,952,589,676,412,928 bytes

2^{64} rows

0	8 byte
1	8 byte
2	8 byte
3	8 byte
4	8 byte
5	8 byte
…	…

Counting Numbers

- Let's assume, we observe an infinite sequence of numbers
- **unsigned 'long'** values, i.e., 2^{64} observable numbers
- maintain a counter per value (again, an unsigned long)

147,573,952,589,676,412,928 bytes

144,115,188,075,855,872 KB

2^{64} rows

2	8 byte
3	8 byte
4	8 byte
5	8 byte
…	…

Chair of Process and Data Science

Counting Numbers

- Let’s assume, we observe an infinite sequence of numbers
- **unsigned ‘long’** values, i.e., 2^{64} observable numbers
- maintain a counter per value (again, an unsigned long)

147,573,952,589,676,412,928 bytes

144,115,188,075,855,872 KB

140,737,488,355,328 MB

3	8 byte
4	8 byte
5	8 byte
…	…

Counting Numbers

- Let's assume, we observe an infinite sequence of numbers
- **unsigned 'long'** values, i.e., 2^{64} observable numbers
- maintain a counter per value (again, an unsigned long)

147,573,952,589,676,412,928 bytes

144,115,188,075,855,872 KB

140,737,488,355,328 MB

137,438,953,472 GB

5	8 byte
…	…

Counting Numbers

- Let’s assume, we observe an infinite sequence of numbers
- **unsigned ‘long’** values, i.e., 2^{64} observable numbers
- maintain a counter per value (again, an unsigned long)

147,573,952,589,676,412,928 bytes

144,115,188,075,855,872 KB

140,737,488,355,328 MB

137,438,953,472 GB

134,217,728 TB

…	…

Counting Numbers

- Let's assume, we observe an infini
- **unsigned 'long'** values, i.e., 2
- maintain a counter per va   ong)

147,573,952,5   bytes

14   5,872 KB

8,355,328 MB

7,438,953,472 GB

134,217,728 TB

...	...

Can we use (significantly) less memory?

Counting Numbers

- Let’s assume, we observe an infinite sequence of numbers
- **unsigned ‘long’** values, i.e., 2^{64} observable numbers
- Let $h : \mathbb{N} \rightarrow \{0, \ldots, n\}$ here n is much smaller than $|\mathbb{N}|$
  - i.e. **hash** function

Counting Numbers

- Let's assume, we observe an infinite sequence of numbers
- **unsigned 'long'** values, i.e., 2^{64} observable numbers
- Let $h: \mathbb{N} \rightarrow \{0, \ldots, n\}$ here n is much smaller than $|\mathbb{N}|$

- Consider n = 4, and h(i) = i mod 5

Counting Numbers

- Let's assume, we observe an infinite sequence of numbers
- **unsigned 'long'** values, i.e., 2^{64} observable numbers
- Let $h: \mathbb{N} \rightarrow \{0, \ldots, n\}$ here n is much smaller than $|\mathbb{N}|$

- Consider n = 4, and h(i) = i mod 5

- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, …

0	1	2	3	4
0,5,10,15,…	1,6,11,16,…	2,7,12,17,…	3,8,13,18,…	4,9,14,19,…

Counting Numbers

- Let’s assume, we observe an infinite sequence of numbers
- **unsigned ‘long’** values, i.e., 2^{64} observable numbers
- Let $h : \mathbb{N} \rightarrow \{0, \ldots, n\}$ here n is much smaller than $|\mathbb{N}|$

- Take *k* hash functions of size *n,* i.e., $h_1, h_2, \ldots, h_k$

Counting Numbers

hash ranges

hash functions

	1	2	3	4	5	6	7	8	…	n
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
…	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0	0	0

Counting Numbers

- Receive 1st number: 5

hash ranges →

hash functions ↓

	1	2	3	4	5	6	7	8	…	n
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
…	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0	0	0

Counting Numbers

- Receive 1st number: 5
- Calculate $h_1(5)$

hash ranges

hash functions

	1	2	3	4	5	6	7	8	…	n
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
…	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0	0	0

Counting Numbers

- Receive 1st number: 5
- Calculate $h_1(5) = 4$

hash ranges

hash functions

	1	2	3	4	5	6	7	8	…	n
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
…	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0	0	0

Counting Numbers

- Receive 1st number: 5
- Calculate $h_1(5) = 4$

hash ranges

hash functions

	1	2	3	4	5	6	7	8	…	n
1	0	0	0	1	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
…	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0	0	0

Counting Numbers

- Receive 1st number: 5
- Calculate $h_2(5) = 2$

hash ranges

hash functions

	1	2	3	4	5	6	7	8	…	n
1	0	0	0	1	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
…	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0	0	0

Counting Numbers

- Receive 1st number: 5
- Calculate $h_2(5) = 2$

hash ranges

hash functions

	1	2	3	4	5	6	7	8	…	n
1	0	0	0	**1**	0	0	0	0	0	0
2	0	**1**	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
…	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0	0	0

Counting Numbers

- Receive 1st number: 5
- Calculate $h_3(5) = 7$

hash ranges →

hash functions ↓

	1	2	3	4	5	6	7	8	…	n
1	0	0	0	1	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	1	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
…	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0	0	0

Counting Numbers

- Receive 1st number: 5
- Calculate the hash value of each hash function

hash ranges →

hash functions ↓

	1	2	3	4	5	6	7	8	…	n
1	0	0	0	**1**	0	0	0	0	0	0
2	0	**1**	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	**1**	0	0	0
4	0	0	**1**	0	0	0	0	0	0	0
5	0	0	0	**1**	0	0	0	0	0	0
…	0	0	0	0	0	0	0	0	**1**	0
k	0	0	0	0	0	0	0	**1**	0	0

Counting Numbers

- Receive 2nd number: 37
- Calculate the hash value of each hash function

hash ranges

hash functions

	1	2	3	4	5	6	7	8	…	n
1	**1**	0	0	1	0	0	0	0	0	0
2	0	1	**1**	0	0	0	0	0	0	0
3	0	0	0	0	0	**1**	1	0	0	0
4	0	0	1	**1**	0	0	0	0	0	0
5	0	0	0	1	0	0	**1**	0	0	0
…	0	**1**	0	0	0	0	0	0	1	0
k	**1**	0	0	0	0	0	0	1	0	0

Counting Numbers

- Receive 3^{d} number: 1,337
- Calculate the hash value of each hash function

hash ranges

hash functions

	1	2	3	4	5	6	7	8	…	n
1	1	**1**	0	1	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0	**1**
3	0	0	0	0	**1**	1	1	0	0	0
4	0	0	**2**	1	0	0	0	0	0	0
5	0	**1**	0	1	0	0	1	0	0	0
…	0	1	**1**	0	0	0	0	0	1	0
k	1	**1**	0	0	0	0	0	1	0	0

Streaming Data
Counting Numbers

- Receive 3^{d} number: 1,337
- Calculate the hash value of each hash function

hash ranges →

hash functions ↓

		2	3	4	5	6	7	8	…	n
1	1	1	0	1	0	0	0	0	0	0
2	0		1							1
3	0	0	0							0
4	0	0	2	1	0	0	0	0	0	0
5	0	1	0	1	0	0	1	0	0	0
…	0	1	1	0	0	0	0	0	1	0
k	1	1	0	0	0	0	0	1	0	0

hash collision: both 5 and 1,337 map to hash value 2 for hash function 4

Counting Numbers

- Query: How many times did we observe number 5?
- Calculate the hash value of each hash function

hash ranges

hash functions

	1	2	3	4	5	6	7	8	…	n
1	1	1	0	1	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0	1
3	0	0	0	0	1	1	1	0	0	0
4	0	0	2	1	0	0	0	0	0	0
5	0	1	0	1	0	0	1	0	0	0
…	0	1	1	0	0	0	0	0	1	0
k	1	1	0	0	0	0	0	1	0	0

Counting Numbers

- Query: How many times did we observe number 5?
- Calculate the hash value of each hash function

hash ranges

hash functions

	1	2	3	4	5	6	7	8	…	n
1	1	1	0	1	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0	1
3	0	0	0	0	1	1	1	0	0	0
4	0	0	2	1	0	0	0	0	0	0
5	0	1	0	1	0	0	1	0	0	0
…	0	1	1	0	0	0	0	0	1	0
k	1	1	0	0	0	0	0	1	0	0

Counting Numbers

- Query: How many times did we observe r
- Calculate the hash value of each ha

What value to pick?

hash ranges →

hash functions ↓

	1	2	3	[illegible]	[illegible]	[illegible]	7	8	…	n
1	1	1	0	[illegible]	[illegible]	0	0	0	0	0
2	0	**1**	[illegible]	[illegible]	0	0	0	0	0	1
3	0	[illegible]	[illegible]	0	1	1	**1**	0	0	0
4	[illegible]	[illegible]	**2**	1	0	0	0	0	0	0
5	0	1	0	**1**	0	0	1	0	0	0
…	0	1	1	0	0	0	0	0	**1**	0
k	1	1	0	0	0	0	0	**1**	0	0

Counting Numbers

- Query: How many times did we observe number 5?
- Calculate the hash value of each hash function

- Every value in each cell can be seen as follows:

number of occurrences of 5	number of occurrences of others

Counting Numbers

- Query: How many times did we observe number 5?
- Calculate the hash value of each hash function

- Every value in each cell can be seen as follows:

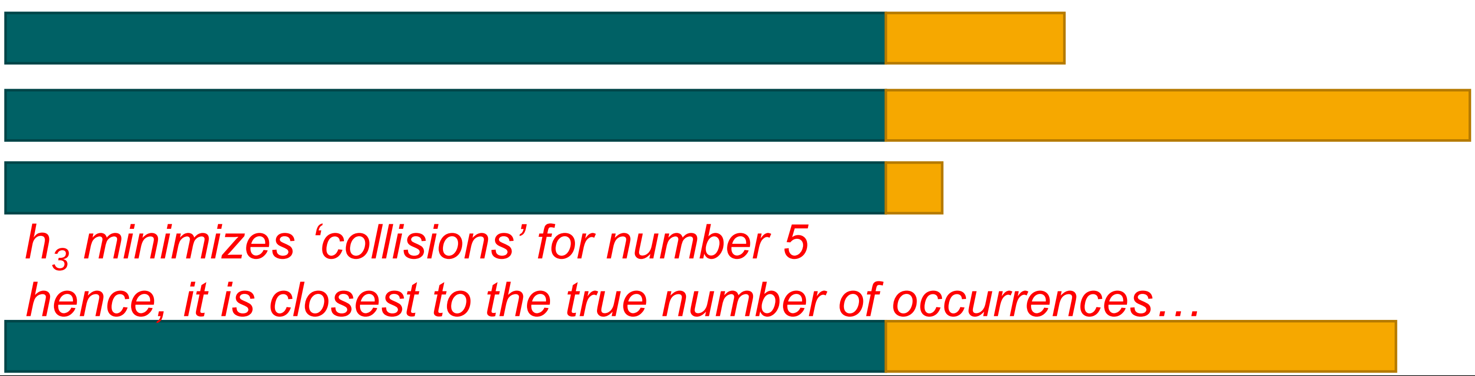
- Looking at all cells for value 5, we see:
- h_1
- h_2
- h_3
- …
- h_k

Counting Numbers

- Query: How many times did we observe number 5?
- Calculate the hash value of each hash function

- Every value in each cell can be seen as follows:

- Looking at all cells for value 5, we see:
- h_1
- h_2
- h_3
- …
- h_k



h_3 minimizes 'collisions' for number 5
hence, it is closest to the true number of occurrences…

Counting Numbers

- *Count-Min-Sketch:*
- Memory: O(kn)

Counting Numbers

- *Count-Min-Sketch:*
- Memory: O(kn)

- Let N denote the number of elements seen so far
- Let $\epsilon, \delta \in [0,1]$
- If we let: k = $\lceil \ln\left(\frac{1}{\delta}\right) \rceil$ and n = $\lceil \frac{e}{\epsilon} \rceil$ then:
- $\widehat{x_i} \leq x_i + \epsilon N$ with probability $1 - \delta$

Counting Numbers

- Let's assume, we observe an infinite sequence of numbers
- What are the *k* most frequent numbers observed?

Counting Numbers

- Let’s assume, we observe an infinite sequence of numbers
- What are the *k* most frequent numbers observed?

Algorithm 3.1: FREQUENT(k)

$$
\begin{array}{l}
n \leftarrow 0; T \leftarrow \emptyset;\\
\textbf{for each } i \textbf{ :}\\
\quad \textbf{do} \begin{cases}
n \leftarrow n + 1;\\
\textbf{if } i \in T\\
\quad \textbf{then } c_i \leftarrow c_i + 1;\\
\quad \textbf{else if } |T| < k - 1\\
\quad \textbf{then} \begin{cases} T \leftarrow T \cup \{i\};\\ c_i \leftarrow 1; \end{cases}\\
\quad \textbf{else for all } j \in T\\
\quad \textbf{do} \begin{cases} c_j \leftarrow c_j - 1;\\ \textbf{if } c_j = 0\\ \quad \textbf{then } T \leftarrow T \backslash \{j\}; \end{cases}
\end{cases}
\end{array}
$$

Counting Numbers

- Let’s assume, we observe an infinite sequence of numbers
- What are the *k* most frequent numbers observed?
- Observe: 5

num	val
5	1

Algorithm 3.1: FREQUENT(k)

$n \leftarrow 0; T \leftarrow \emptyset;$
for each i **:**

$$\textbf{do} \begin{cases} n \leftarrow n+1; \\ \textbf{if } i \in T \\ \quad \textbf{then } c_i \leftarrow c_i + 1; \\ \quad \textbf{else if } |T| < k-1 \\ \quad \textbf{then } \begin{cases} T \leftarrow T \cup \{i\}; \\ c_i \leftarrow 1; \end{cases} \\ \quad \textbf{else for all } j \in T \\ \quad \textbf{do } \begin{cases} c_j \leftarrow c_j - 1; \\ \textbf{if } c_j = 0 \\ \quad \textbf{then } T \leftarrow T \backslash \{j\}; \end{cases} \end{cases}$$

Counting Numbers

- Let's assume, we observe an infinite sequence of numbers
- What are the *k* most frequent numbers observed?
- Observe: 13

num	val
5	1
13	1

Algorithm 3.1: FREQUENT(k)

$n \leftarrow 0; T \leftarrow \emptyset;$
for each i **:**

$$\textbf{do} \begin{cases} n \leftarrow n+1; \\ \textbf{if } i \in T \\ \quad \textbf{then } c_i \leftarrow c_i + 1; \\ \quad \textbf{else if } |T| < k-1 \\ \quad \textbf{then } \begin{cases} T \leftarrow T \cup \{i\}; \\ c_i \leftarrow 1; \end{cases} \\ \quad \textbf{else for all } j \in T \\ \quad \textbf{do } \begin{cases} c_j \leftarrow c_j - 1; \\ \textbf{if } c_j = 0 \\ \quad \textbf{then } T \leftarrow T \backslash \{j\}; \end{cases} \end{cases}$$

Counting Numbers

- Let's assume, we observe an infinite sequence of numbers
- What are the *k* most frequent numbers observed?
- Observe: 1337

num	val
5	1
13	1
1337	1

Algorithm 3.1: FREQUENT(k)

$$
\begin{array}{l}
n \leftarrow 0; T \leftarrow \emptyset; \\
\textbf{for each } i: \\
\quad \textbf{do} \begin{cases}
n \leftarrow n + 1; \\
\textbf{if } i \in T \\
\quad \textbf{then } c_i \leftarrow c_i + 1; \\
\quad \textbf{else if } |T| < k - 1 \\
\quad \textbf{then } \begin{cases} T \leftarrow T \cup \{i\}; \\ c_i \leftarrow 1; \end{cases} \\
\quad \textbf{else for all } j \in T \\
\quad \textbf{do } \begin{cases} c_j \leftarrow c_j - 1; \\ \textbf{if } c_j = 0 \\ \quad \textbf{then } T \leftarrow T \backslash \{j\}; \end{cases}
\end{cases}
\end{array}
$$

Counting Numbers

- Let's assume, we observe an infinite sequence of numbers
- What are the *k* most frequent numbers observed?
- Observe: …

num	val
5	1
13	1
1337	1
…	…
…	…
…	…

Algorithm 3.1: FREQUENT(k)

$n \leftarrow 0; T \leftarrow \emptyset;$
for each i **:**
do $\left\{\begin{array}{l} n \leftarrow n+1; \\ \textbf{if } i \in T \\ \quad \textbf{then } c_i \leftarrow c_i + 1; \\ \quad \textbf{else if } |T| < k-1 \\ \quad \textbf{then } \left\{\begin{array}{l} T \leftarrow T \cup \{i\}; \\ c_i \leftarrow 1; \end{array}\right. \\ \quad \textbf{else for all } j \in T \\ \quad \textbf{do } \left\{\begin{array}{l} c_j \leftarrow c_j - 1; \\ \textbf{if } c_j = 0 \\ \quad \textbf{then } T \leftarrow T \backslash \{j\}; \end{array}\right. \end{array}\right.$

Counting Numbers

- Let’s assume, we observe an infinite sequence of numbers
- What are the *k* most frequent numbers observed?
- Observe: 8

num	val
5	2
13	5
1337	1
…	…
…	…
…	…
8	1

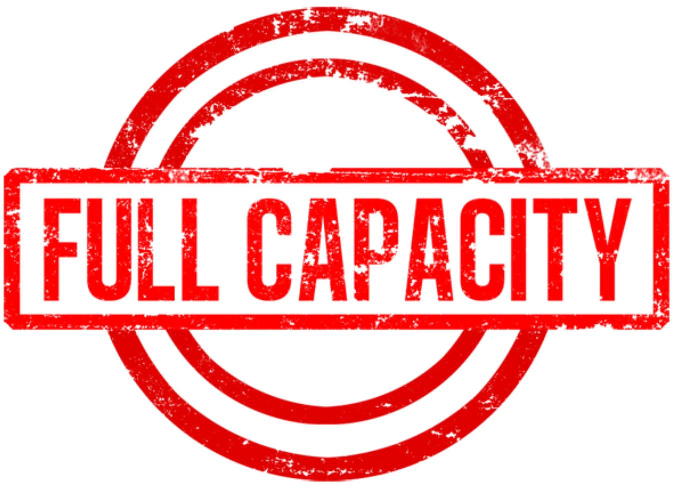
Algorithm 3.1: FREQUENT(k)

$$
\begin{array}{l}
n \leftarrow 0; T \leftarrow \emptyset;\\
\textbf{for each } i:\\
\textbf{do} \begin{cases}
n \leftarrow n+1;\\
\textbf{if } i \in T\\
\quad \textbf{then } c_i \leftarrow c_i + 1;\\
\quad \textbf{else if } |T| < k-1\\
\quad \textbf{then } \begin{cases} T \leftarrow T \cup \{i\};\\ c_i \leftarrow 1; \end{cases}\\
\quad \textbf{else for all } j \in T\\
\quad \textbf{do } \begin{cases} c_j \leftarrow c_j - 1;\\ \textbf{if } c_j = 0\\ \quad \textbf{then } T \leftarrow T \backslash \{j\}; \end{cases}
\end{cases}
\end{array}
$$

Counting Numbers

- Let's assume, we observe an infinite sequence of numbers
- What are the *k* most frequent numbers observed?
- Observe: 175

num	val
5	2
13	5
1337	1
…	…
…	…
…	…
8	1



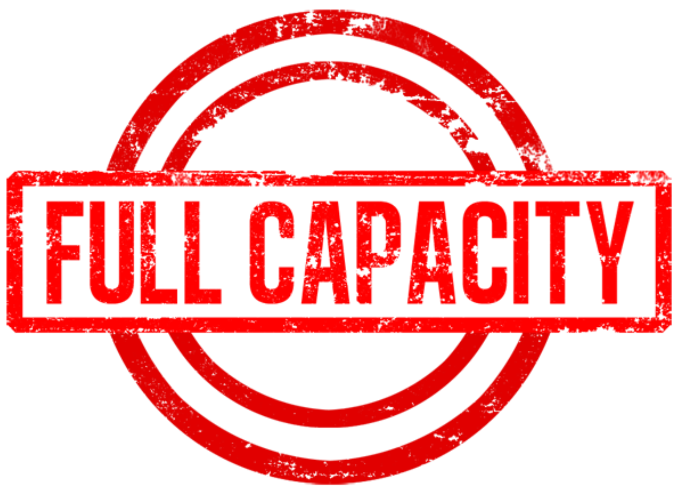
Algorithm 3.1: FREQUENT(k)

$$
\begin{array}{l}
n \leftarrow 0; T \leftarrow \emptyset; \\
\textbf{for each } i : \\
\quad \textbf{do} \begin{cases}
n \leftarrow n + 1; \\
\textbf{if } i \in T \\
\quad \textbf{then } c_i \leftarrow c_i + 1; \\
\quad \textbf{else if } |T| < k - 1 \\
\quad \textbf{then} \begin{cases} T \leftarrow T \cup \{i\}; \\ c_i \leftarrow 1; \end{cases} \\
\quad \textbf{else for all } j \in T \\
\quad \textbf{do} \begin{cases} c_j \leftarrow c_j - 1; \\ \textbf{if } c_j = 0 \\ \quad \textbf{then } T \leftarrow T \backslash \{j\}; \end{cases}
\end{cases}
\end{array}
$$

