FASE: Fast Asynchronous System Evaluation

A tool for performance evaluation

Federico Buti, Massimo Callisto De Donato
Flavio Corradini, Maria Rita Di Berardini

{federico.buti, massimo.callisto, flavio.corradini, mariarita.diberardini}@unicam.it

Dipartimento di Informatica - Università di Camerino
Via Madonna delle Carceri, 9 - 62032 Camerino
http://www.cs.unicam.it

PaCo Meeting - June, 26 2009

Agenda

- FASE presentation
- Quantitative performance evaluation techniques, some hints
- A case study: three buffer implementations
- Conclusions

FASE

- Software *cross-platform*, completely written in Java

FASE

- Software *cross-platform*, completely written in Java
- Interprets PAFAS strings from different dialects

FASE

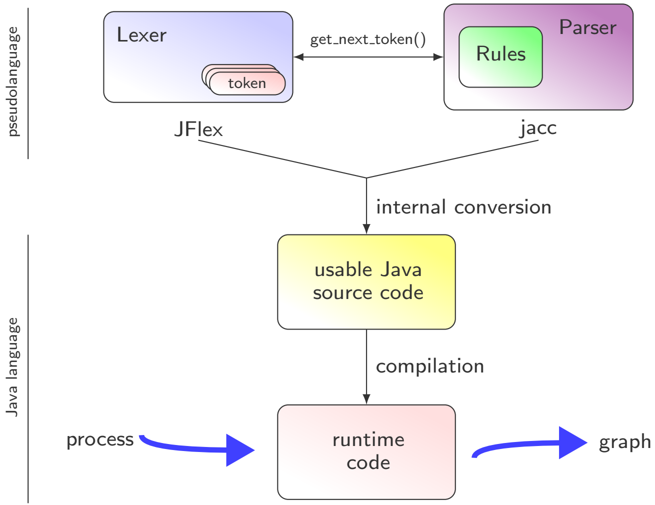
- Software *cross-platform*, completely written in Java
- Interprets PAFAS strings from different dialects
- Provides an arc-labeled graph representation of the given input (RTS, rRTS...)

FASE

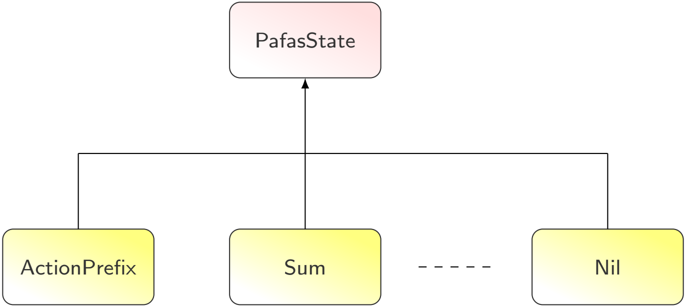
- Software *cross-platform*, completely written in Java
- Interprets PAFAS strings from different dialects
- Provides an arc-labeled graph representation of the given input (RTS, rRTS...)
- On these representations, applies performance calculation techniques based on the PAFAS *efficiency preorder* theory.

- Software *cross-platform*, completely written in Java
- Interprets PAFAS strings from different dialects
- Provides an arc-labeled graph representation of the given input (RTS, rRTS...)
- On these representations, applies performance calculation techniques based on the PAFAS *efficiency preorder* theory.
- Allows to compare performance of systems that *functionally* execute the same tasks but that are implemented in different ways.

Architectural overview

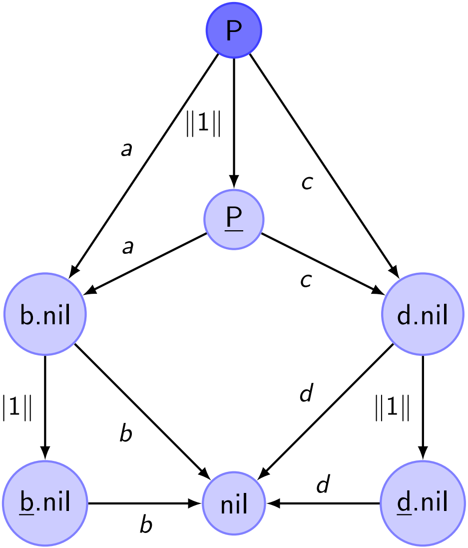


Architectural overview



Input & Output (rRTS)