Counting Numbers

- Let’s assume, we observe an infinite sequence of numbers
- What are the *k* most frequent numbers observed?
- Observe: 175

num	val
5	**1**
13	**4**
1337	**0**
…	…
…	…
…	…
8	**0**



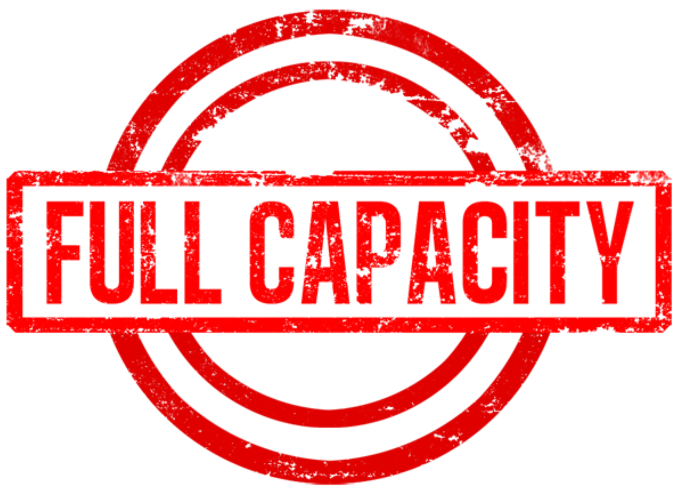
Algorithm 3.1: FREQUENT(k)

$$
\begin{array}{l}
n \leftarrow 0; T \leftarrow \emptyset; \\
\textbf{for each } i: \\
\quad \textbf{do} \begin{cases}
n \leftarrow n + 1; \\
\textbf{if } i \in T \\
\quad \textbf{then } c_i \leftarrow c_i + 1; \\
\quad \textbf{else if } |T| < k - 1 \\
\quad \textbf{then} \begin{cases} T \leftarrow T \cup \{i\}; \\ c_i \leftarrow 1; \end{cases} \\
\quad \textbf{else for all } j \in T \\
\quad \textbf{do} \begin{cases} c_j \leftarrow c_j - 1; \\ \textbf{if } c_j = 0 \\ \quad \textbf{then } T \leftarrow T \backslash \{j\}; \end{cases}
\end{cases}
\end{array}
$$

Counting Numbers

- Let's assume, we observe an infinite sequence of numbers
- What are the *k* most frequent numbers observed?
- Observe: 175

num	val
5	1
13	4
~~1337~~	~~0~~
…	…
…	…
…	…
~~8~~	~~0~~



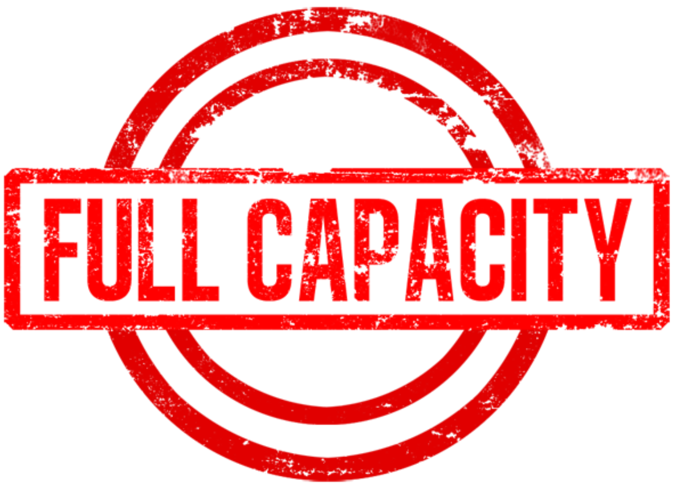
Algorithm 3.1: FREQUENT(k)

$n \leftarrow 0; T \leftarrow \emptyset;$
for each i **:**
do $\left\{\begin{array}{l} n \leftarrow n+1; \\ \textbf{if } i \in T \\ \quad \textbf{then } c_i \leftarrow c_i + 1; \\ \quad \textbf{else if } |T| < k-1 \\ \quad \textbf{then } \left\{\begin{array}{l} T \leftarrow T \cup \{i\}; \\ c_i \leftarrow 1; \end{array}\right. \\ \quad \textbf{else for all } j \in T \\ \quad \textbf{do } \left\{\begin{array}{l} c_j \leftarrow c_j - 1; \\ \textbf{if } c_j = 0 \\ \quad \textbf{then } T \leftarrow T \backslash \{j\}; \end{array}\right. \end{array}\right.$

Counting Numbers

- Let's assume, we observe an infinite sequence of numbers
- What are the *k* most frequent numbers observed?
- Observe: 175

num	val
5	1
13	4
…	…
…	…
…	…



Algorithm 3.1: FREQUENT(k)

$$
\begin{array}{l}
n \leftarrow 0; T \leftarrow \emptyset; \\
\textbf{for each } i : \\
\quad \textbf{do} \begin{cases}
n \leftarrow n + 1; \\
\textbf{if } i \in T \\
\quad \textbf{then } c_i \leftarrow c_i + 1; \\
\quad \textbf{else if } |T| < k - 1 \\
\quad \textbf{then} \begin{cases} T \leftarrow T \cup \{i\}; \\ c_i \leftarrow 1; \end{cases} \\
\quad \textbf{else for all } j \in T \\
\quad \textbf{do} \begin{cases} c_j \leftarrow c_j - 1; \\ \textbf{if } c_j = 0 \\ \quad \textbf{then } T \leftarrow T \backslash \{j\}; \end{cases}
\end{cases}
\end{array}
$$

Counting Numbers

- Let's assume, we observe an infinite sequence of numbers
- What are the *k* most frequent numbers observed?

Algorithm 3.1: FREQUENT(k)

$$
\begin{array}{l}
n \leftarrow 0; T \leftarrow \emptyset; \\
\textbf{for each } i : \\
\quad \textbf{do} \begin{cases}
n \leftarrow n + 1; \\
\textbf{if } i \in T \\
\quad \textbf{then } c_i \leftarrow c_i + 1; \\
\quad \textbf{else if } |T| < k - 1 \\
\quad \textbf{then } \begin{cases} T \leftarrow T \cup \{i\}; \\ c_i \leftarrow 1; \end{cases} \\
\quad \textbf{else for all } j \in T \\
\quad \textbf{do } \begin{cases} c_j \leftarrow c_j - 1; \\ \textbf{if } c_j = 0 \\ \quad \textbf{then } T \leftarrow T \backslash \{j\}; \end{cases}
\end{cases}
\end{array}
$$

Algorithm 3.3: SPACESAVING(k)

$$
\begin{array}{l}
n \leftarrow 0; \\
T \leftarrow \emptyset; \\
\textbf{for each } i : \\
\quad \textbf{do} \begin{cases}
n \leftarrow n + 1; \\
\textbf{if } i \in T \\
\quad \textbf{then } c_i \leftarrow c_i + 1; \\
\quad \textbf{else if } |T| < k \\
\quad \textbf{then } \begin{cases} T \leftarrow T \cup \{i\}; \\ c_i \leftarrow 1; \end{cases} \\
\quad \textbf{else } \begin{cases} j \leftarrow \arg\min_{j \in T} c_j; \\ c_i \leftarrow c_j + 1; \\ T \leftarrow T \cup \{i\} \backslash \{j\}; \end{cases}
\end{cases}
\end{array}
$$

Counting Numbers

- Let's assume, we observe an infinite sequence of numbers
- What are the *k* most frequent numbers observed?
- Observe: 8

num	val
5	2
13	5
1337	1
…	…
…	…
…	…
8	1

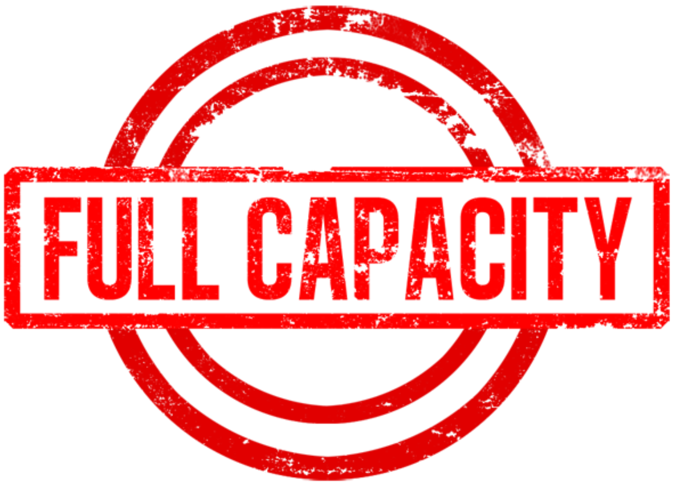
Algorithm 3.3: SPACESAVING(k)

$$
\begin{array}{l}
n \leftarrow 0; \\
T \leftarrow \emptyset; \\
\textbf{for each } i: \\
\quad \textbf{do} \begin{cases}
n \leftarrow n + 1; \\
\textbf{if } i \in T \\
\quad \textbf{then } c_i \leftarrow c_i + 1; \\
\quad \textbf{else if } |T| < k \\
\quad \textbf{then } \begin{cases} T \leftarrow T \cup \{i\}; \\ c_i \leftarrow 1; \end{cases} \\
\quad \textbf{else } \begin{cases} j \leftarrow \arg\min_{j \in T} c_j; \\ c_i \leftarrow c_j + 1; \\ T \leftarrow T \cup \{i\} \backslash \{j\}; \end{cases}
\end{cases}
\end{array}
$$

Counting Numbers

- Let's assume, we observe an infinite sequence of numbers
- What are the *k* most frequent numbers observed?
- Observe: 175

num	val
5	2
13	5
1337	1
…	…
…	…
…	…
8	1



Algorithm 3.3: SPACESAVING(k)

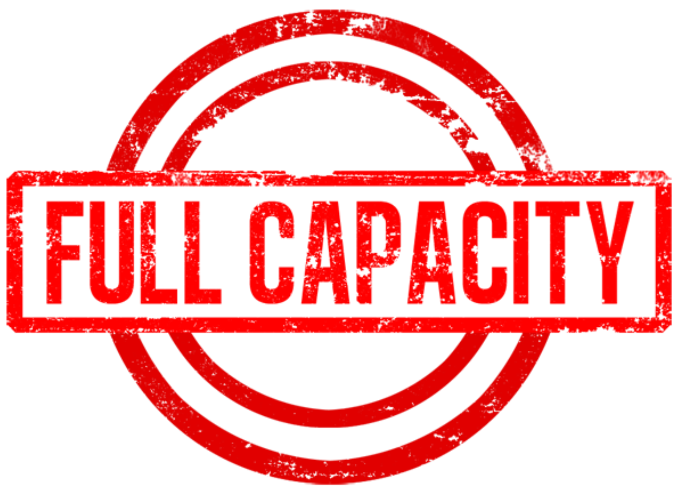
$n \leftarrow 0;$
$T \leftarrow \emptyset;$
for each i **:**

$$\textbf{do} \begin{cases} n \leftarrow n+1; \\ \textbf{if } i \in T \\ \quad \textbf{then } c_i \leftarrow c_i + 1; \\ \quad \textbf{else if } |T| < k \\ \quad \textbf{then } \begin{cases} T \leftarrow T \cup \{i\}; \\ c_i \leftarrow 1; \end{cases} \\ \quad \textbf{else } \begin{cases} j \leftarrow \arg\min_{j \in T} c_j; \\ c_i \leftarrow c_j + 1; \\ T \leftarrow T \cup \{i\} \setminus \{j\}; \end{cases} \end{cases}$$

Counting Numbers

- Let’s assume, we observe an infinite sequence of numbers
- What are the *k* most frequent numbers observed?
- Observe: 175

num	val
5	2
13	5
1337	**1**
…	…
…	…
…	…
8	1



Algorithm 3.3: SpaceSaving(k)

$n \leftarrow 0;$
$T \leftarrow \emptyset;$
for each i **:**

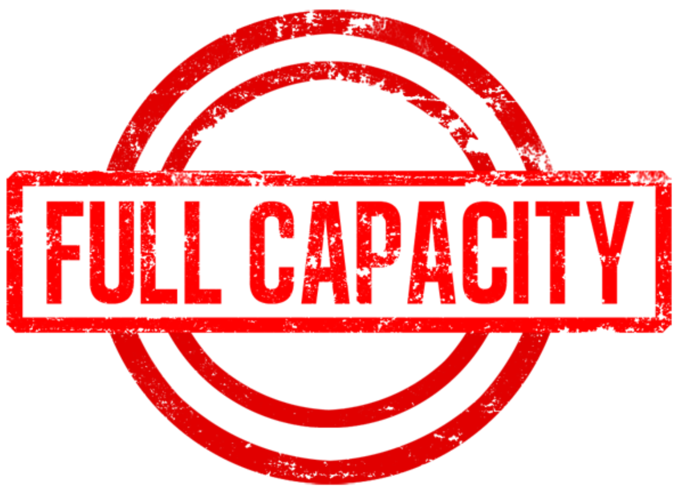
$$
\textbf{do} \begin{cases} n \leftarrow n+1; \\ \textbf{if } i \in T \\ \quad \textbf{then } c_i \leftarrow c_i + 1; \\ \quad \textbf{else if } |T| < k \\ \quad \textbf{then } \begin{cases} T \leftarrow T \cup \{i\}; \\ c_i \leftarrow 1; \end{cases} \\ \quad \textbf{else } \begin{cases} j \leftarrow \arg\min_{j \in T} c_j; \\ c_i \leftarrow c_j + 1; \\ T \leftarrow T \cup \{i\} \backslash \{j\}; \end{cases} \end{cases}
$$

Chair of Process and Data Science

Counting Numbers

- Let’s assume, we observe an infinite sequence of numbers
- What are the *k* most frequent numbers observed?
- Observe: 175

num	val
5	2
13	5
175	2
…	…
…	…
…	…
8	1



Algorithm 3.3: SpaceSaving(k)

$n \leftarrow 0;$
$T \leftarrow \emptyset;$
for each i **:**

$$\textbf{do} \begin{cases} n \leftarrow n + 1; \\ \textbf{if } i \in T \\ \quad \textbf{then } c_i \leftarrow c_i + 1; \\ \quad \textbf{else if } |T| < k \\ \quad \textbf{then } \begin{cases} T \leftarrow T \cup \{i\}; \\ c_i \leftarrow 1; \end{cases} \\ \quad \textbf{else } \begin{cases} j \leftarrow \arg\min_{j \in T} c_j; \\ c_i \leftarrow c_j + 1; \\ T \leftarrow T \cup \{i\} \backslash \{j\}; \end{cases} \end{cases}$$

Counting Numbers

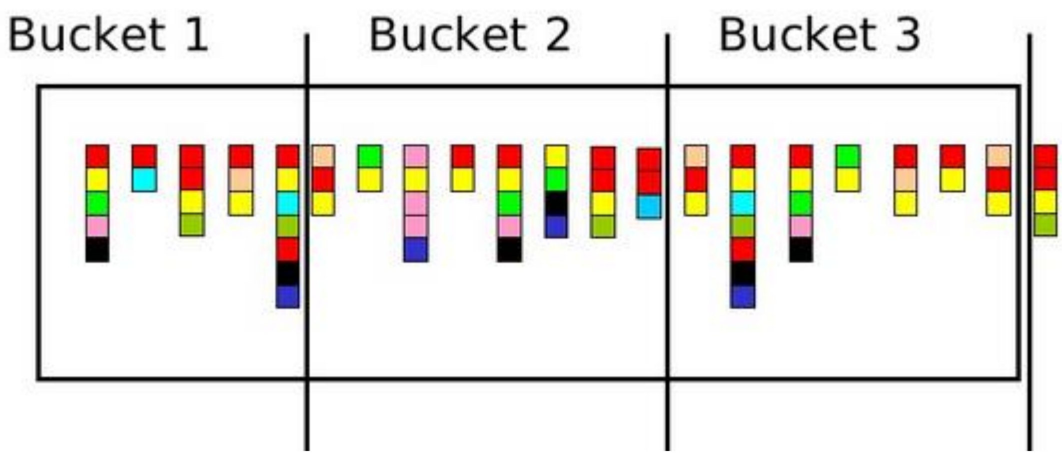
- Let's assume, we observe an infinite sequence of numbers
- What are the *k* most frequent numbers observed?

Algorithm 3.2: LossyCounting(k)

$$
\begin{array}{l}
n \leftarrow 0; \Delta \leftarrow 0; T \leftarrow \emptyset; \\
\textbf{for each } i : \\
\quad \textbf{do} \begin{cases}
n \leftarrow n + 1; \\
\textbf{if } i \in T \\
\quad \textbf{then } c_i \leftarrow c_i + 1; \\
\quad \textbf{else } \begin{cases} T \leftarrow T \cup \{i\}; \\ c_j \leftarrow 1 + \Delta; \end{cases} \\
\textbf{if } \lfloor \frac{n}{k} \rfloor \neq \Delta \\
\quad \textbf{then } \begin{cases} \Delta \leftarrow n/k; \\ \textbf{for all } j \in T \\ \quad \textbf{do if } c_j < \Delta \\ \quad \textbf{then } T \leftarrow T \backslash \{j\} \end{cases}
\end{cases}
\end{array}
$$

Counting Numbers

- Let’s assume, we observe an infinite sequence of numbers
- What are the *k* most frequent numbers observed?



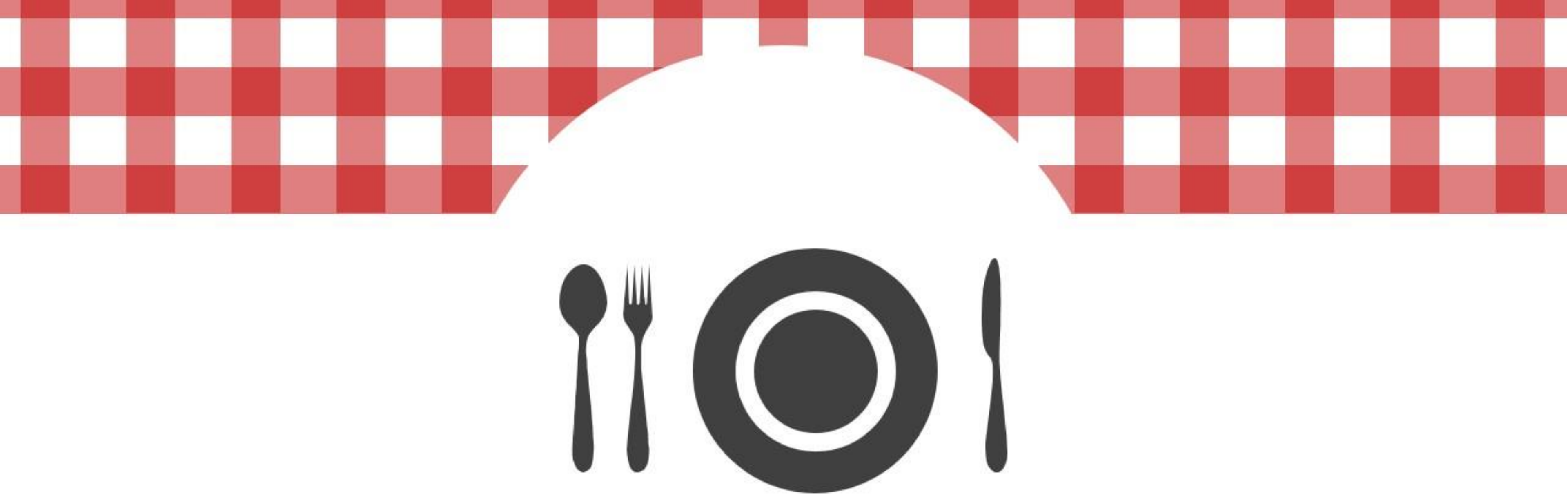
General Idea/Requirements

- Let's assume, we observe an infinite sequence of objects

- Any answer we can provide, typically, is an approximate answer.