P = a.b.nil
+ c.d.nil



Response processes

- Response performance calculation is applied to a particular kind of processes called *response processes* which have some particular characteristics:

Response processes

- Response performance calculation is applied to a particular kind of processes called *response processes* which have some particular characteristics:
    1. finite state processes

Response processes

- Response performance calculation is applied to a particular kind of processes called *response processes* which have some particular characteristics:
  1. finite state processes
  2. available actions are only *in*'s and *out*'s (and τ's)

Response processes

- Response performance calculation is applied to a particular kind of processes called *response processes* which have some particular characteristics:
  1. finite state processes
  2. available actions are only *in*'s and *out*'s (and τ's)
  3. their number have to be *balanced* (and in general *out*'s cannot exceed *in*'s in number)

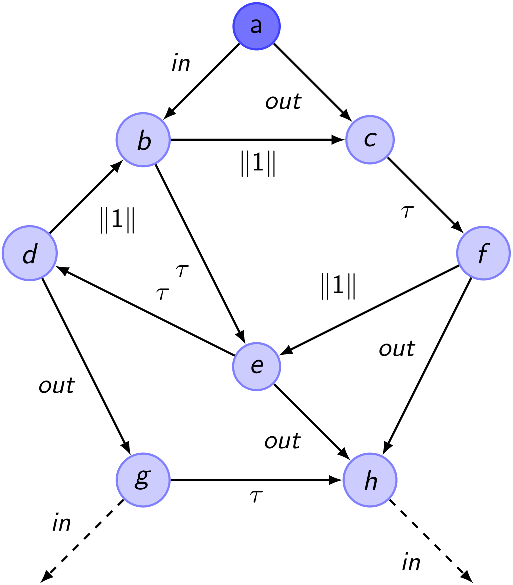
Response processes

- Response performance calculation is applied to a particular kind of processes called *response processes* which have some particular characteristics:
  1. finite state processes
  2. available actions are only *in*'s and *out*'s (and τ's)
  3. their number have to be *balanced* (and in general *out*'s cannot exceed *in*'s in number)
- If a response process is **correct** (verification could be done in *linear time*), then its response performance is finite and can be calculated.

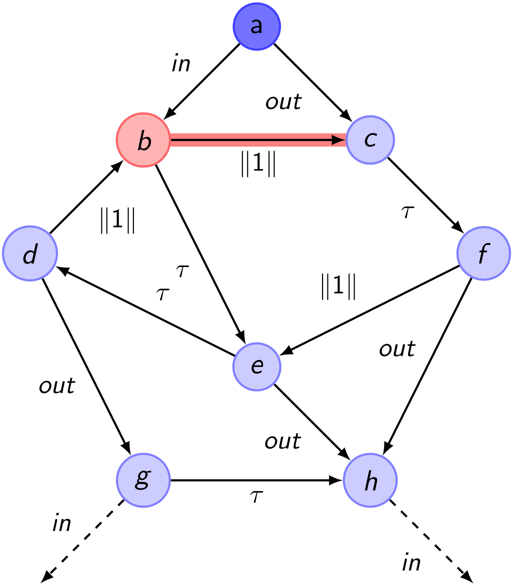
Incorrectness

- If the response process is not correct its response performance is ∞. This means that some requests will not be satisfied from the process within any time bound, which is certainly an incorrect behaviour.
- Typically an incorrect response process contains one (or more) *catastrophic cycle(s)*.
- When a process enters one of these cycles, it is impossible to know when (and if) it will come out.
- Verification of these cycles is so crucial for testing performances.