General Idea/Requirements

- Let's assume, we observe an infinite sequence of objects
- Any answer we can provide, typically, is an approximate answer.
- Typically:
  - Result is 1 +/- ϵ from true value
  - With probability 1 – δ
- Memory: (for example) $O(\frac{1}{\epsilon^2}\log\left(\frac{1}{\delta}\right))$
- Ideally: $O(\log^k(N))$ (polylog in number of observed items)



- Motivation
- Streaming Data (Theory)
- Stream Based Discovery
- Online Alignments

https://giffiles.alphacoders.com/133/133737.gif

Stream-Based Process Discovery

- Let’s assume, we observe an infinite sequence of events

- Events describe:
    - Case-Identifier
    - Activity
    - Payload

Stream-Based Process Discovery

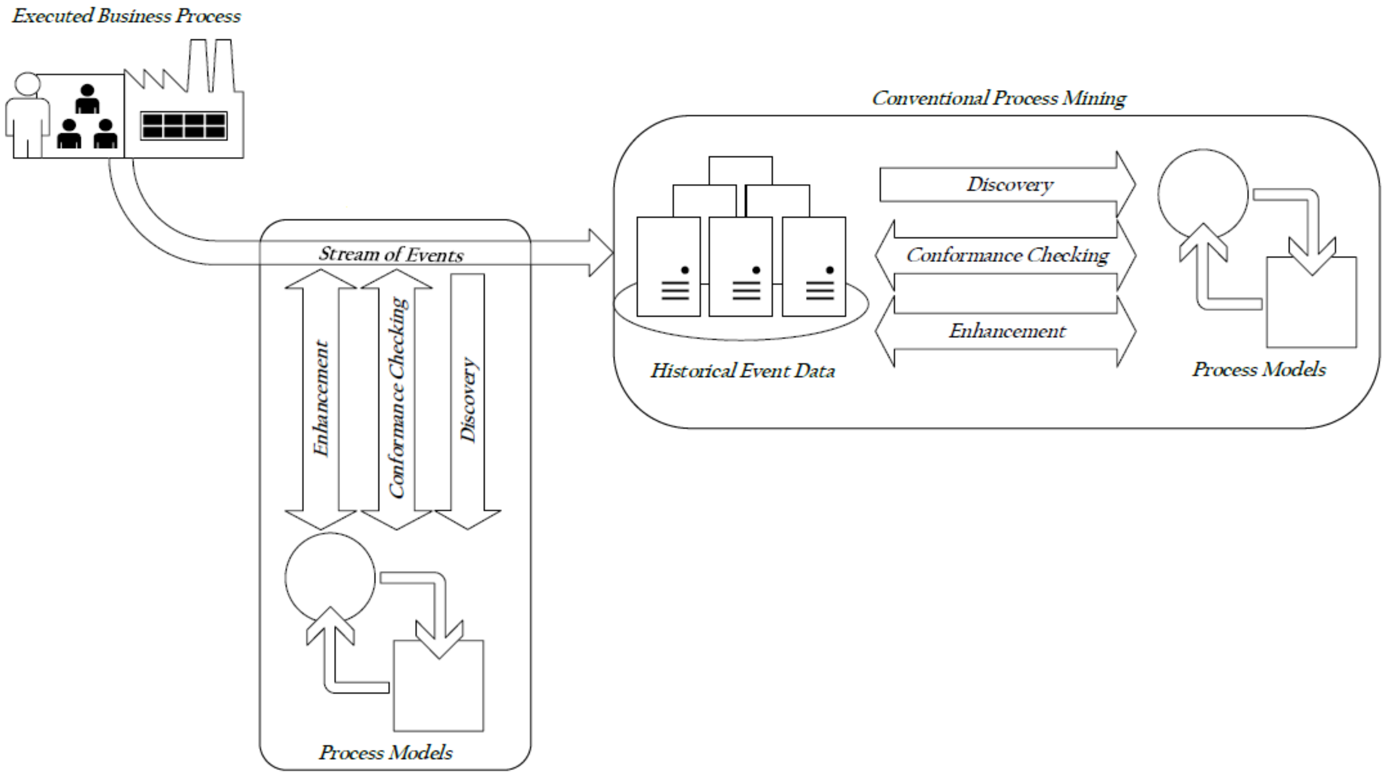
- Let’s assume, we observe an infinite sequence of events

- Events describe:
  - Case-Identifier
  - Activity
  - Payload

- e=(c,a,…)

Stream-Based Process Discovery

- Let’s assume, we observe an infinite sequence of events



Stream-Based Process Discovery

- Let’s assume, we observe an infinite sequence of events

- Typical ‘streaming’ analysis:

- How many times does an event occur?

- What are the most frequent events?

Stream-Based Process Discovery

- Let’s assume, we observe an infinite sequence of events

- Typical ‘streaming’ analysis:

- How many times does an event occur?
  - Once, every event is unique

- What are the most frequent events?
  - Every event is equally frequent, every event is unique

Stream-Based Process Discovery

- Let’s assume, we observe an infinite sequence of events

- Typical ‘streaming’ analysis:

- How many times does a case occur?

- What are the most frequent cases?

Stream-Based Process Discovery

- Let's assume, we observe an infinite sequence of events

- Typical 'streaming' analysis:

- How many times does a case occur?
  - Trivial adoption of standard streaming algorithms…

- What are the most frequent cases?
  - Trivial adoption of standard streaming algorithms…

Stream-Based Process Discovery

- Let's assume, we observe an infinite sequence of events

- Typical 'streaming' analysis:

- How many

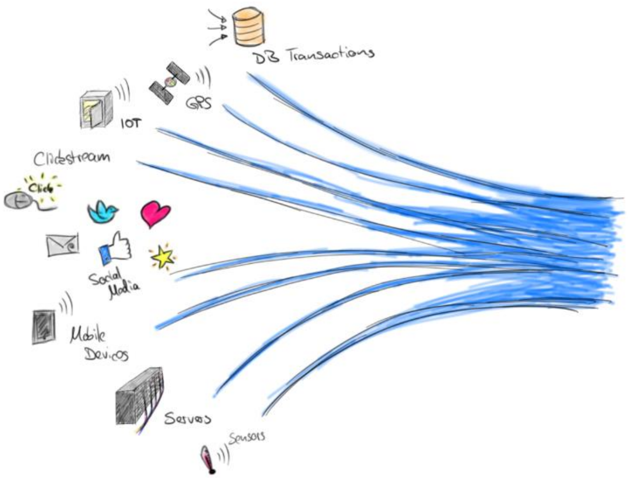
  - Trivial adop

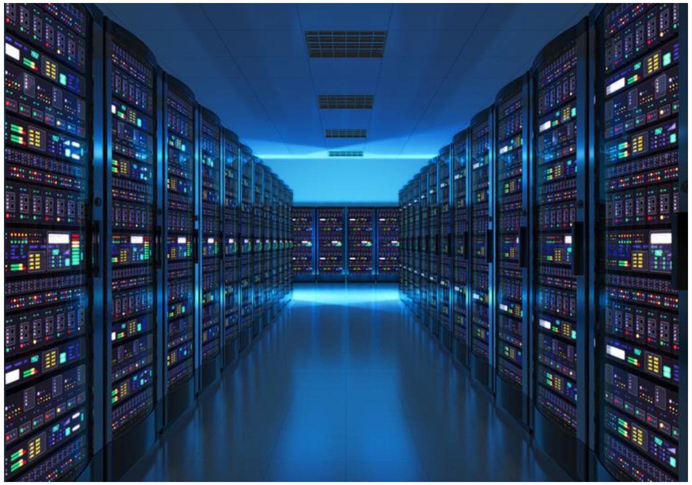
Nice statistics, but, it does not give us any process models!

- What are the most frequent cases?
  - Trivial adoption of standard streaming algorithms…

Stream-Based Process Discovery

- Let’s assume, we observe an infinite sequence of events

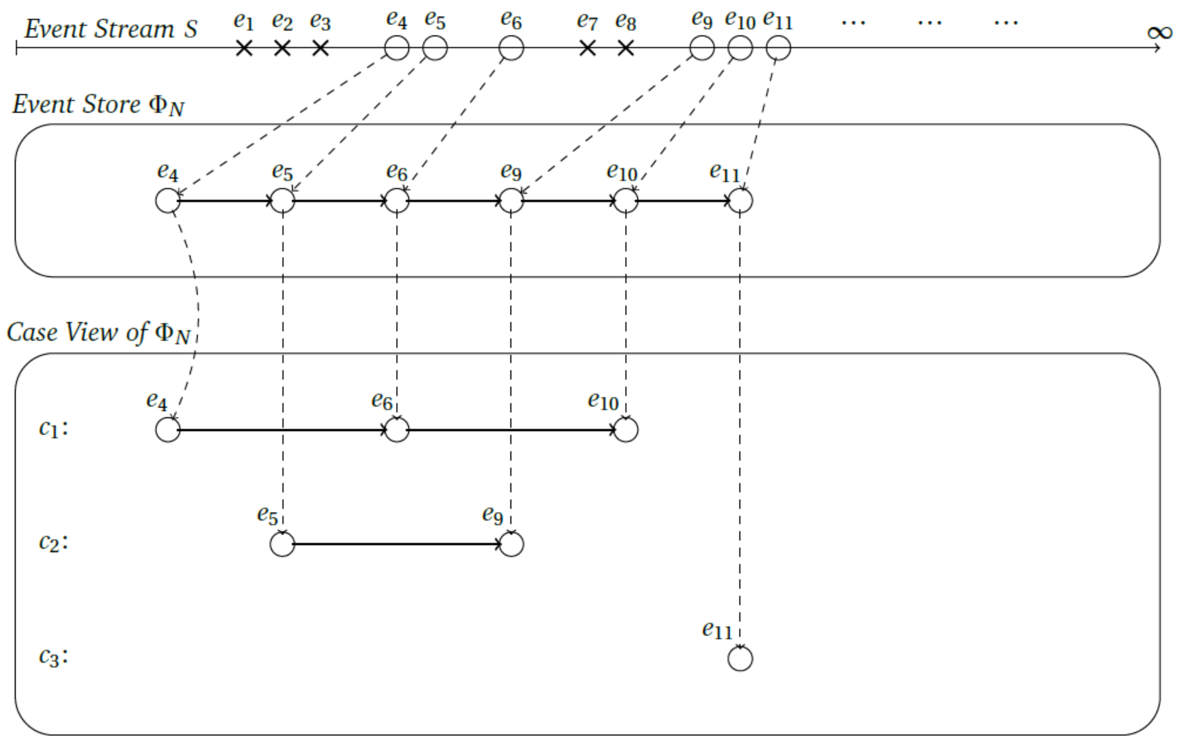





- We need smart techniques to store events, temporarily, and (from time to time) discover a model

Stream-Based Process Discovery

- Let’s assume, we observe an infinite sequence of events



Stream-Based Process Discovery

- Let's assume, we observe an infinite sequence of events
- Using a "sliding window"

Stream-Based Process Discovery

- Let’s assume, we observe an infinite sequence of events
- Using a “sliding window”

Stream-Based Process Discovery

- Let's assume, we observe an infinite sequence of events
- Using a "sliding window"

Stream-Based Process Discovery

- Let's assume, we observe an infinite sequence of events

1. a ← array of length k
2. i, j← 0
3. **while** true:
4.   a[i **mod** k] ← S(i)
5.   i++
6.   **if** i >= k:
7.   j ← (j+1) **mod** k

Stream-Based Process Discovery

- Let's assume, we observe an infinite sequence of events

1. a ← array of length k
2. i, j ← 0
3. **while** true:
4. a[i **mod** k] ← S(i)
5. i++
6. **if** i >= k:
7. j ← (j+1) **mod** k

Element on stream

Chair of Process and Data Science

RWTH AACHEN UNIVERSITY

Stream-Based Process Discovery

- Let’s assume, we observe an infinite sequence of events

```
a ← array of length k
i, j ← 0
while true:
    a[i mod k ] ← S(i)
    i++
    if i >= k:
        j ← (j+1) mod k
```

Index of ‘oldest’ item

Stream-Based Process Discovery

- Let’s assume, we observe an infinite sequence of events

```
a ← array of length k
i, j← 0
while true:
    a[i mod k ] ← S(i)
    i++
    if i >= k:
        j ← (j+1) mod k
```

- a[j] = oldest element

Stream-Based Process Discovery

- Let's assume, we observe an infinite sequence of events

1. a ← array of length k
2. i, j← 0
3. **while** true:
4.   a[i **mod** k] ← S(i)
5.   i++
6.   **if** i >= k:
7.   j ← (j+1) **mod** k

- a[j-1] = newest element (or |a|-1 if j=0)

Stream-Based Process Discovery

- Let’s assume, we observe an infinite sequence of events

```
a ← array of length k
i, j← 0
while true:
    a[i mod k ] ← S(i)
    i++
    if i >= k:
        j ← (j+1) mod k
```

- [a[j],a[j+1],…,a[|a|-1],a[0],…,a[j-1]] = window

Stream-Based Process Discovery

- Let's assume, we observe an infinite sequence of events

- Store events in a sliding window
- (projection onto cases is easy)
  - Either when we need it, or, continuously…

Stream-Based Process Discovery

- Let’s assume, we observe an infinite sequence of events
- Store events in a sliding window
- (projection onto cases is easy)
  - Either when we need it, or, continuously…
- Give to any algorithm of choice

Stream-Based Process Discovery

- Let’s assume, we observe an infinite sequence of events

- Store events in a sliding window
- (projection onto cases is easy)
  - Either when we need it, or, continuously…

- Let’s focus on Inductive Miner (Alpha / Heuristics…)

Stream-Based Process Discovery

- Let's assume, we observe an infinite sequence of events

- Store events in a sliding window
- (projection onto cases is easy)
  - Either when we need it, or, continuously…

- Let's focus on Inductive Miner (Alpha / Heuristics…)
- a>b, b>c, …

Stream-Based Process Discovery

- Idea:
  - For each case, store the last received activity

Stream-Based Process Discovery

- Idea:
  - For each case, store the last received activity
  - Upon receiving a new event, assess that activity, deduce dfg and update last received activity

Stream-Based Process Discovery

- $e_1 = (c_1, a)$

case	act
c_1	a

pair	count

Stream-Based Process Discovery

- $e_2 = (c_2, a)$

case	act
c_1	a
c_2	a

pair	count

Stream-Based Process Discovery

- $e_3 = (c_1, b)$

case	act
c_1	a
c_2	a

pair	count

Stream-Based Process Discovery

STREAM

e_1, e_2, e_3

- $e_3 = (c_1, b)$

case	act
c_1	a
c_2	a

pair	count

Stream-Based Process Discovery

- $e_3 = (c_1, b)$

case	act
c_1	b
c_2	a

pair	count
(a,b)	1

Stream-Based Process Discovery

- $e_4 = (c_1, c)$

case	act
c_1	b
c_2	a

pair	count
(a,b)	1

Stream-Based Process Discovery

- $e_4 = (c_1, c$

case	act
c_1	b
c_2	a

pair	count
(a,b)	1

Stream-Based Process Discovery

- $e_4 = (c_1, c)$

case	act
c_1	c
c_2	a

pair	count
(a,b)	1
(b,c)	1

Stream-Based Process Discovery

- $e_5 = (c_2, c)$

case	act
c_1	c
c_2	a

pair	count
(a,b)	1
(b,c)	1

Stream-Based Process Discovery

STREAM

e_1, e_2, e_3, e_4, e_5

- $e_5 = (c_2, c$

case	act
c_1	c
c_2	a

pair	count
(a,b)	1
(b,c)	1

Stream-Based Process Discovery

- $e_5 = (c_2, c)$

case	act
c_1	c
c_2	c

pair	count
(a,b)	1
(b,c)	1
(a,c)	1

Stream-Based Process Discovery

- $e_6 = (c_3, a)$

case	act
c_1	c
c_2	c
c_3	a

pair	count
(a,b)	1
(b,c)	1
(a,c)	1

Stream-Based Process Discovery

STREAM

$e_1, e_2, e_3, e_4, e_5, e_6, e_7$

- $e_7 = (c_3, b)$

case	act
c_1	c
c_2	c
c_3	b

pair	count
(a,b)	2
(b,c)	1
(a,c)	1

Stream-Based Process Discovery

STREAM

$e_1, e_2, e_3, e_4, e_5, e_6, e_7$

- $e_7 = (c_3, b)$

case	act
c_1	c
c_2	c
c_3	b

pair	count

Algorithm 3.1: FREQUENT(k)

$n \leftarrow 0; T \leftarrow \emptyset;$
for each i :

do $\begin{cases} n \leftarrow n+1; \\ \textbf{if } i \in T \\ \quad \textbf{then } c_i \leftarrow c_i + 1; \\ \quad \textbf{else if } |T| < k-1 \\ \quad \textbf{then } \begin{cases} T \leftarrow T \cup \{i\}; \\ c_i \leftarrow 1; \end{cases} \\ \quad \textbf{else for all } j \in T \\ \quad \textbf{do } \begin{cases} c_j \leftarrow c_j - 1; \\ \textbf{if } c_j = 0 \\ \quad \textbf{then } T \leftarrow T \backslash \{j\}; \end{cases} \end{cases}$

Stream-Based Process Discovery

STREAM

$e_1, e_2, e_3, e_4, e_5, e_6, e_7$

- $e_7 = (c_3, b)$

case	act
c_1	c
c_2	c
c_3	b

pair	count
(a,b)	2
(b,c)	1
(a,c)	1

Stream-Based Process Discovery

STREAM

$e_1, e_2, e_3, e_4, e_5, e_6, e_7, \ldots$

- $e_7 = (c_3, b)$

case	act	win
c_1	c	3
c_2	c	2
c_3	b	2
…	…	…
…	…	…
…	…	…
…	…	…

pair	count
(a,b)	2
(b,c)	1
(a,c)	1

Stream-Based Process Discovery

STREAM

$e_2, e_3, e_4, e_5, e_6, e_7, \ldots$

- $e_7 = (c_3, b)$

case	act	win
c_1	c	~~3~~ 2
c_2	c	2
c_3	b	2
…	…	…
…	…	…
…	…	…
…	…	…

pair	count
(a,b)	2
(b,c)	1
(a,c)	1

Stream-Based Process Discovery

STREAM

$e_2, e_3, e_4, e_5, e_6, e_7, \ldots$

- $e_7 = (c_3, b)$

case	act	win
c_1	c	~~3~~ 2
c_2	c	2
c_3	b	2
…	…	…
…	…	…
…	…	…
…	…	…

Decoupled

pair	count
(a,b)	2
(b,c)	1
(a,c)	1

Stream-Based Process Discovery

STREAM

$e_2, e_3, e_4, e_5, e_6, e_7, \ldots$

- $e_7 = (c_3, b)$

case	act	win
…	…	…
…	…	…
…	…	…

Inductive Miner (DFG Only Option)

pair	count
(a,b)	2
(b,c)	1
(a,c)	1

Stream-Based Process Discovery

STREAM

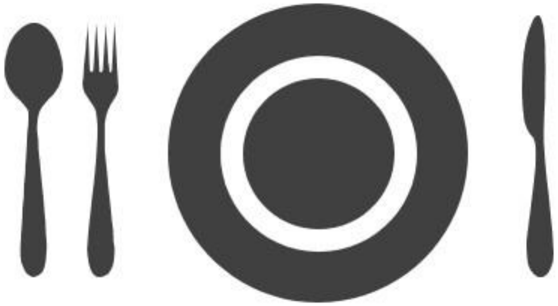
$e_2, e_3, e_4, e_5, e_6, e_7, \ldots$

- $e_7 = (c_3, b)$

case	act	win
c_1		
c_2		
c_3	b	2
…	…	…
…	…	…
…	…	…
…	…	…

Challenges?