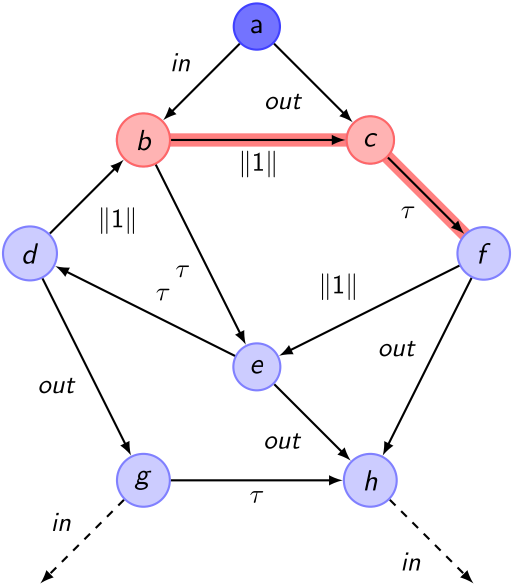
Catastrophic cycles - an example



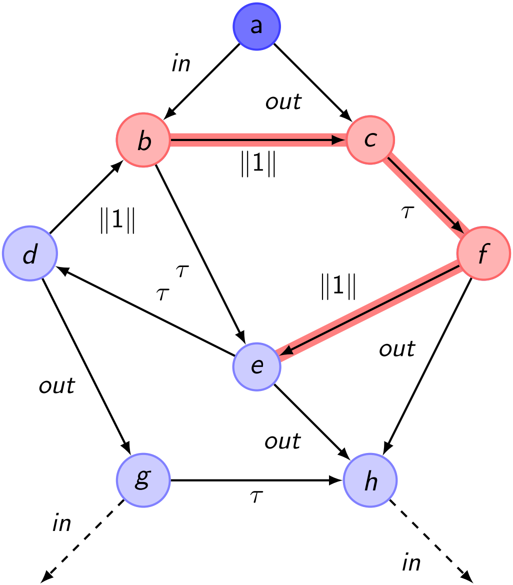
Catastrophic cycles - an example



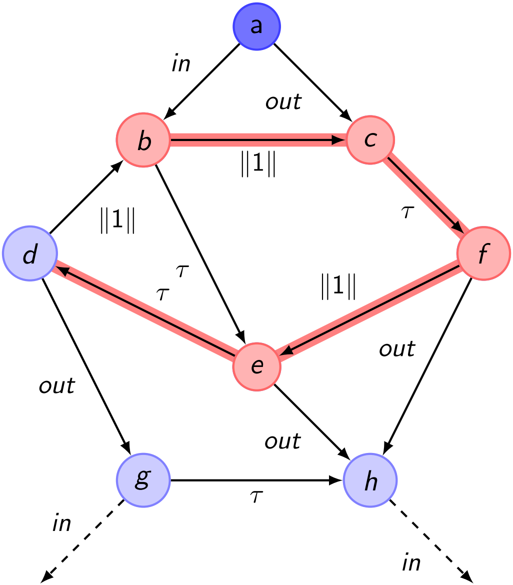
Catastrophic cycles - an example



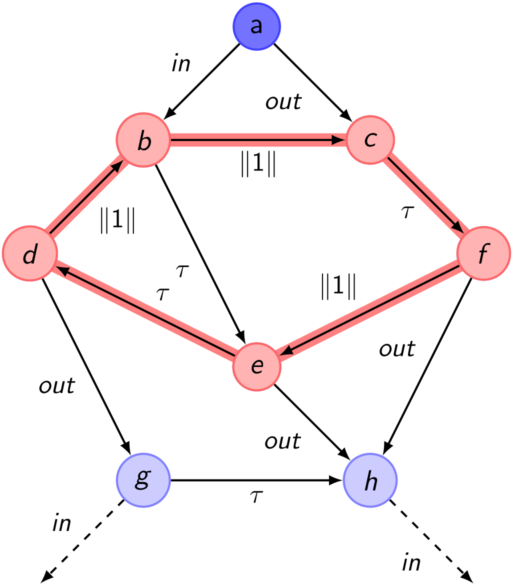
Catastrophic cycles - an example



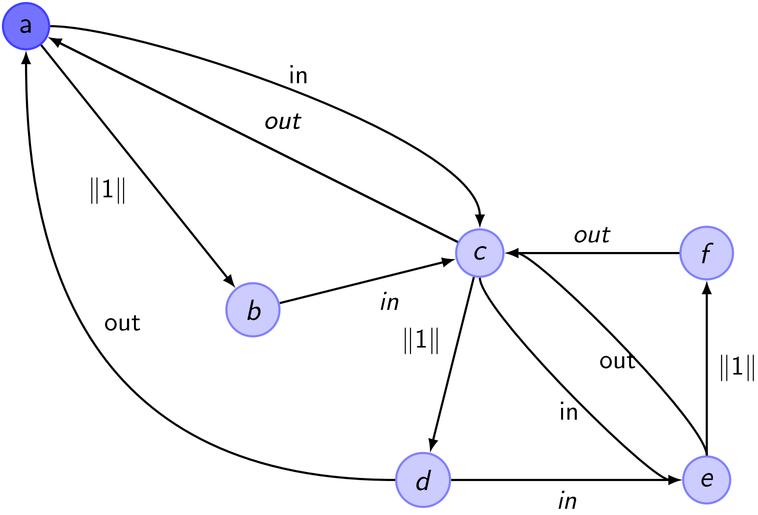
Catastrophic cycles - an example



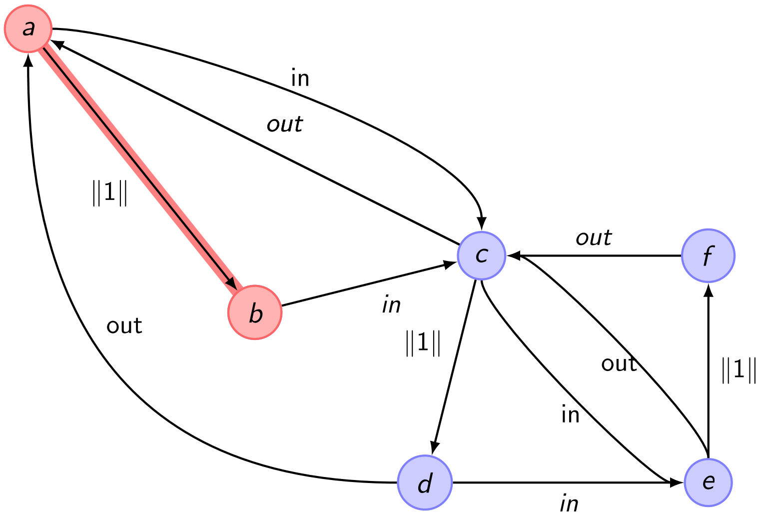