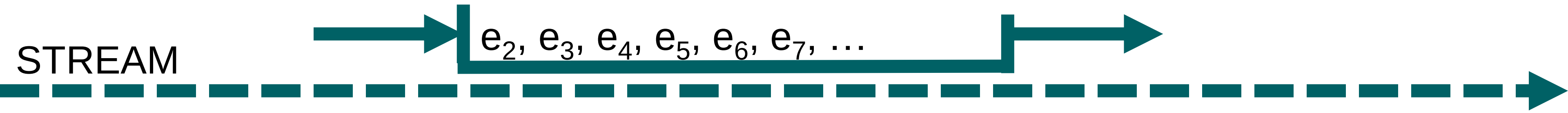
pair	count
(a,b)	2
(b,c)	1
(a,c)	1



- Motivation
- Streaming Data (Theory)
- Stream Based Discovery
- **Online Alignments**

https://giffiles.alphacoders.com/133/133737.gif

Stream-Based Conformance Checking

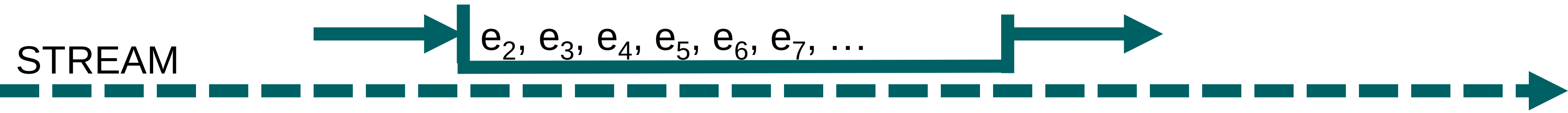


- … we don’t care how we store stuff, we assume you (smartly) store (currently active) traces:

case	act
c_1	<a,b,c>
c_2	<a,c>
c_3	<a,b>
…	…
…	…
…	…
…	…

Streaming Data

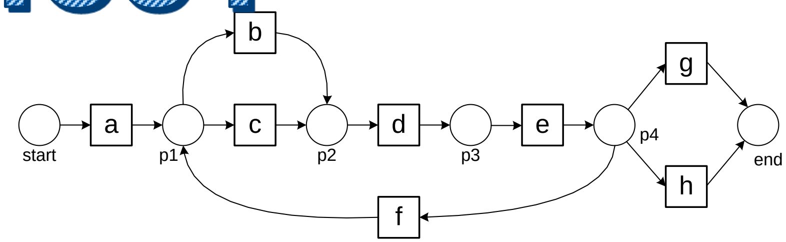
Stream-Based Conformance Checking



- … we don’t care how we store stuff, we assume you (smartly) store (currently active) traces:

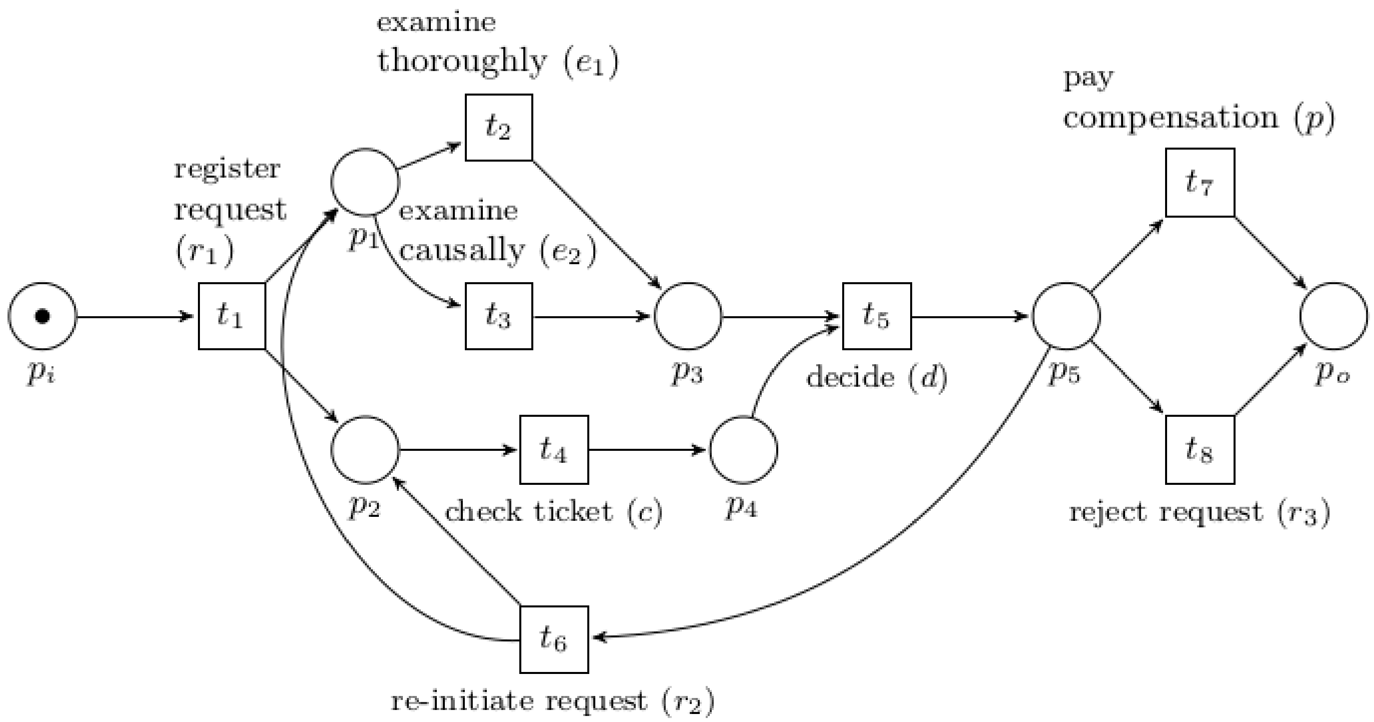
case	act
c_1	<a,b,c>
c_2	<a,c>
c_3	<a,b>
…	…
…	…
…	…
…	…

Conformance?