Catastrophic cycles - an example



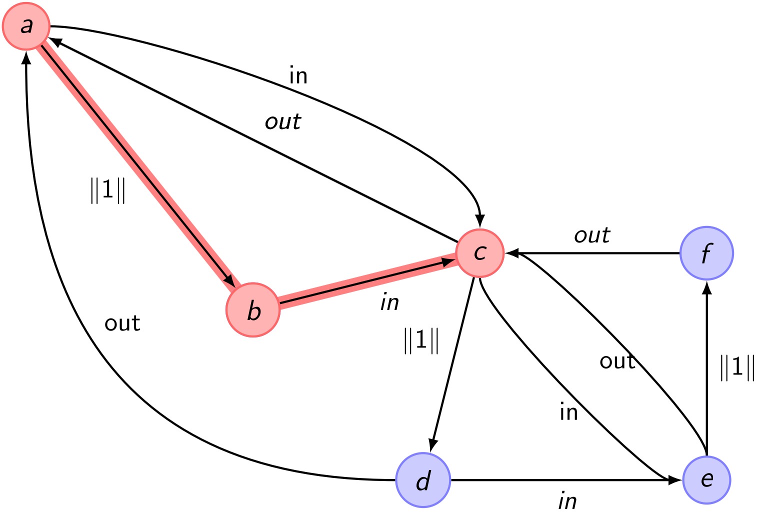
N-critical paths - two input example



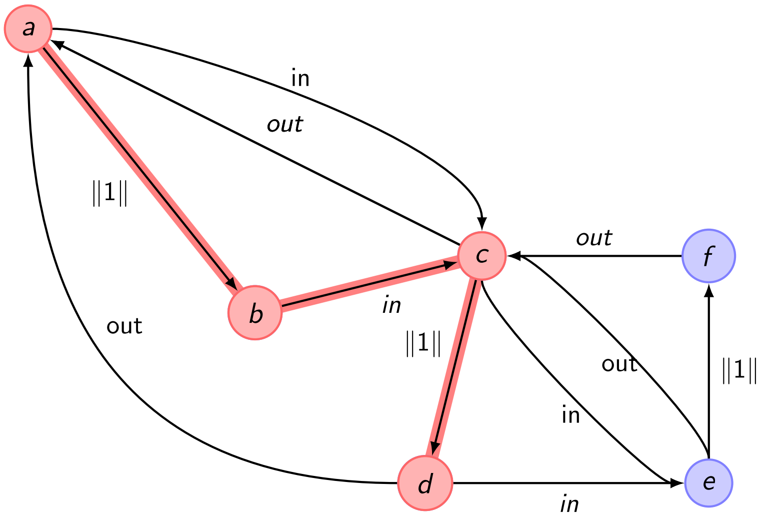
N-critical paths - two input example



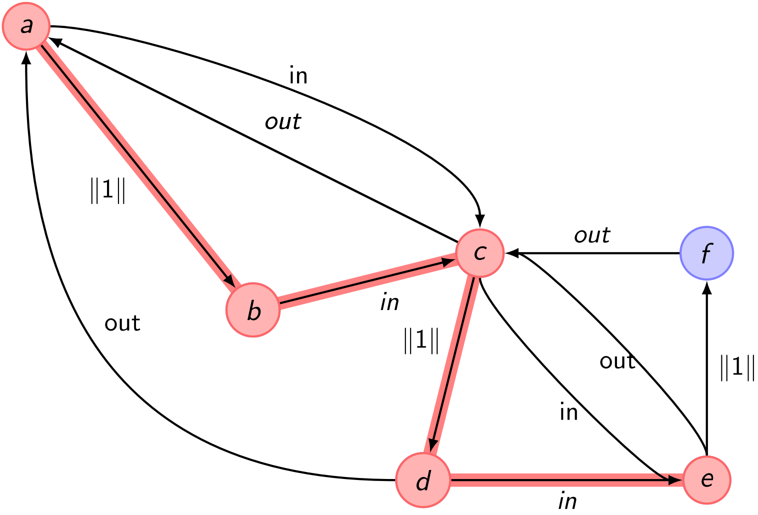
N-critical paths - two input example



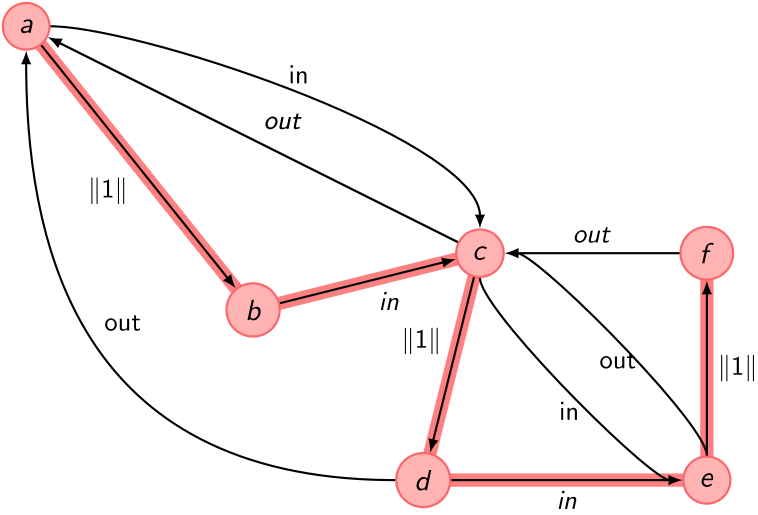
N-critical paths - two input example



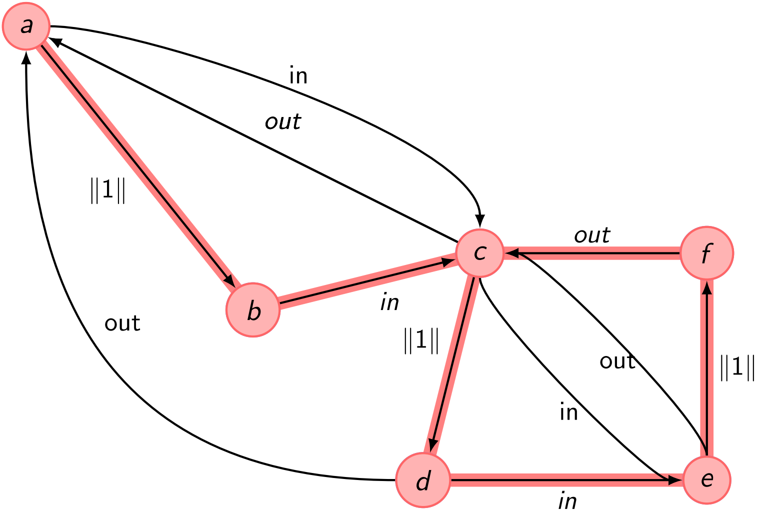
N-critical paths - two input example



N-critical paths - two input example



N-critical paths - two input example



Bad Cycle

- The average cost (*average performance*) of a cycle can be calculated as the ratio between the number of full time steps and the number of *in*'s that compound it.

Average performance

$$\frac{|FTS|}{|in|}$$

- This ratio defines the *mean* cost for an input on the cycle. The maximum mean cycle is called **bad cycle**.