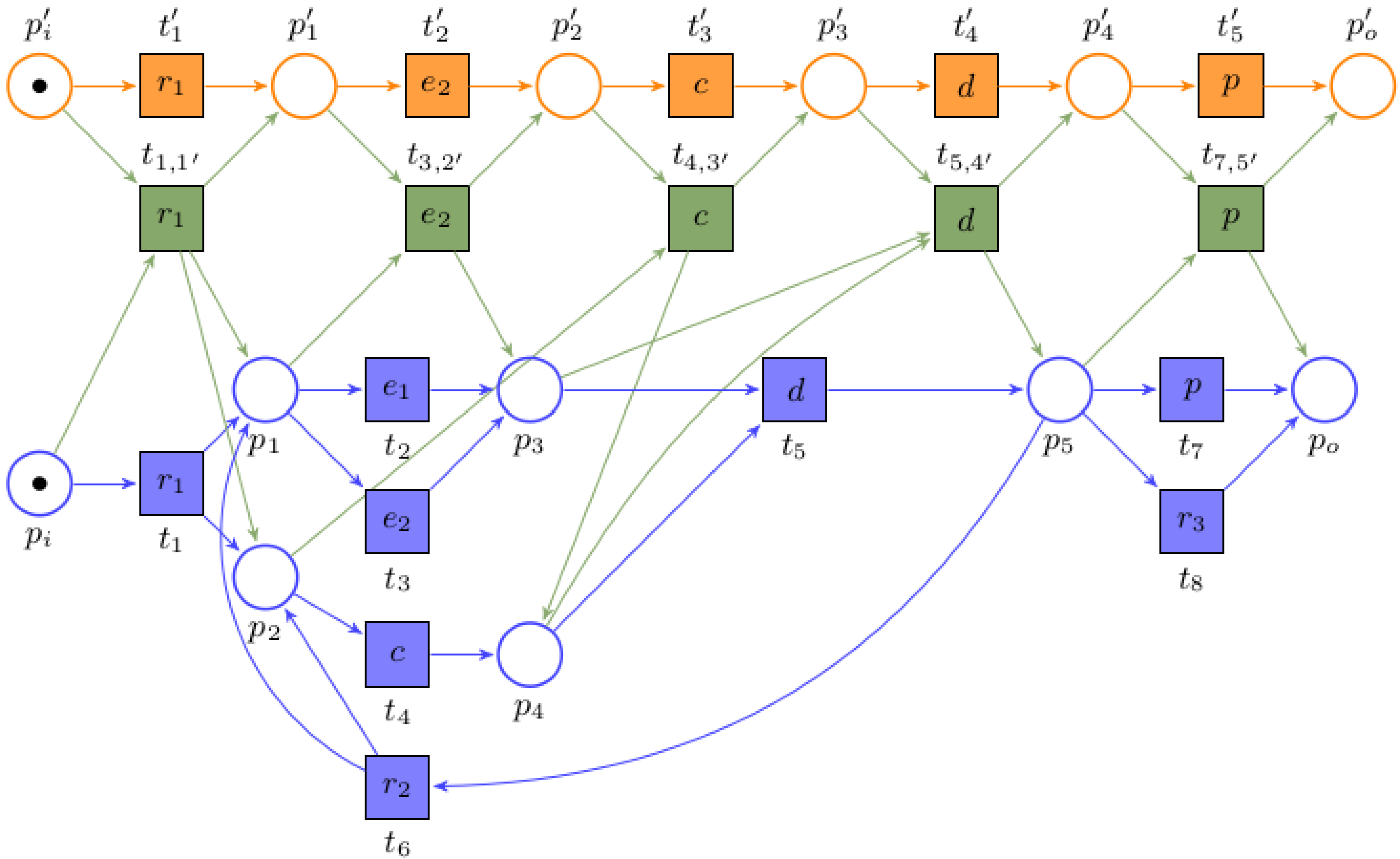


Stream-Based Conformance Checking

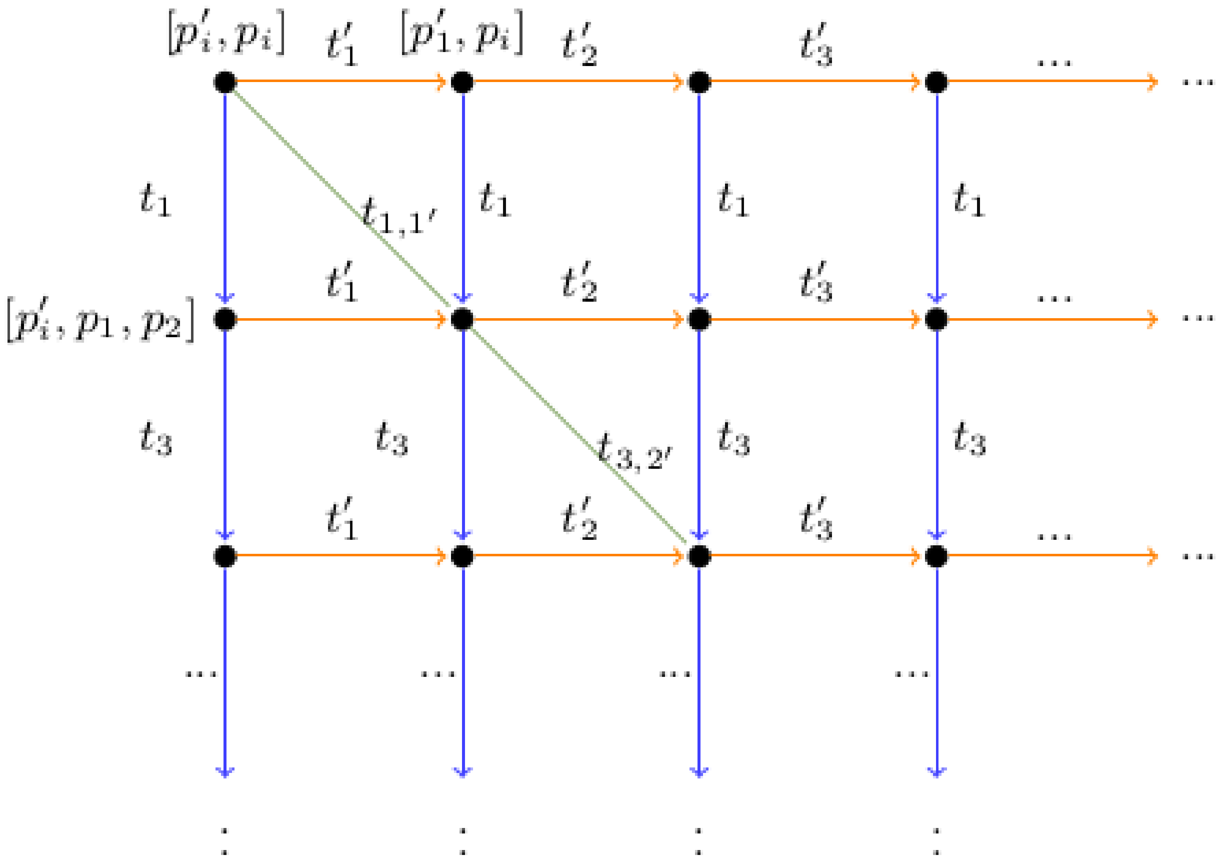
- <r1, e2, c, d, p>



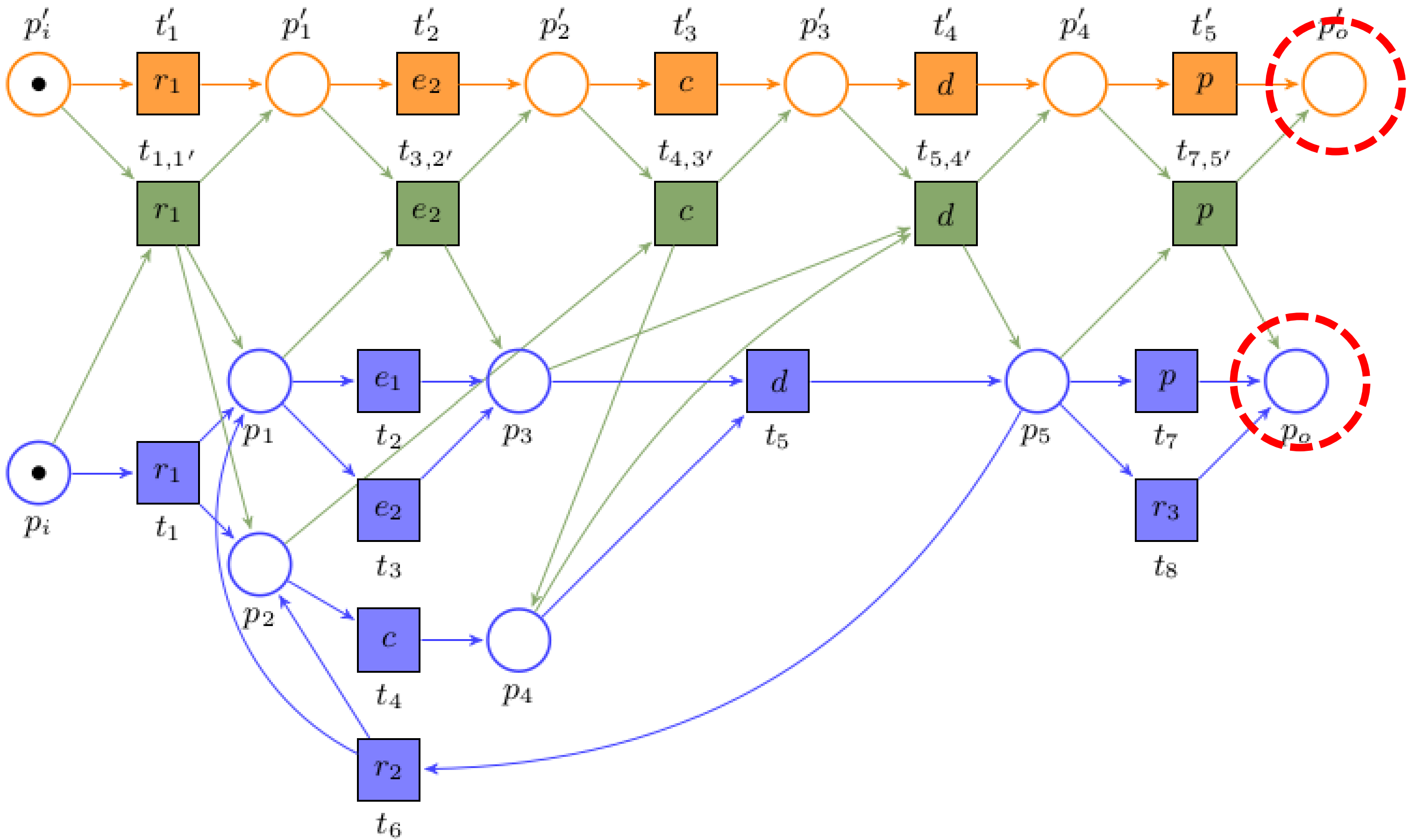
Stream-Based Conformance Checking



Stream-Based Conformance Checking

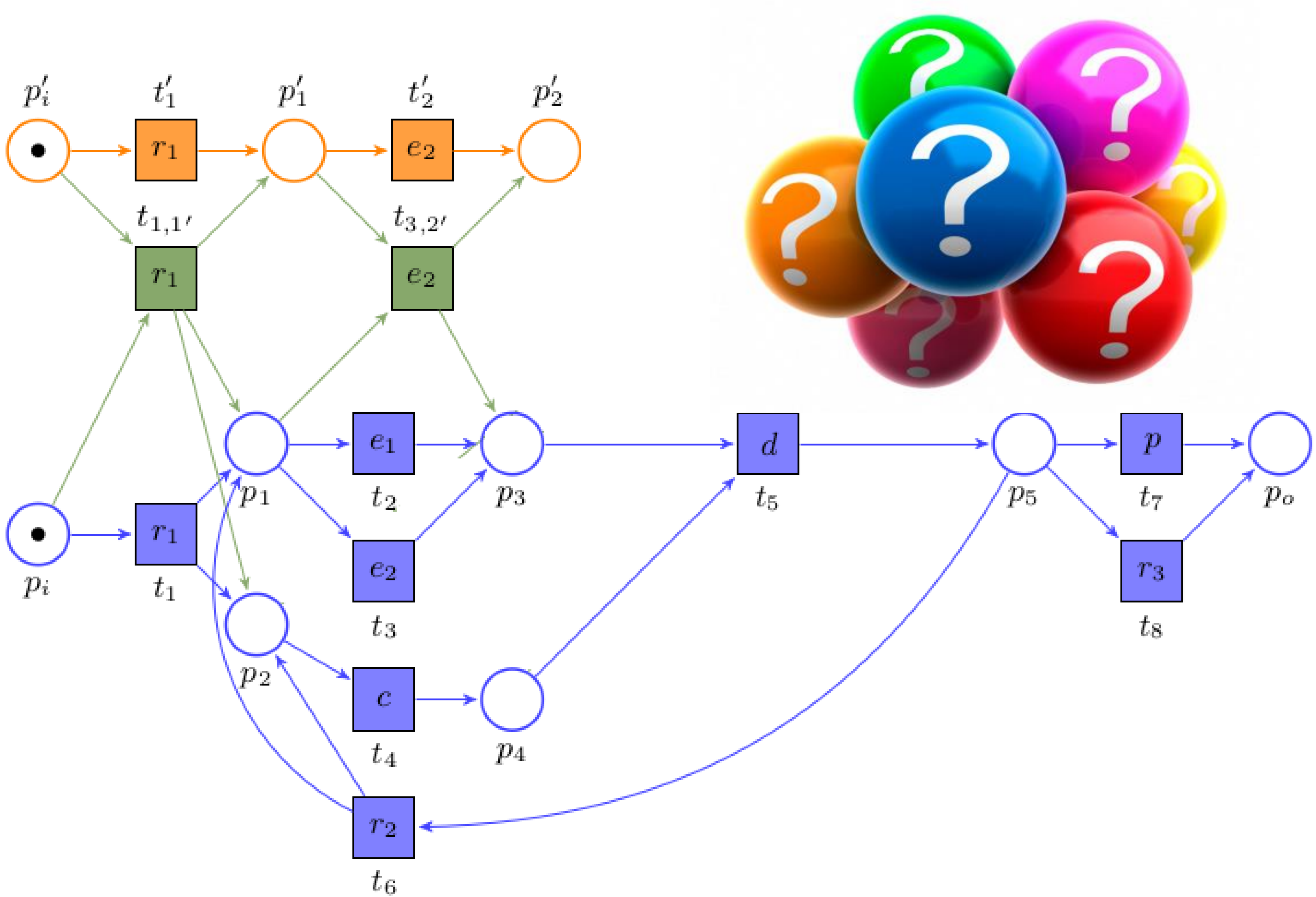


Stream-Based Conformance Checking



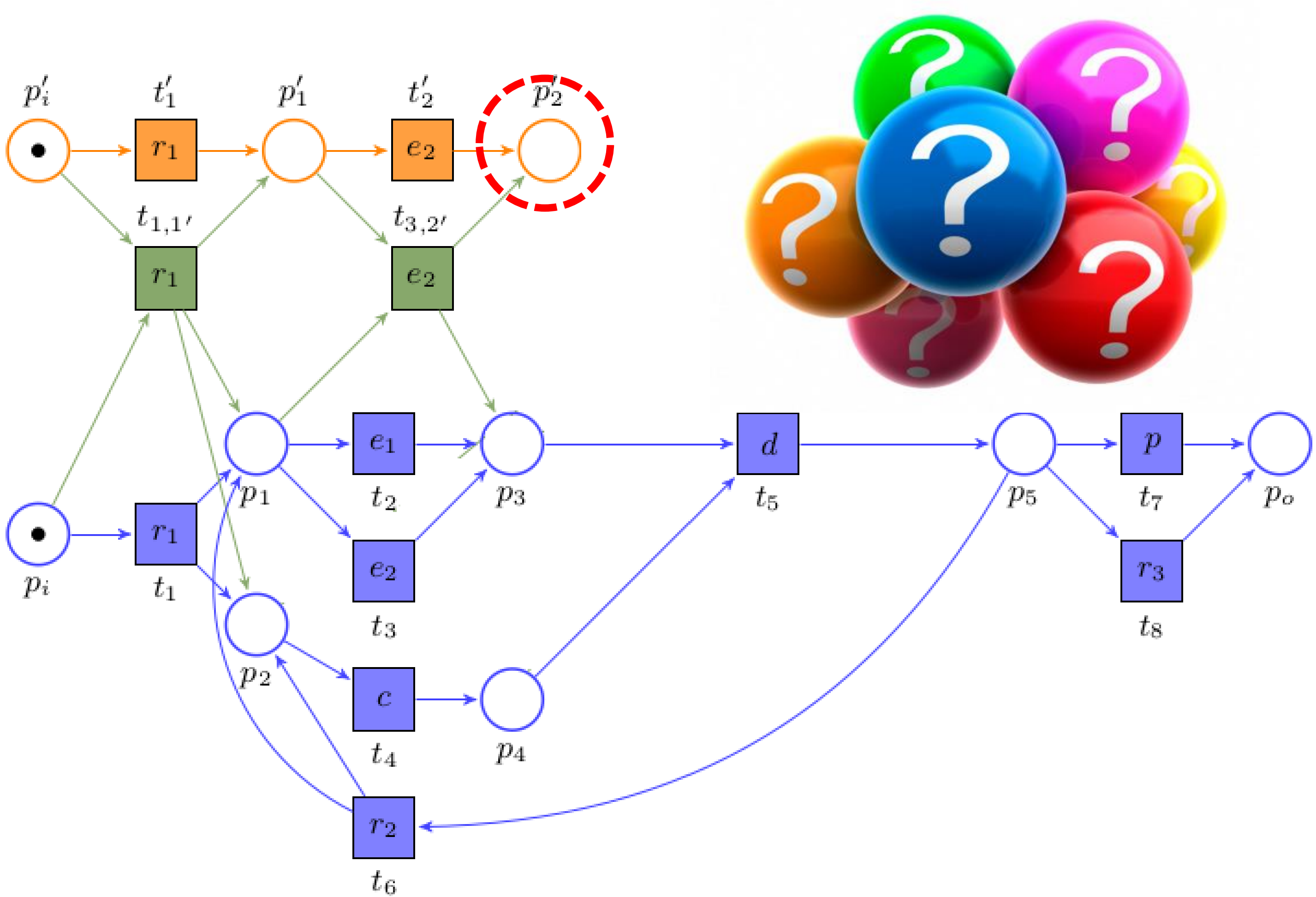
Stream-Based Conformance Checking

- <r1, e2,> → all we have seen so far



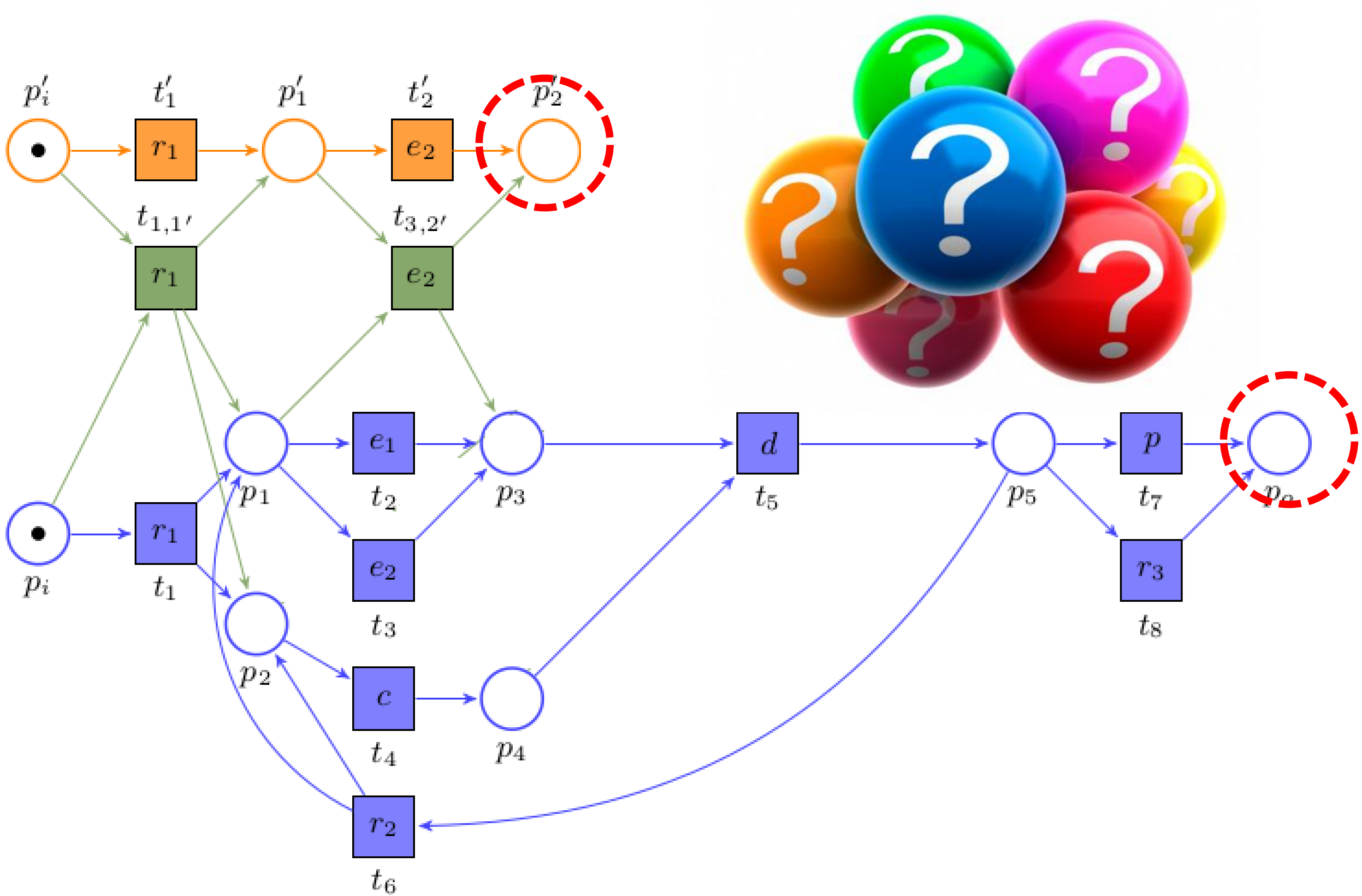
Stream-Based Conformance Checking

- <r1, e2,> → all we have seen so far



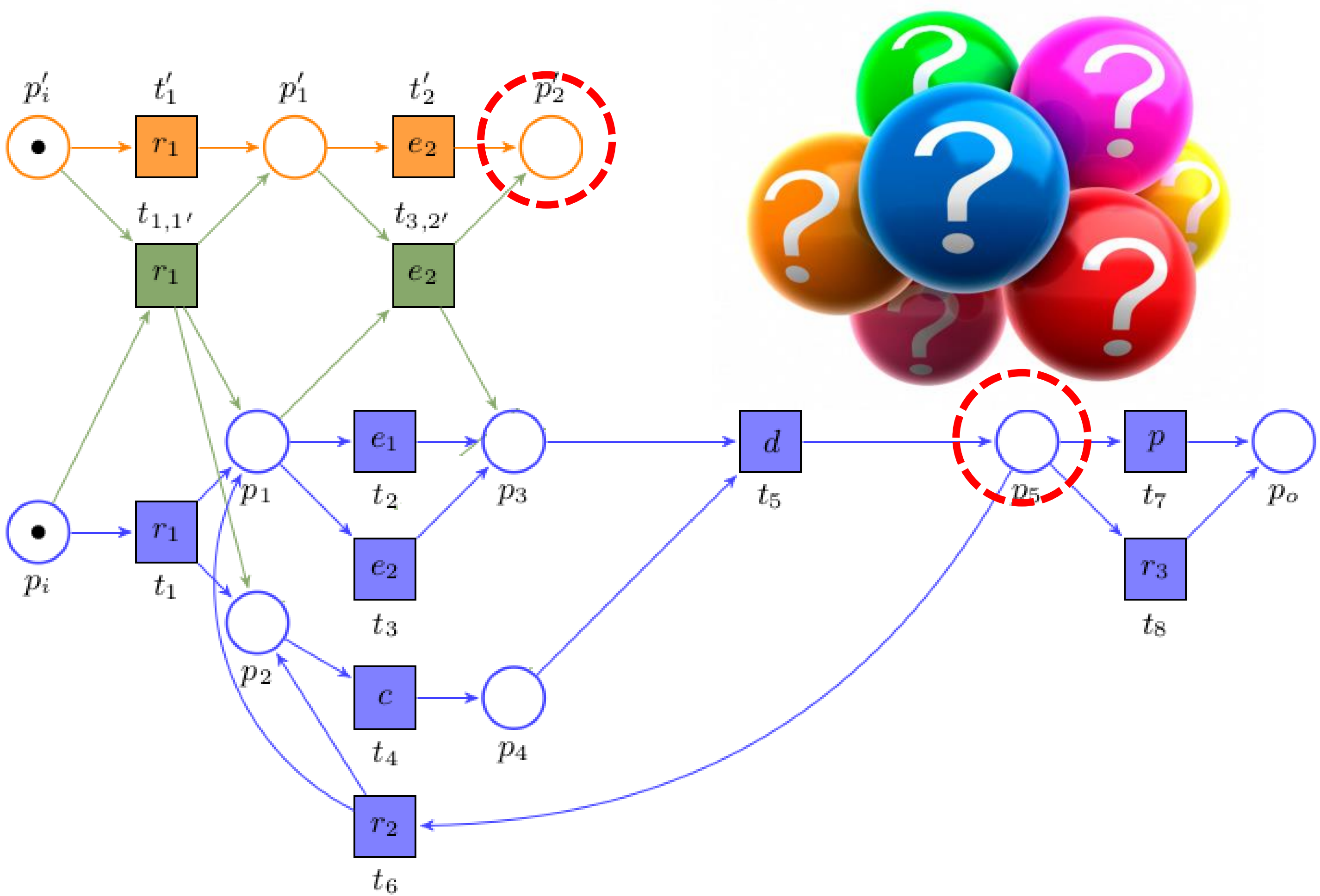
Stream-Based Conformance Checking

- <r1, e2,> → all we have seen so far



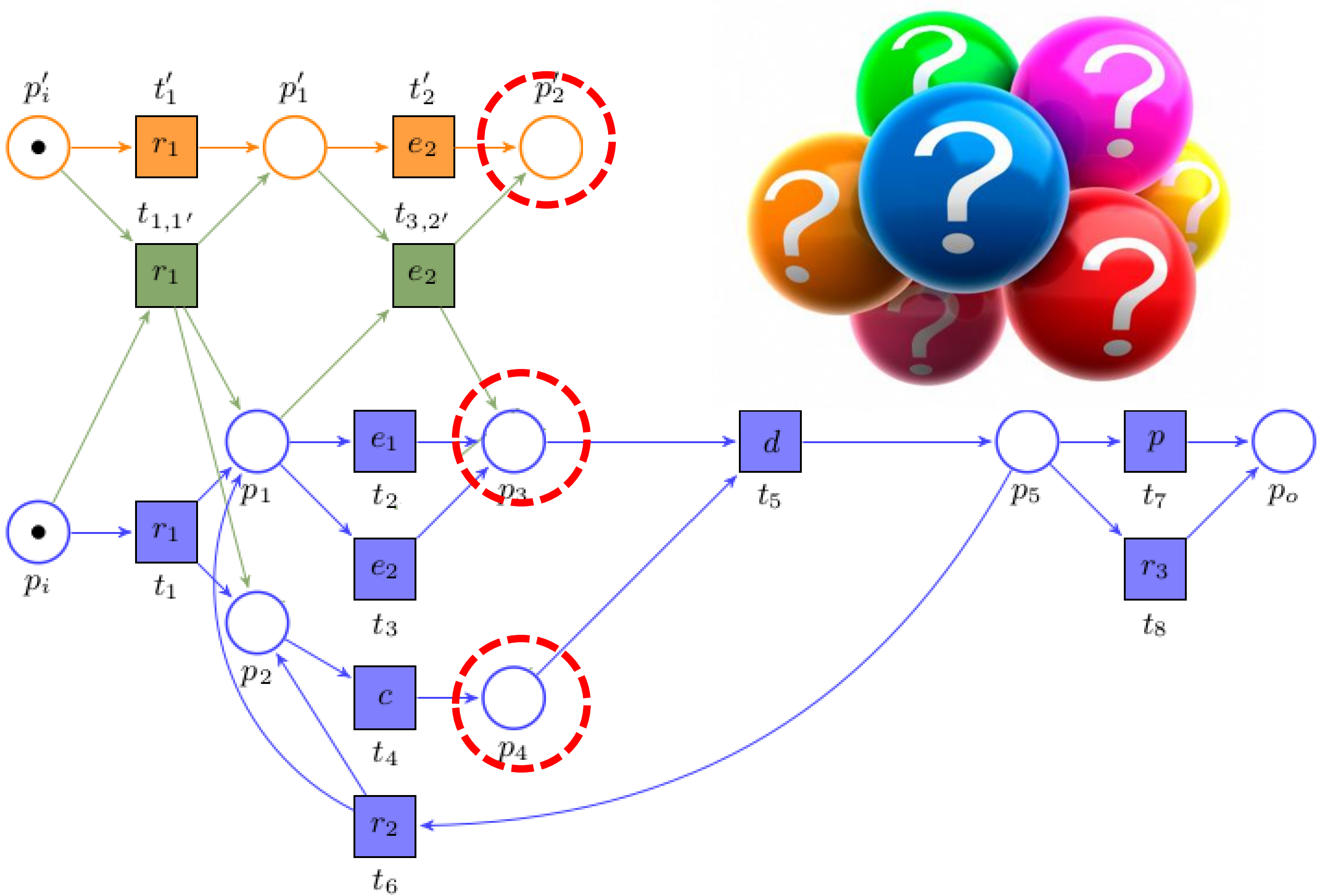
Stream-Based Conformance Checking

- <r1, e2,> → all we have seen so far



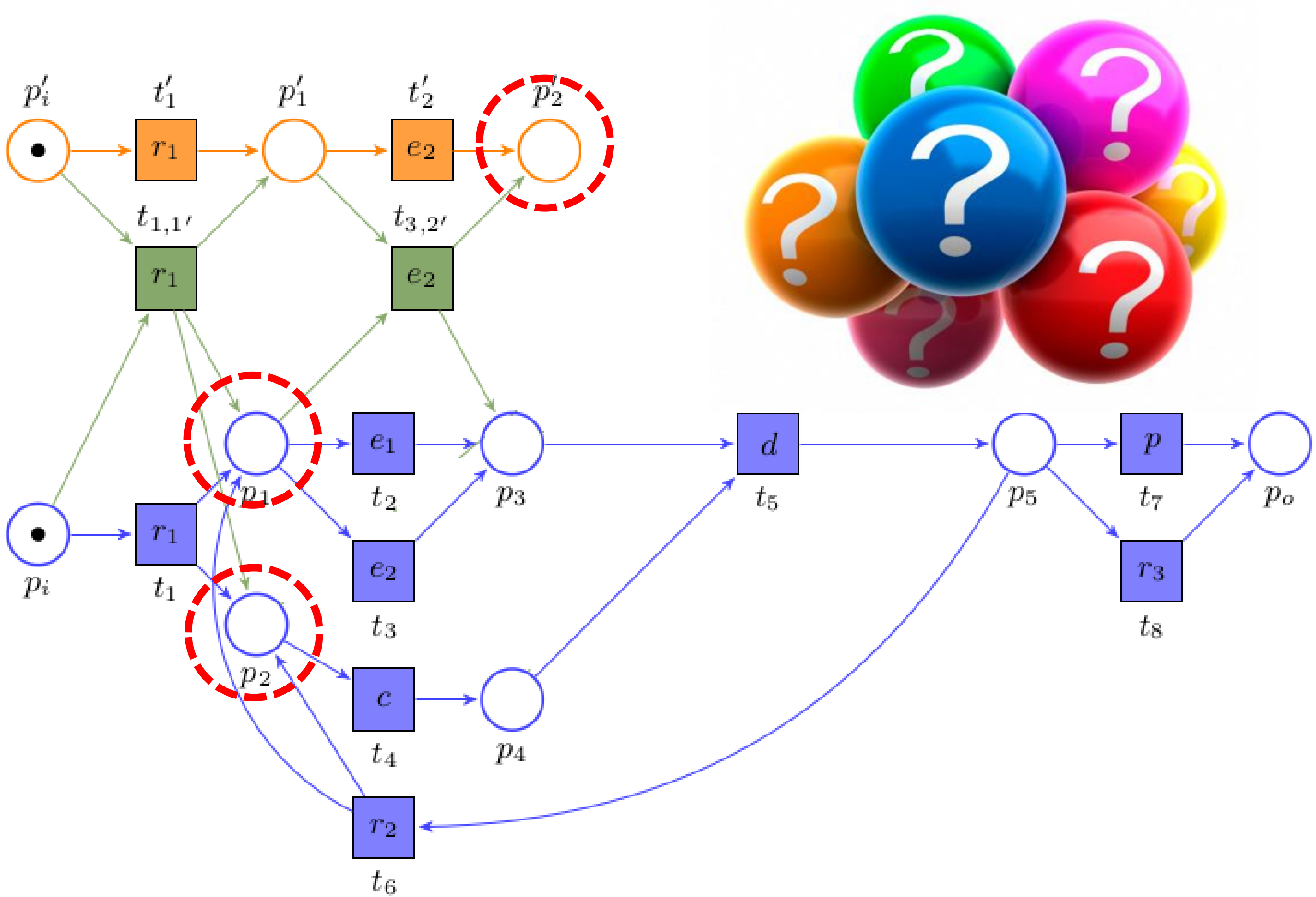
Stream-Based Conformance Checking

- <r1, e2,> → all we have seen so far



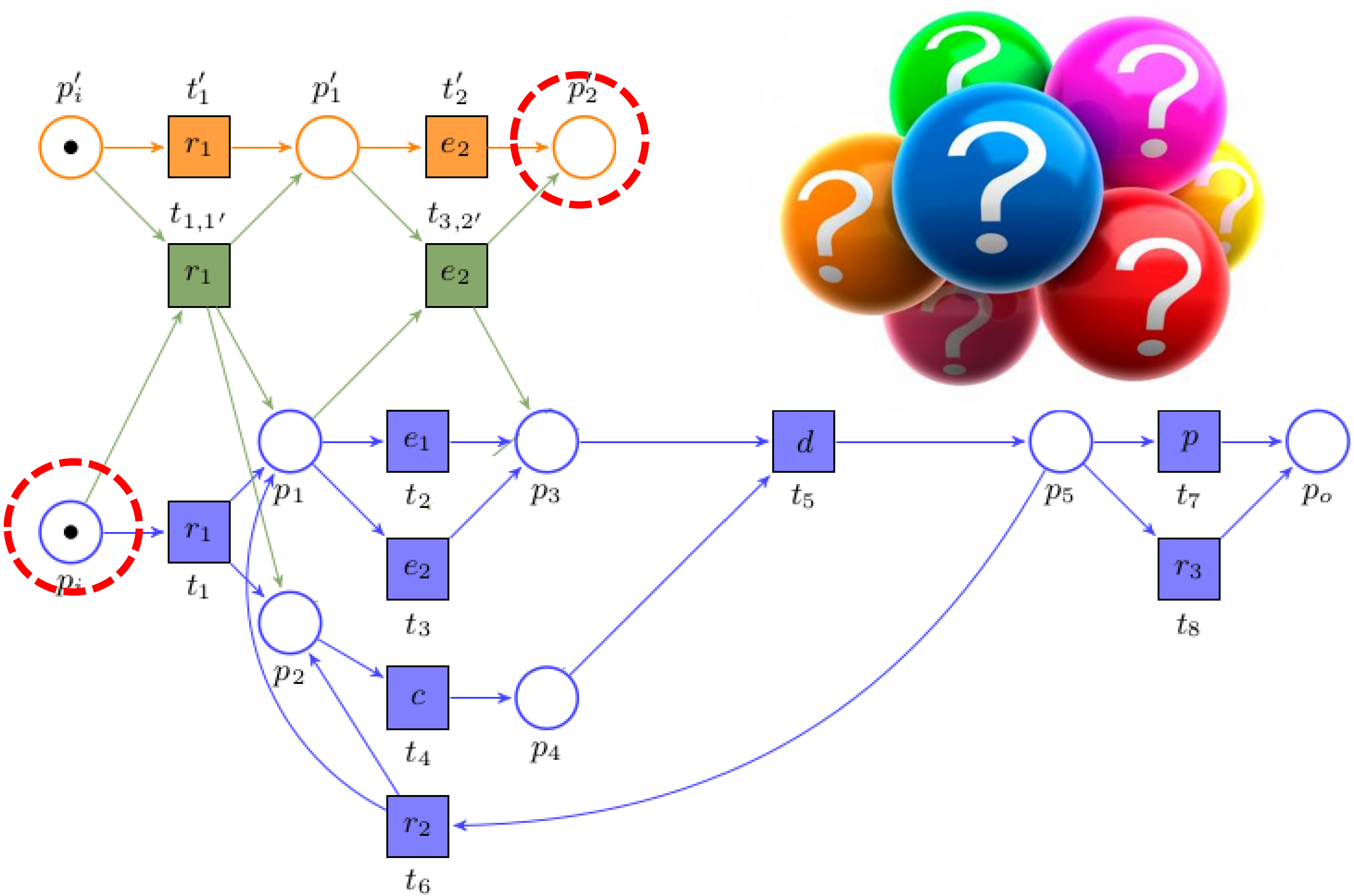
Stream-Based Conformance Checking

- <r1, e2,> → all we have seen so far



Stream-Based Conformance Checking

- <r1, e2,> → all we have seen so far



Stream-Based Conformance Checking

- Prefix-Alignment
  - Explain all events in the trace
  - Finish in a marking that still enables us to reach the final marking