Asymptotic Linearity

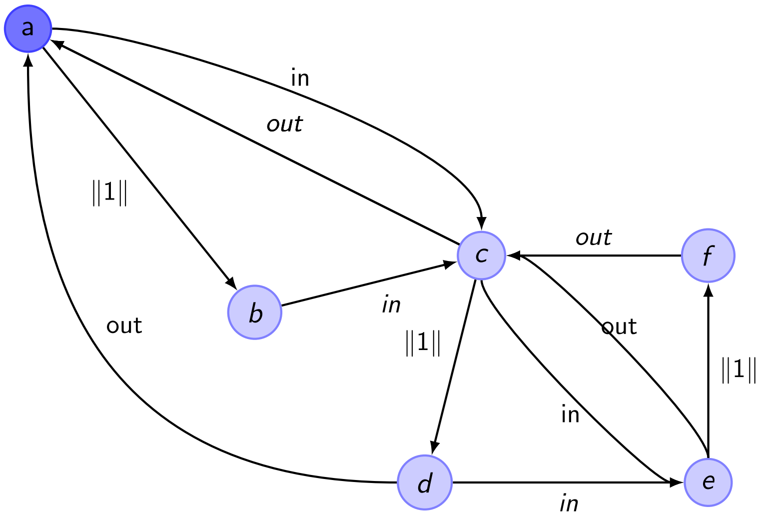
- As stated in theory, the response performance is asymptotically linear for response processes so that:

Response Performance

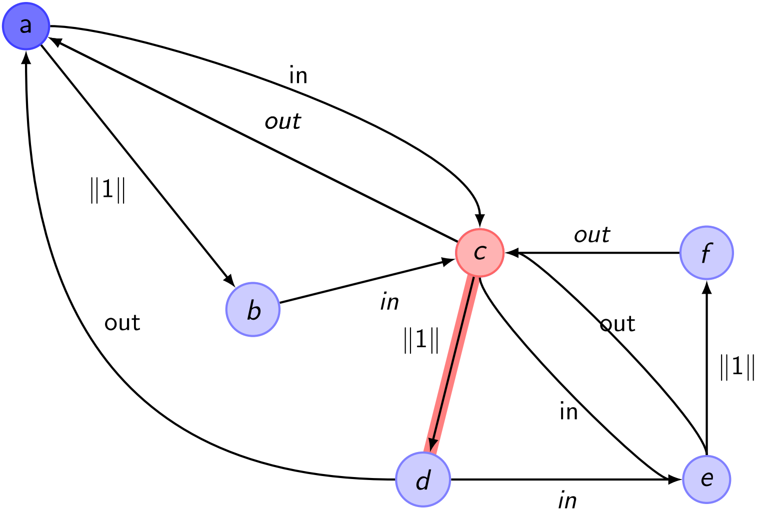
$$an - c \leq f(n) \leq an + c \quad \text{with} \quad n \in \mathbb{N}.$$

- While complete calculation can be taken out through n-critical path verification, the bad cycle can give a generic characterization of it.
- The average performance of the bad cycle corrisponds to the coefficient a of the response performance.