Stream-Based Conformance Checking

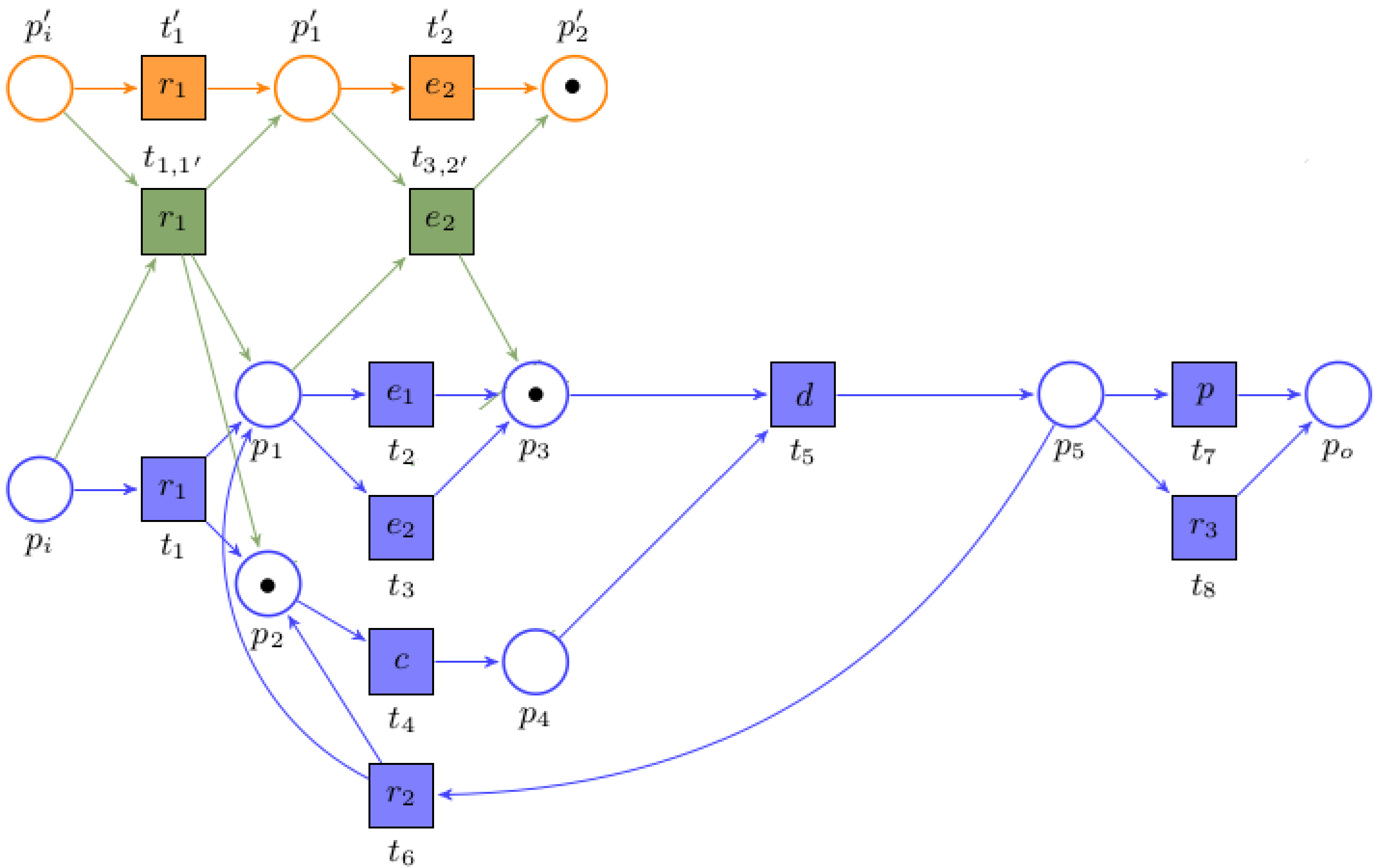
- Prefix-Alignment
  - Explain all events in the trace
  - Finish in a marking that still enables us to reach the final marking

  - … to fix req #2: We assume that the model is sound

Stream-Based Conformance Checking

Approximation Scheme

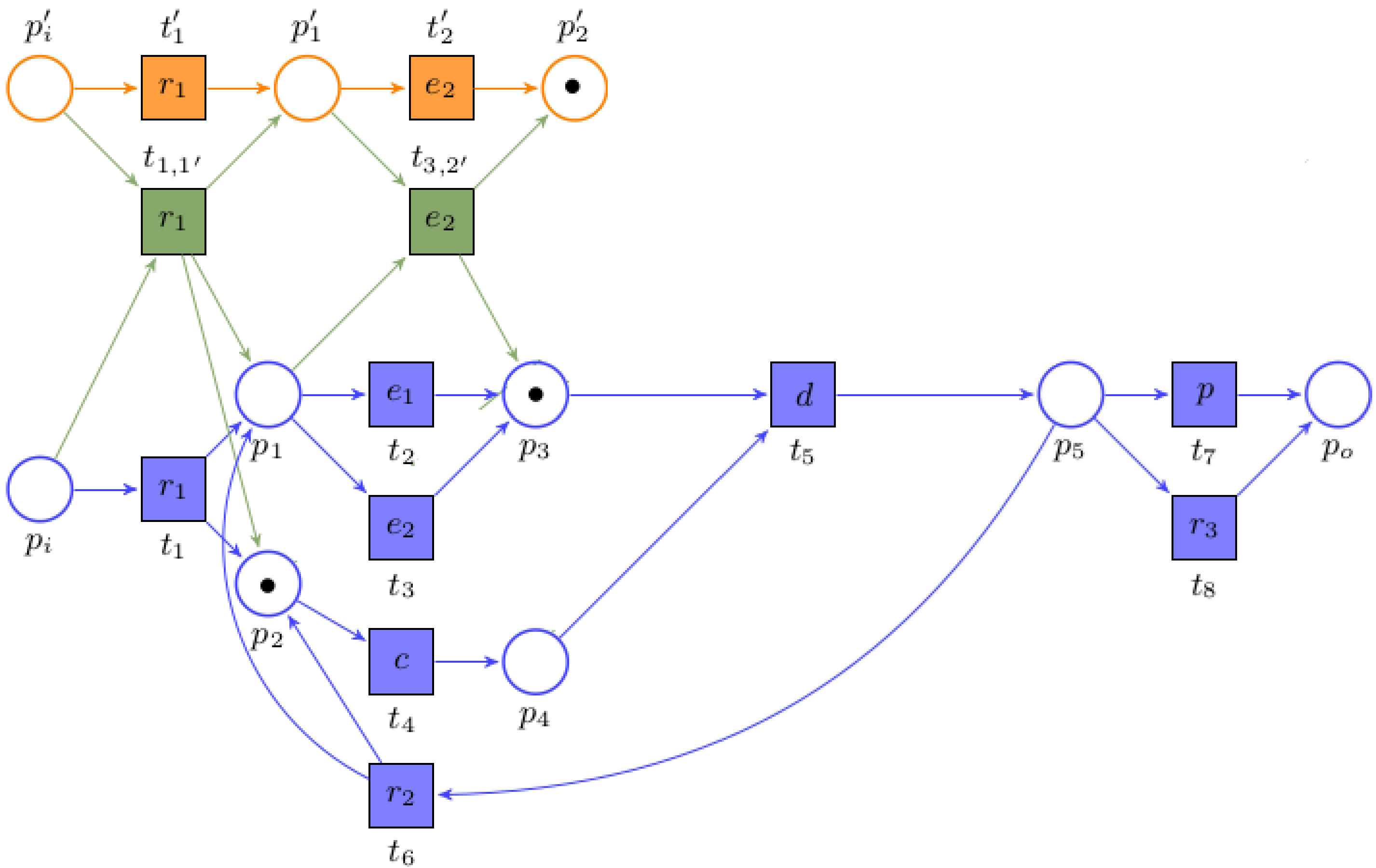
- <$t_{1,1'}$, $t_{3,2'}$> **Current Alignment**



Stream-Based Conformance Checking

Approximation Scheme

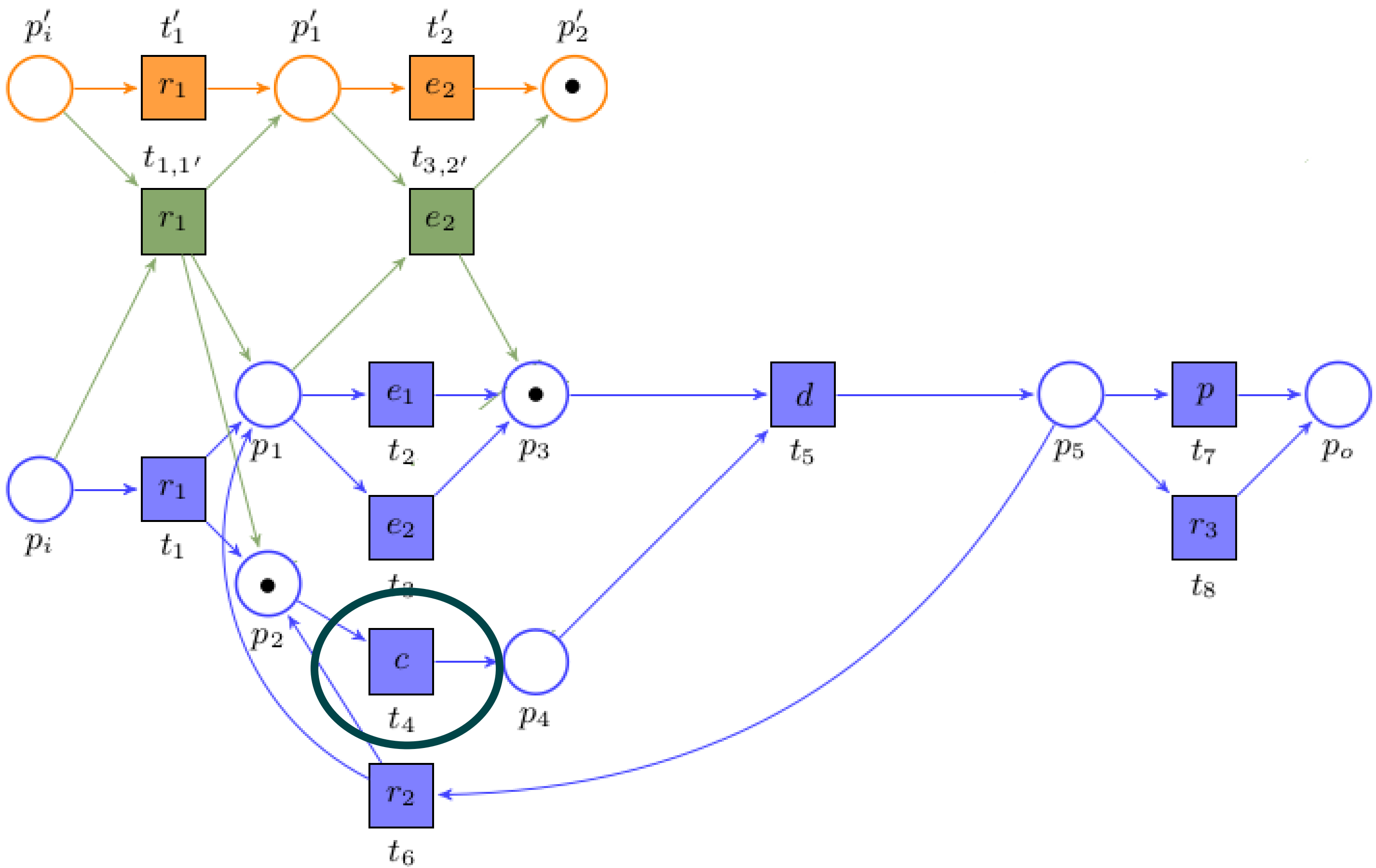
- $<t_{1,1'}, t_{3,2'}>$ receive: activity c



Stream-Based Conformance Checking

Approximation Scheme

- $\langle t_{1,1'}, t_{3,2'} \rangle$ receive: activity c

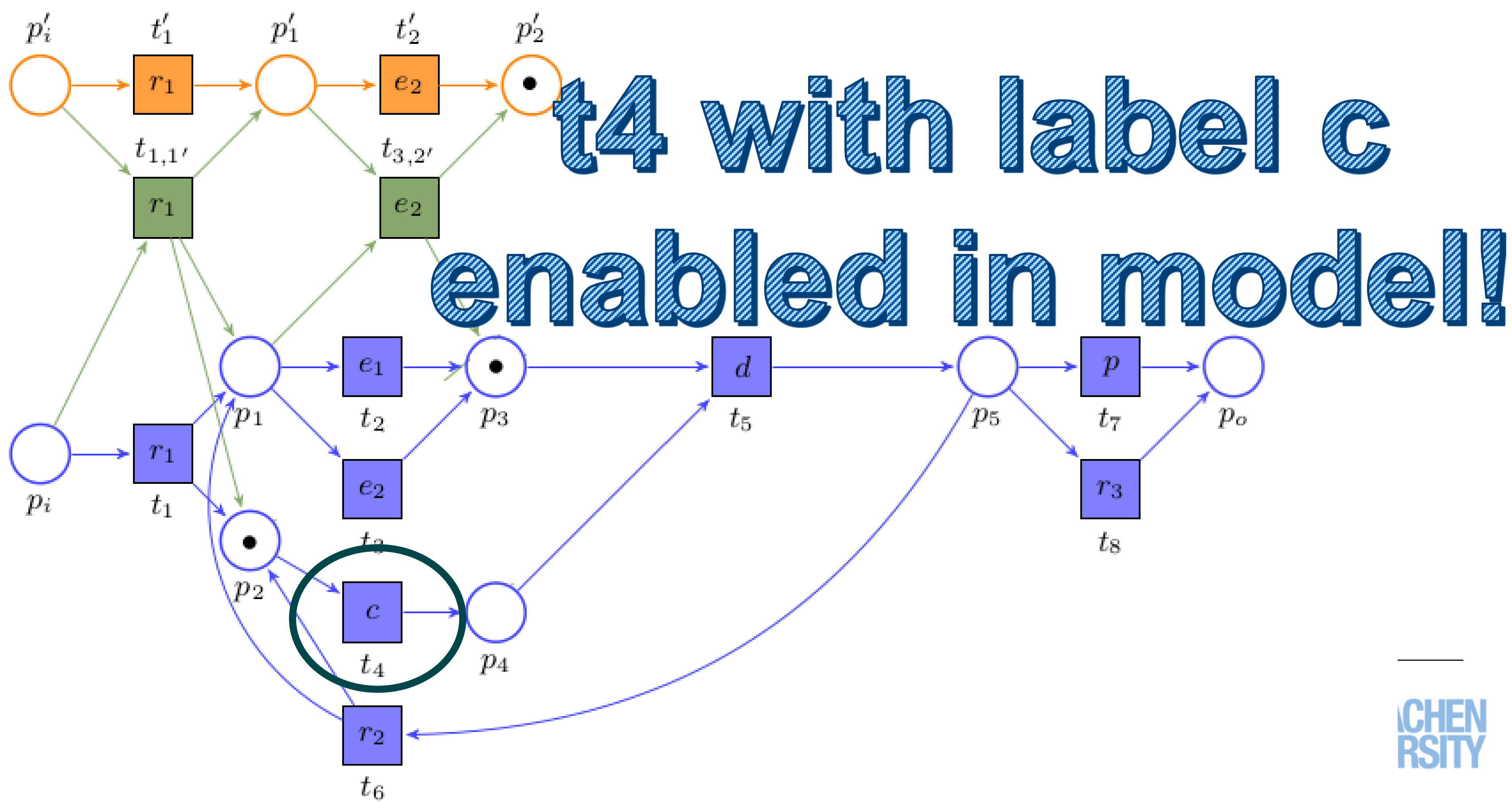


Stream-Based Conformance Checking

Approximation Scheme

- <$t_{1,1'}$, $t_{3,2'}$>

receive: activity c



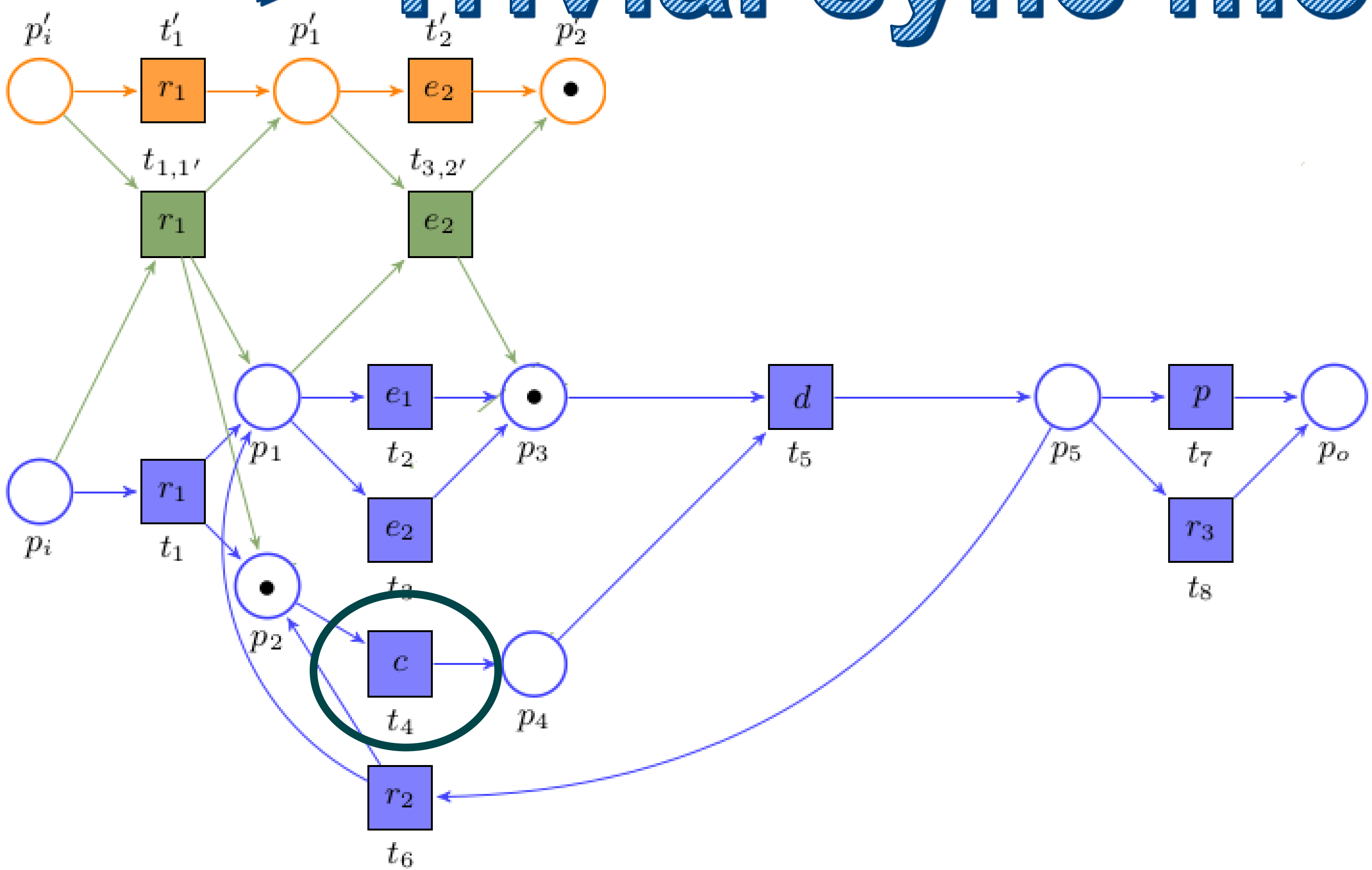
Stream-Based Conformance Checking

Approximation Scheme

- <$t_{1,1'}$, $t_{3,2'}$>

receive: activity c

=> Trivial sync move

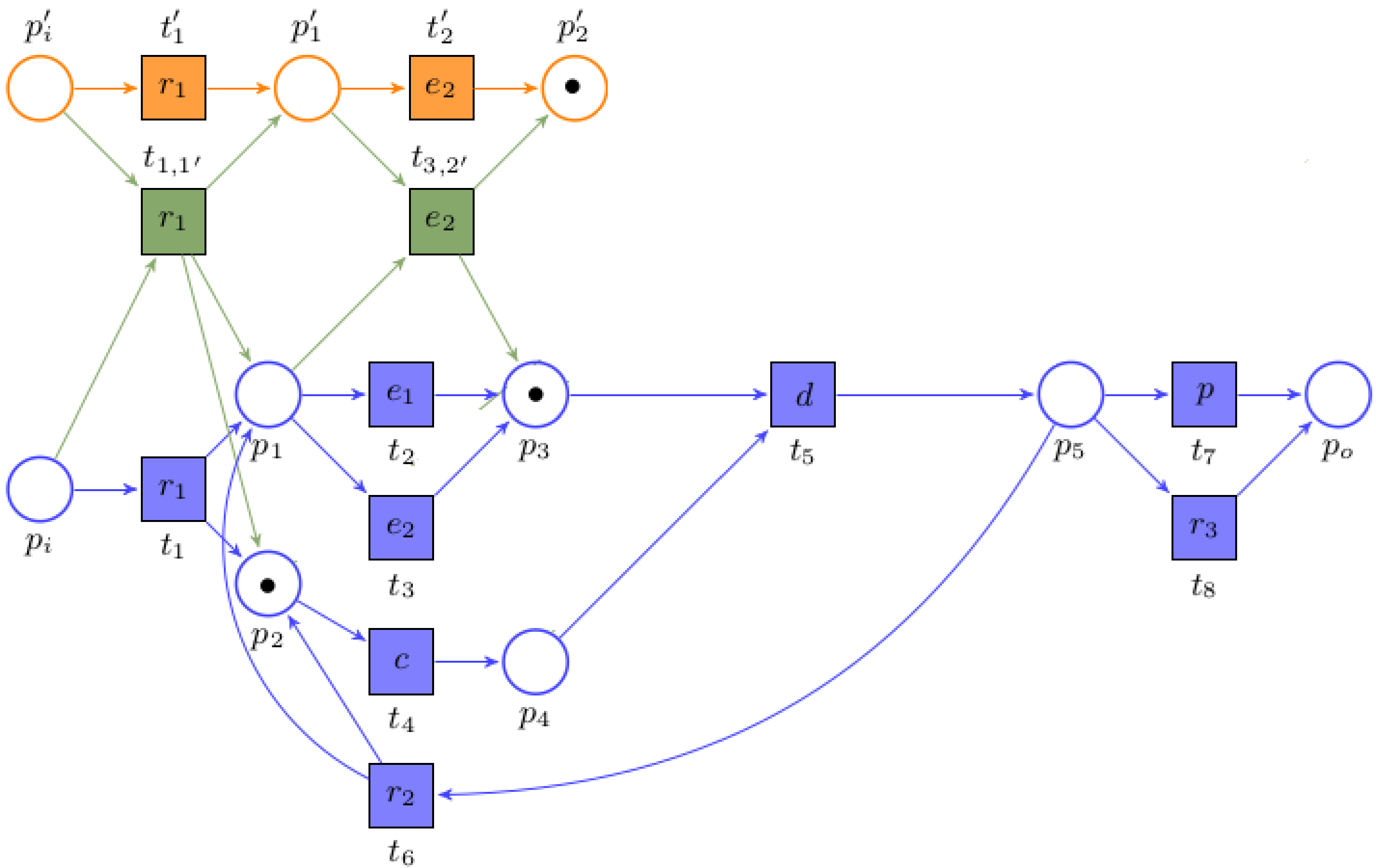


Stream-Based Conformance Checking

Approximation Scheme

- $\langle t_{1,1'}, t_{3,2'} \rangle$

receive: activity f

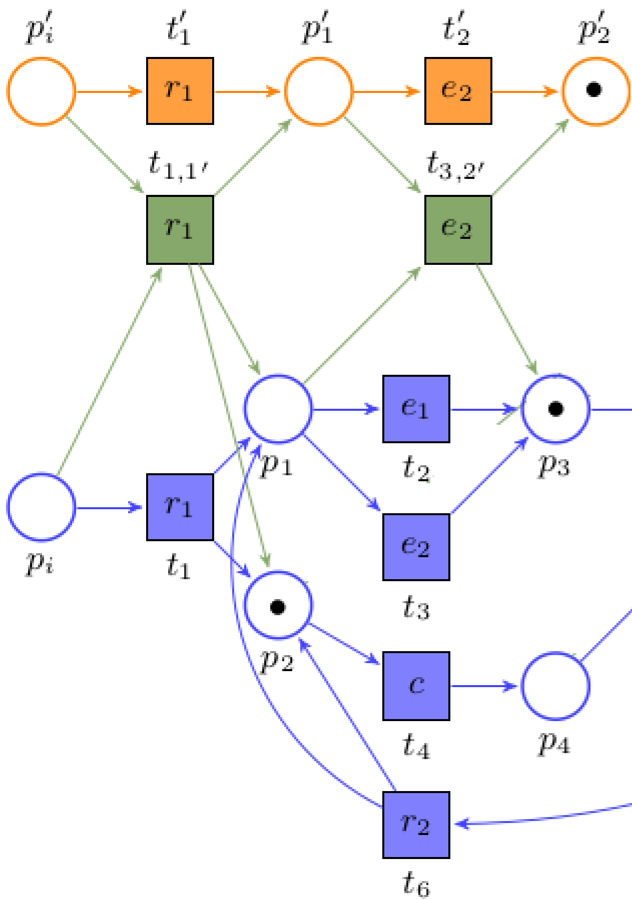


Stream-Based Conformance Checking

Approximation Scheme

- $\langle t_{1,1'}, t_{3,2'} \rangle$

receive: activity f



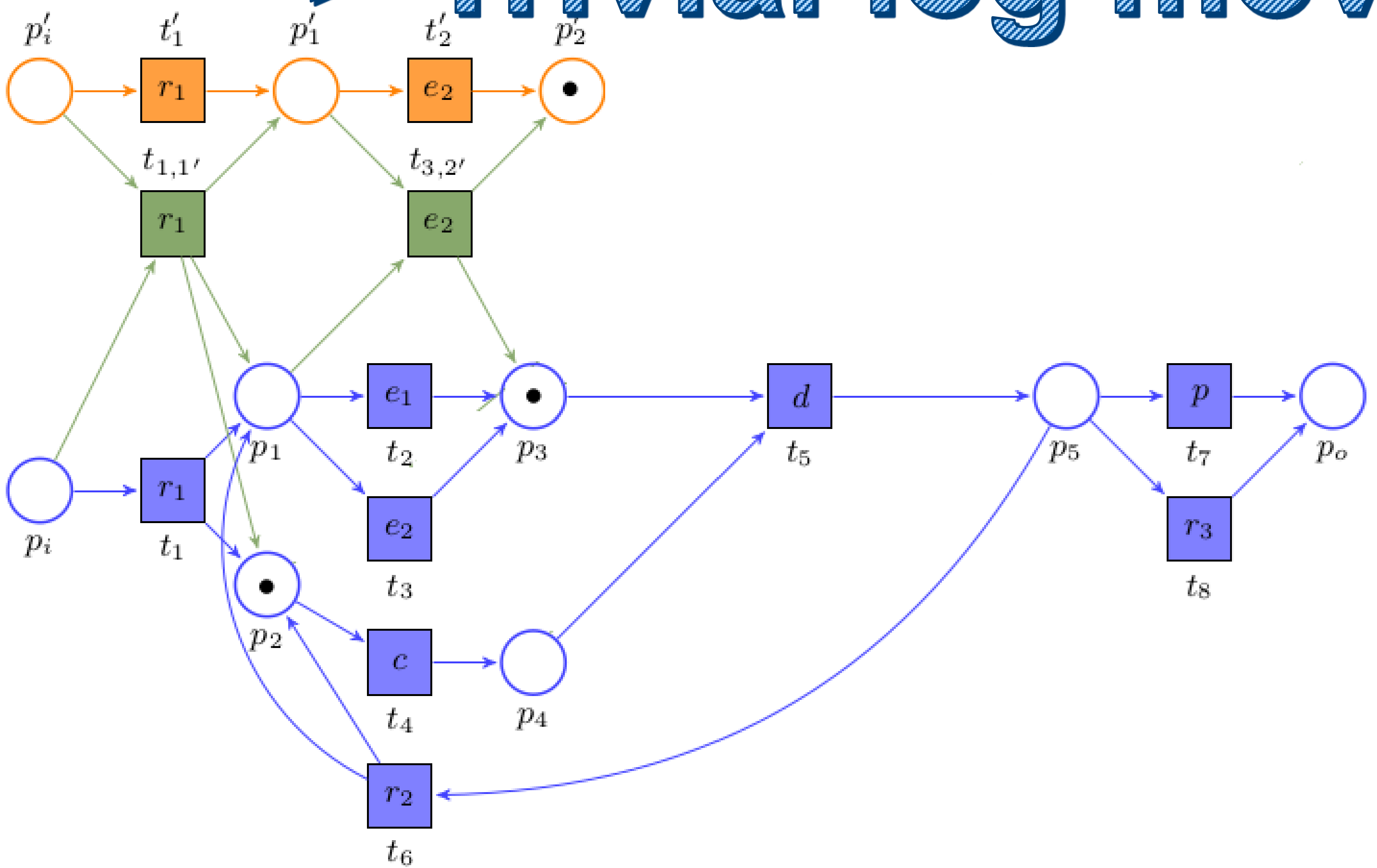
Not in model

Stream-Based Conformance Checking

Approximation Scheme

- $<t_{1,1'}, t_{3,2'}>$ receive: activity f

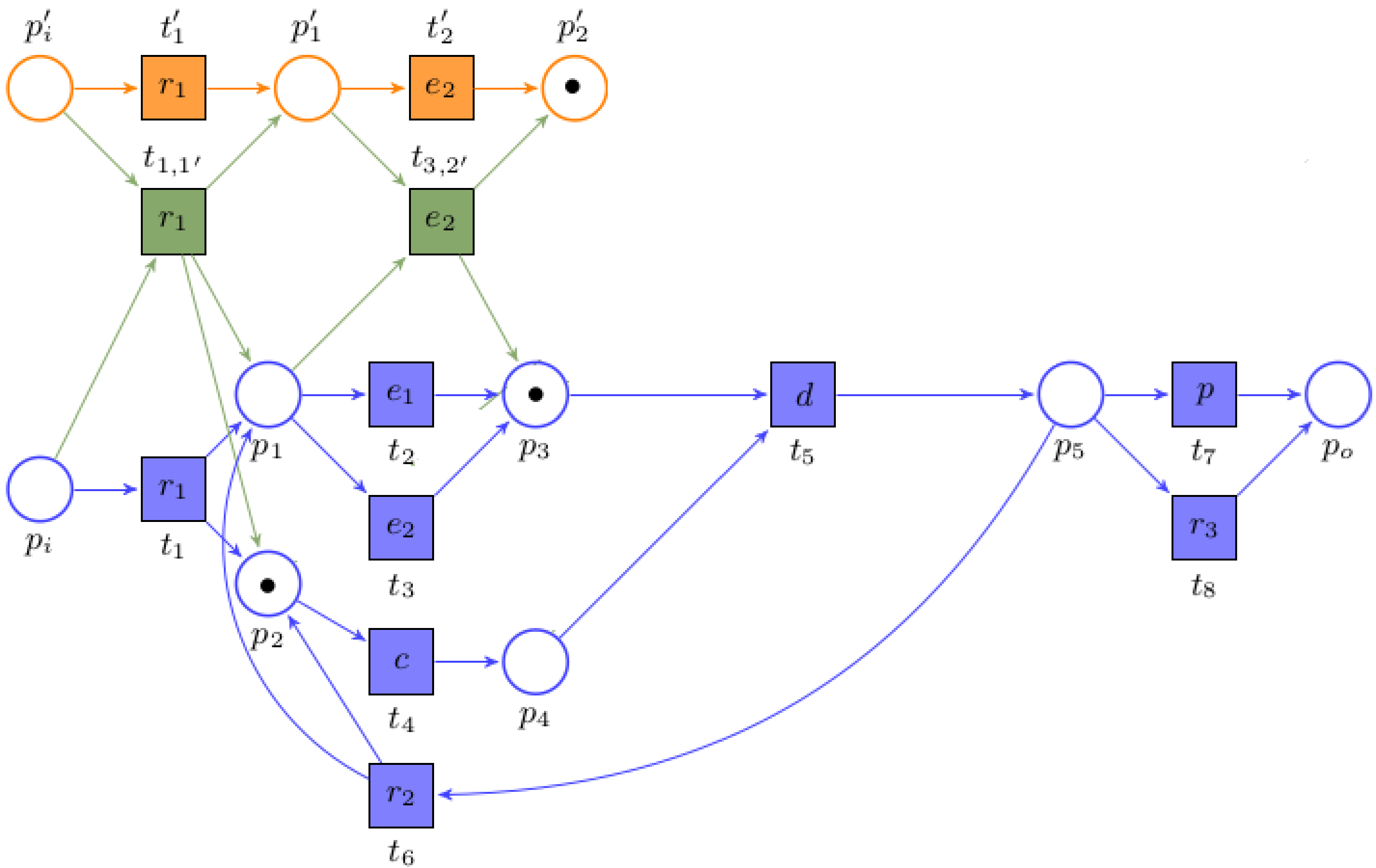
=> Trivial log move



Stream-Based Conformance Checking

Approximation Scheme

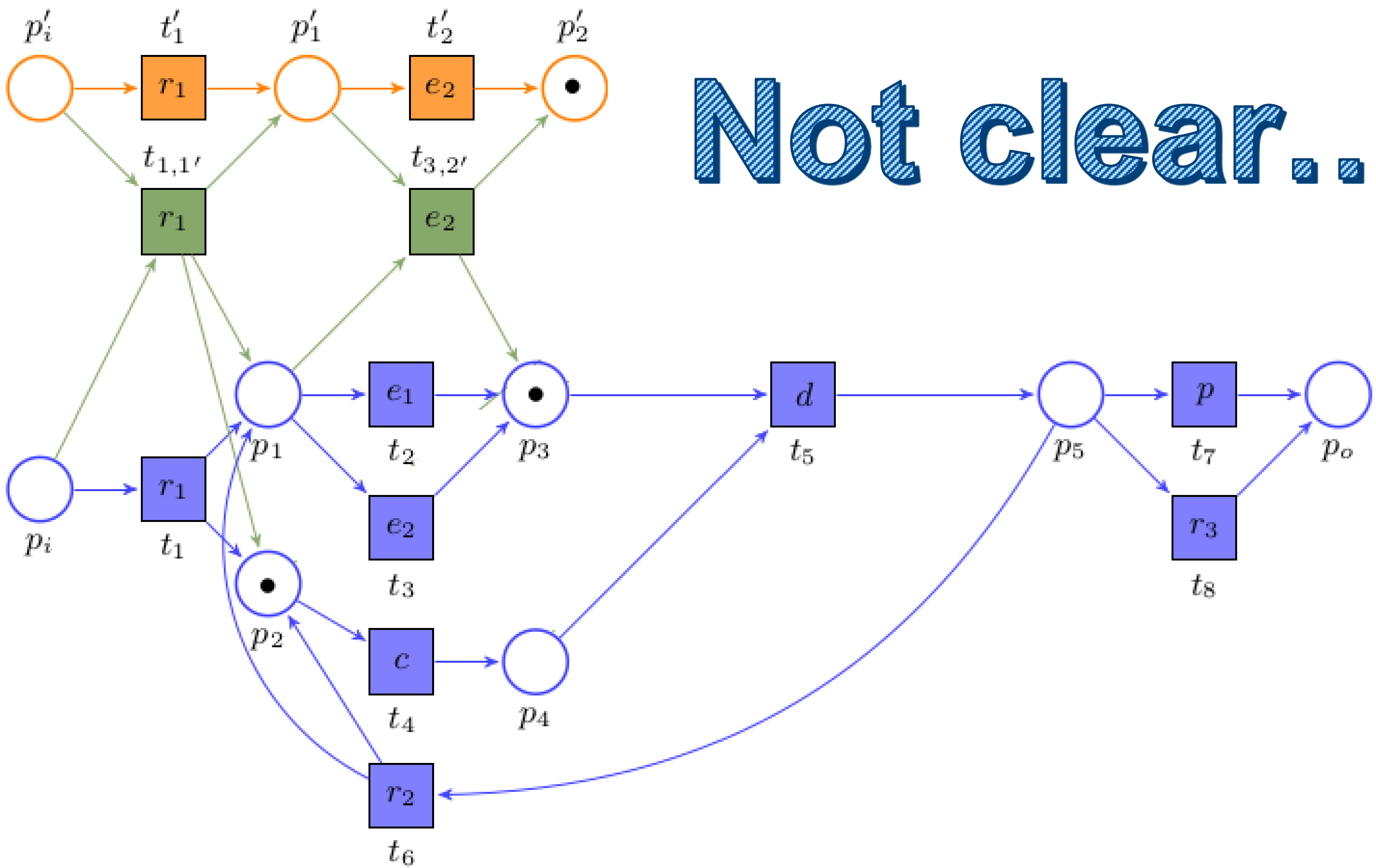
- <$t_{1,1'}$, $t_{3,2'}$> receive: activity e_1