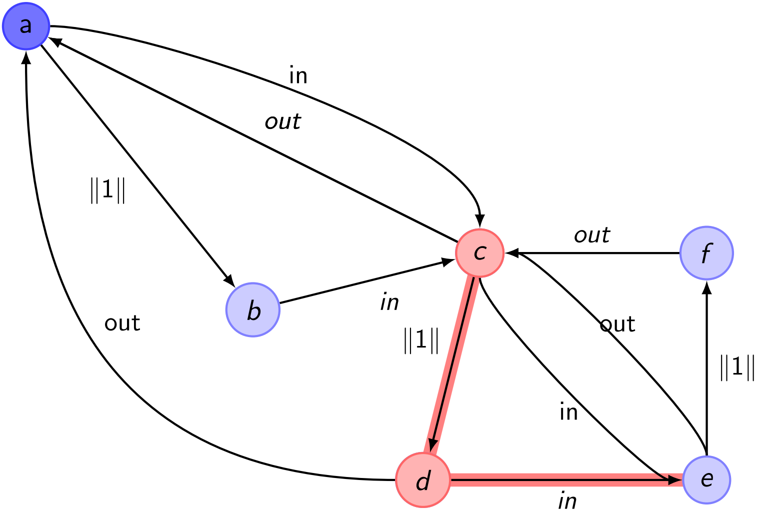
Bad Cycle - an example



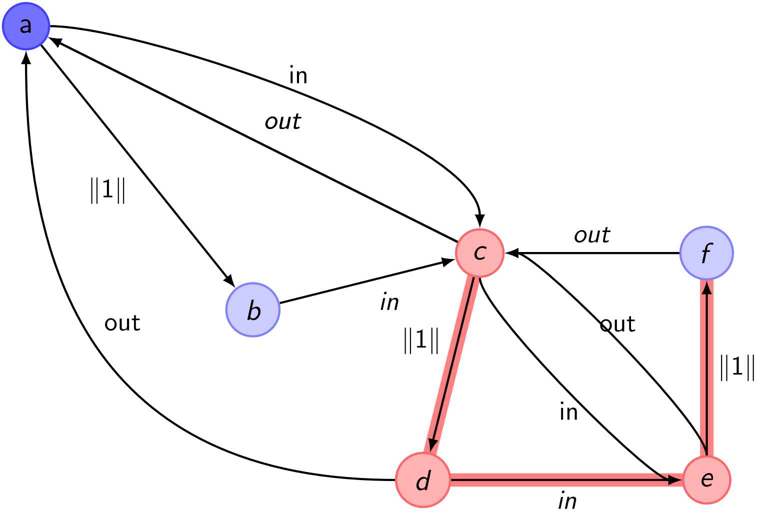
Bad Cycle - an example



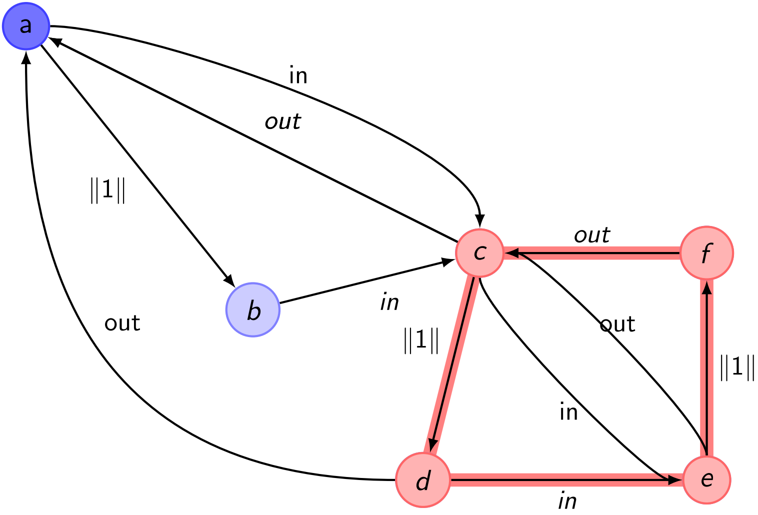
Bad Cycle - an example



Bad Cycle - an example



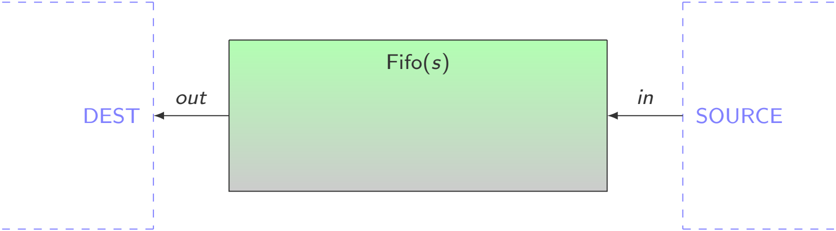
Bad Cycle - an example



FASE...

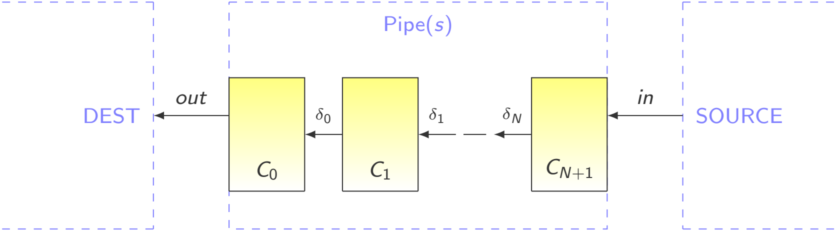
- verifies that a process is a response process.
- verifies that the process does not contain catastrophic cycle(s) (returning them if present).
- calculates the average performance of the bad cycle.
- calculates the maximum n-critical path, and so, the response performance of the process for a given number of input.
- The next few slides will review three different implementations of a buffer. We will see how FASE can be profitably used for the *comparison* of their performance.