Stream-Based Conformance Checking

Approximation Scheme

- <$t_{1,1'}$, $t_{3,2'}$> receive: activity e_1



Not clear…

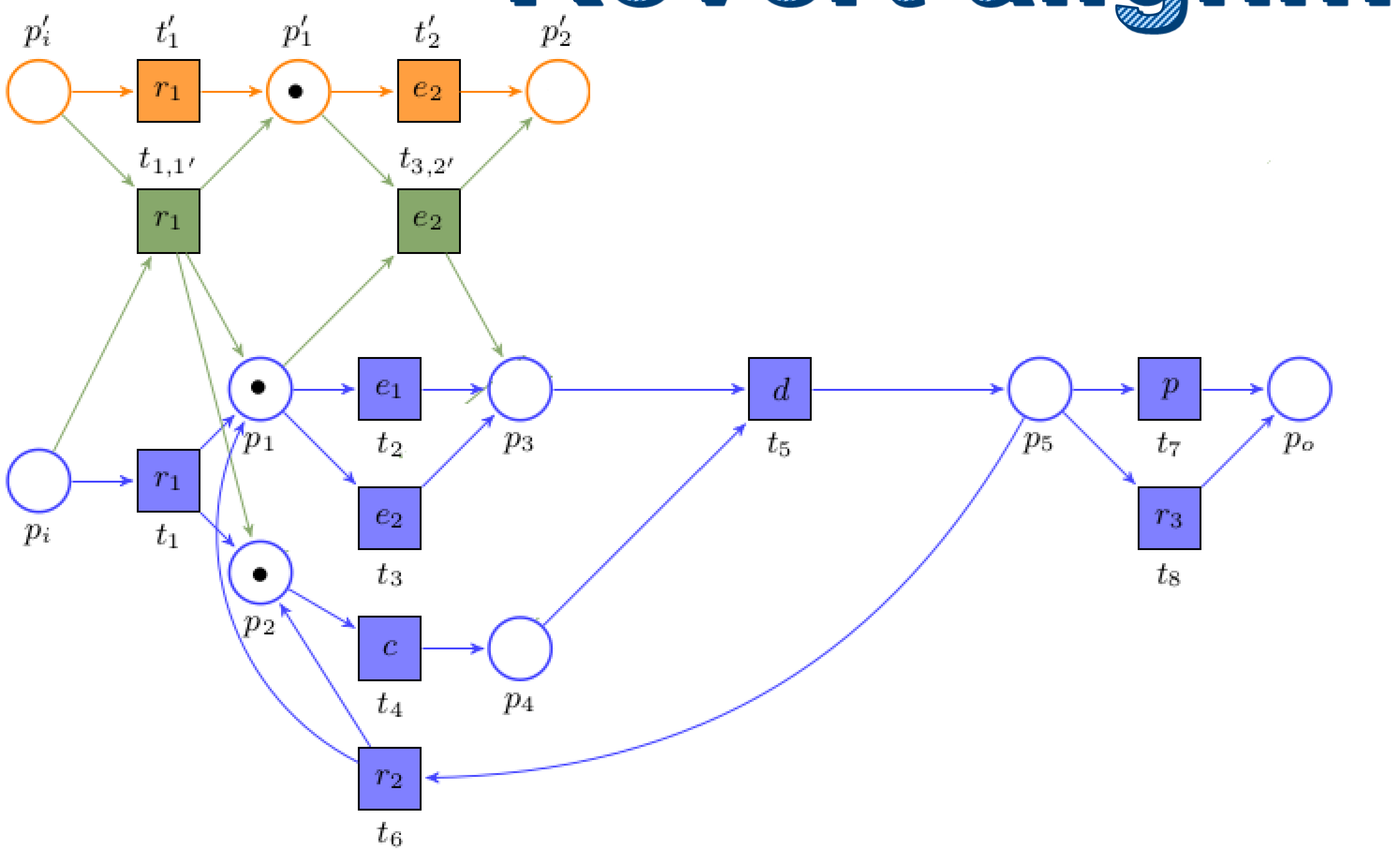
Stream-Based Conformance Checking

Approximation Scheme

- $\langle t_{1,1'}, t_{3,2'} \rangle$

receive: activity e_1

Revert alignment



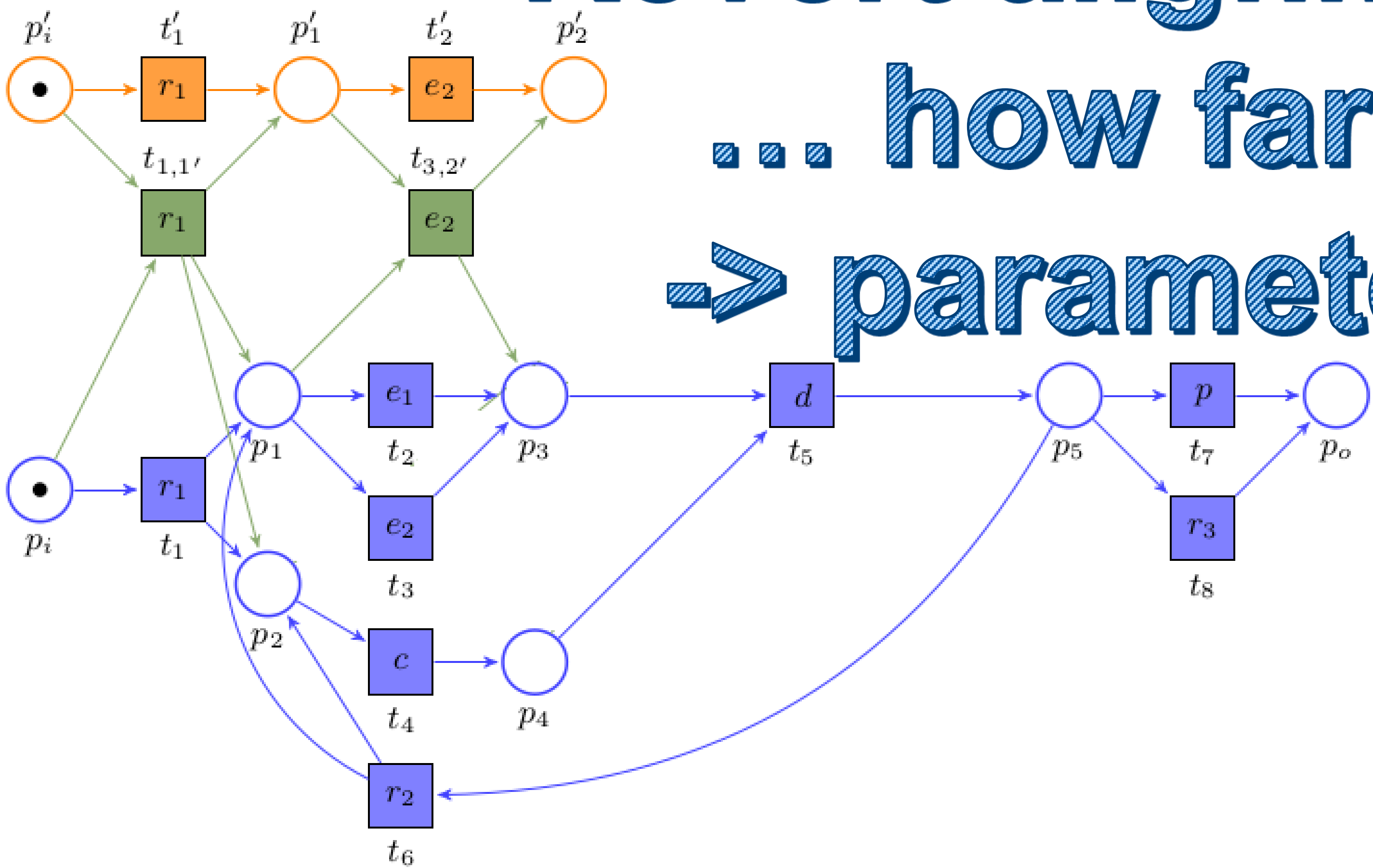
Stream-Based Conformance Checking

Approximation Scheme

- <$t_{1,1'}$, $t_{3,2'}$>

receive: activity e_1

Revert alignment ... how far? -> parameter



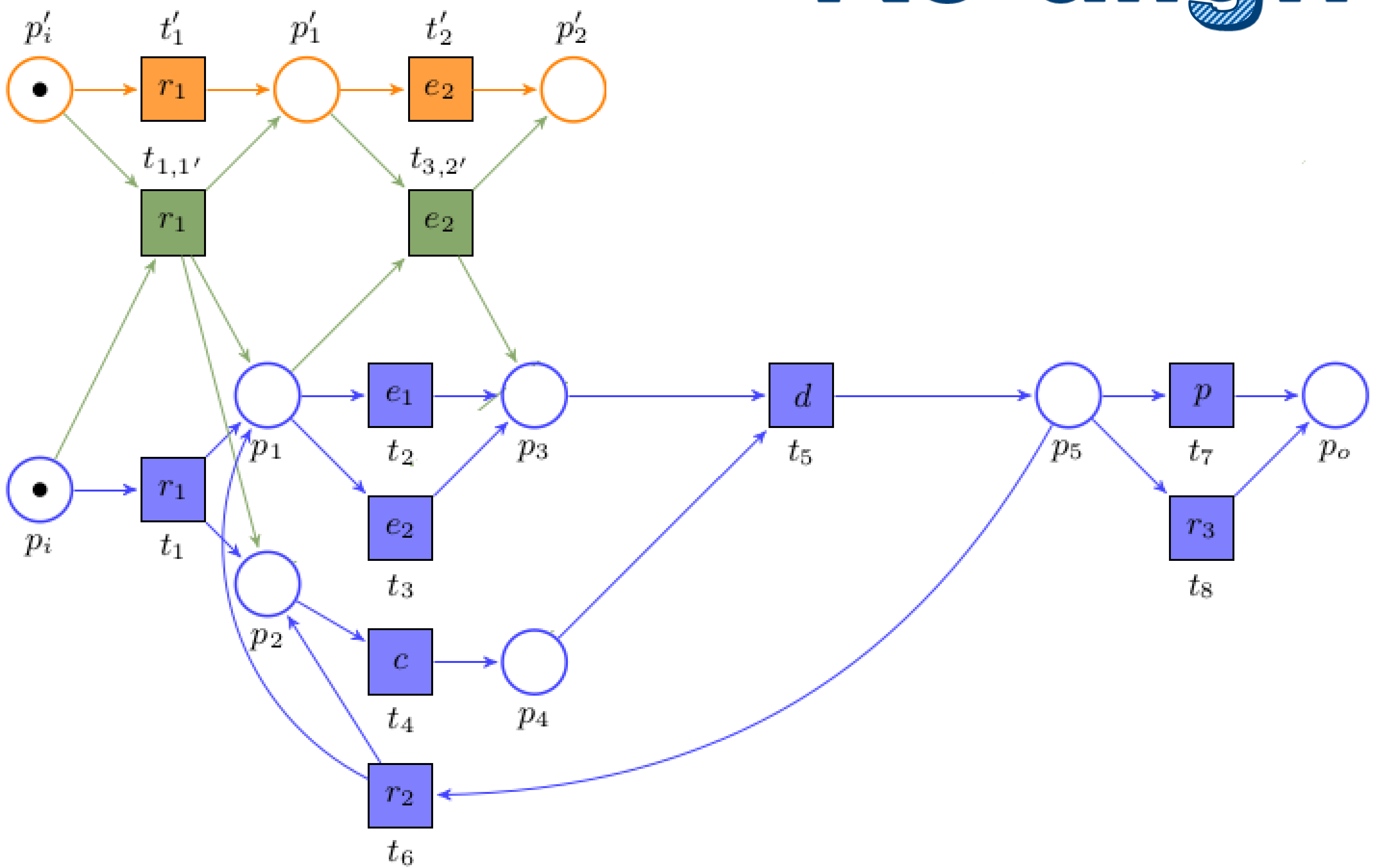
Stream-Based Conformance Checking

Approximation Scheme

- <t$_{1,1'}$, t$_{3,2'}$>

receive: activity e$_1$

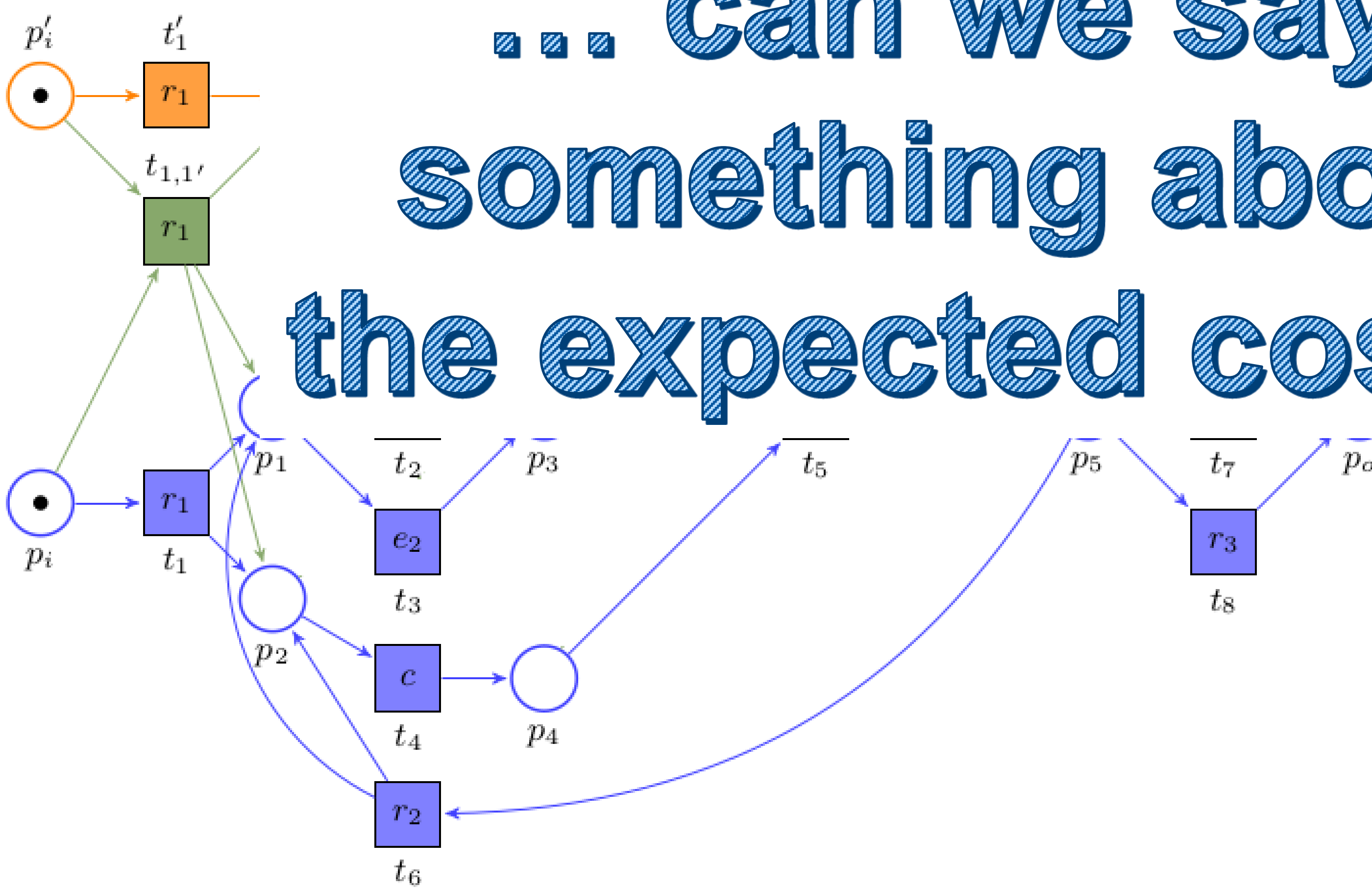
Re-align



Stream-Based Conformance Checking

Approximation Scheme

- $<t_{1,1'}, t_{3,2'}>$ receive: activity e_1



... can we say something about the expected costs?

Stream-Based Conformance Checking

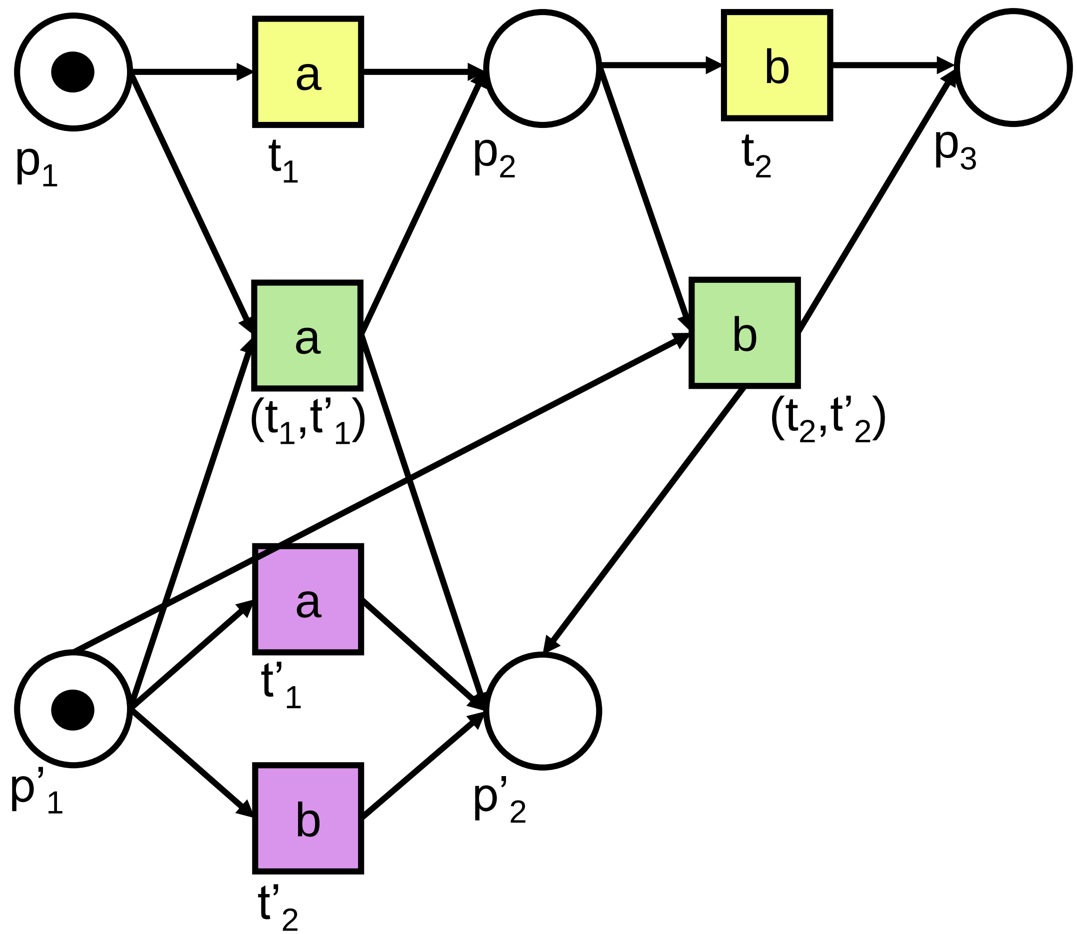
Approximation Scheme

$$\boldsymbol{min}(\vec{c}^{\intercal}\vec{x} \mid \vec{m}' = \vec{m} + \mathbf{A}^{\intercal}\vec{x})$$

- What to do when we do not know m’ exactly???

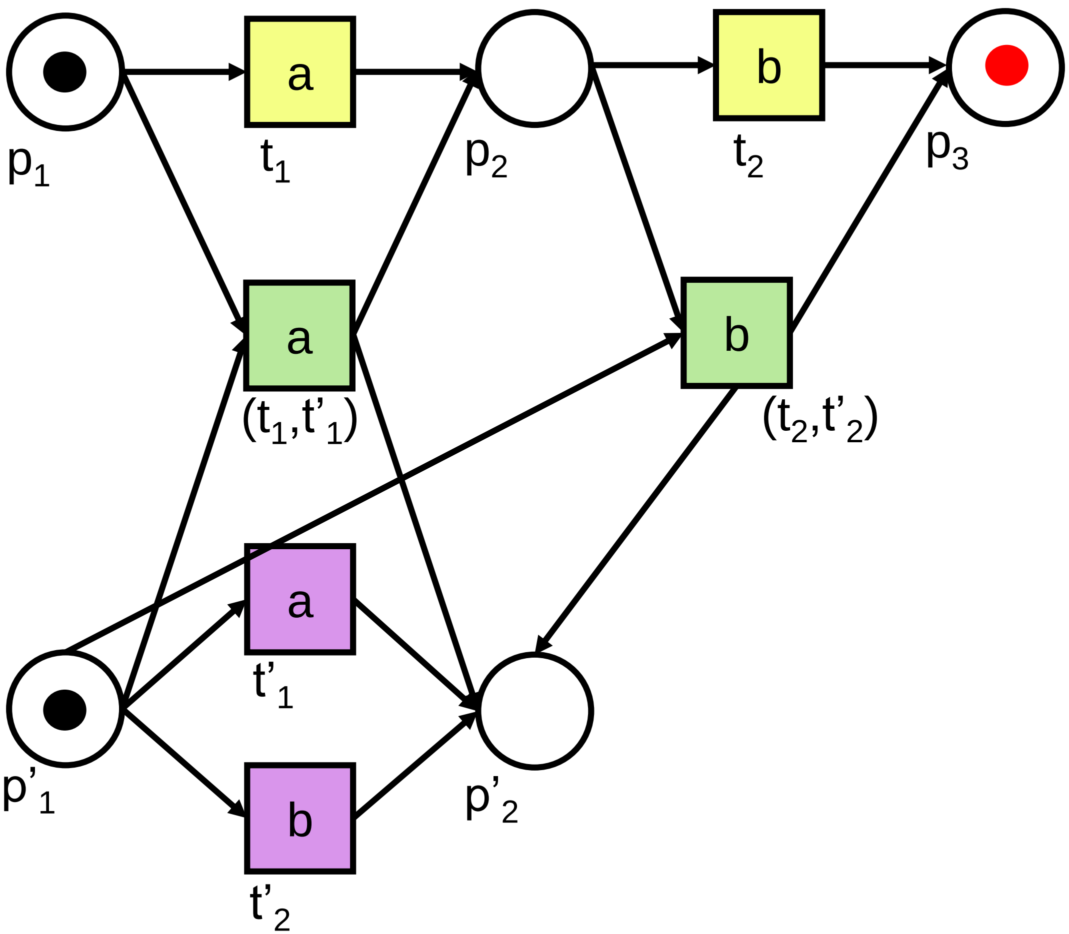
Stream-Based Conformance Checking

$$\begin{bmatrix} t_1 \\ t_2 \\ (t_1, t'_1) \\ (t_2, t'_2) \\ t'_1 \\ t'_2 \end{bmatrix}$$



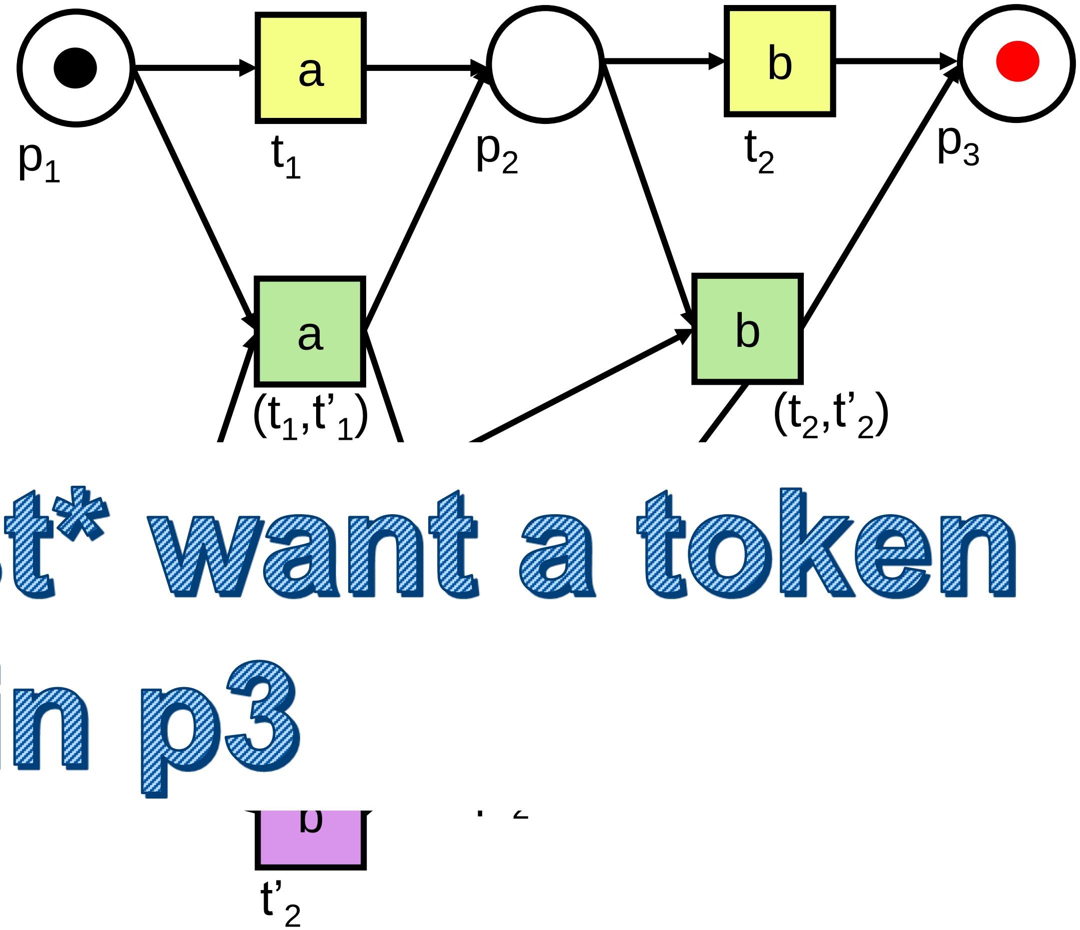
Limiting the Exploration Area

p_1: 0
p_2: 0
p_3: 1
p'_1: 0
p'_2: 0



Limiting the Exploration Area

p_1: 0
p_2: 0
p_3: 1
p'_1: 0
p'_2: 0



We *at least* want a token in p3

Stream-Based Conformance Checking

- More recently ‘discovered’
  - (MSc thesis Daniel Schuster @ PADS RWTH)
- We can simply continue the search in conventional A*
  - Recalculate the Heuristic for each state in the Open Set when a new event arrives

Stream-Based Conformance Checking

- More recently ‘discovered’
  - (MSc thesis Daniel Schuster @ PADS RWTH)

- We can simply continue the search in conventional A*

- Guarantees Optimality!!!
- Comparative with revert windows of 5-10…

Stream-Based Conformance Checking

- More recently ‘discovered’
  - (MSc thesis Daniel Schuster @ PADS RWTH)
- When models are flower-like models:
  - Use approximation scheme with small windows
    - Often leads to optimal-ish results

Stream-Based Conformance Checking

- More recently ‘discovered’
  - (MSc thesis Daniel Schuster @ PADS RWTH)
- When models are flower-like models:
  - Use approximation scheme with small windows
    - Often leads to optimal-ish results
- When models contain choices
  - The Optimal scheme starts to beat the approximation scheme

Chair of Process and Data Science

RWTH AACHEN UNIVERSITY

https://goodjobpal.com/wp-content/uploads/2019/01/Thank-you-word-cloud.jpg