Fifo



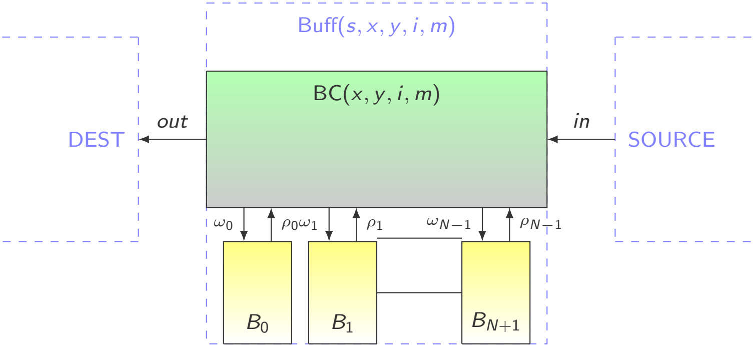
- Implements a first-in-first-out queue with capacity $N + 2$. It has no overhead and is purely sequential.

Pipe



- Implementation with concatenated and single-valued cells, of capacity $N+2$. Each cell is concatenated to the previous and the following ones and can be seen as a I/O "device"; every time a value is shifted, an internal action (*tau*) is necessarily executed.

Buff



- Implementation of $N + 2$ capacity, with indipendent and mono-valued cells. Each cell is connected to a *buffer controller* that stores a value as input and one as output, the index of the oldest value saved and the number of values stored in memory. The cells are maintained as a circular queue.

Fifo performances

	$rRTS_{nodes/edges}$	U_1	U_2	U_3	U_4	U_5	U_6	U_7
$rp_{\text{Fifo}}(3)$	9/18	2	4	6	8	10	12	14
$rp_{\text{Fifo}}(4)$	11/23	2	4	6	8	10	12	14
$rp_{\text{Fifo}}(5)$	13/28	2	4	6	8	10	12	14
$rp_{\text{Fifo}}(6)$	15/33	2	4	6	8	10	12	14
$rp_{\text{Fifo}}(7)$	17/38	2	4	6	8	10	12	14

Complexity function

2n

Pipe Performances

	$rRTS_{nodes/edges}$	U_1	U_2	U_3	U_4	U_5	U_6	U_7
$rp_{\text{Pipe}}(3)$	20/42	4	6	8	10	12	14	16
$rp_{\text{Pipe}}(4)$	48/112	5	7	9	11	13	15	17
$rp_{\text{Pipe}}(5)$	114/292	6	8	10	12	14	16	18
$rp_{\text{Pipe}}(6)$	272/759	7	9	11	13	15	17	19
$rp_{\text{Pipe}}(7)$	648/1958	8	10	12	14	16	18	20

Complexity function

2n + N - 1 (with N, equal to cells number)

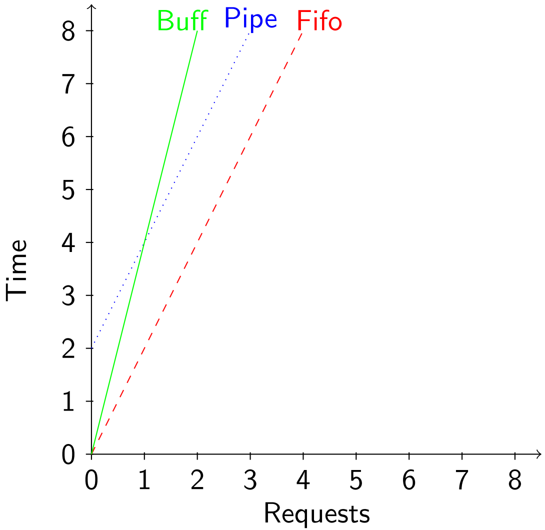
Buff performances

	$rRTS_{nodes/edges}$	U_1	U_2	U_3	U_4	U_5	U_6	U_7
$rp_{\mathrm{Buff}}(3)$	17/34	4	8	12	16	20	24	28
$rp_{\mathrm{Buff}}(4)$	48/104	4	8	12	16	20	24	28
$rp_{\mathrm{Buff}}(5)$	96/216	4	8	12	16	20	24	28
$rp_{\mathrm{Buff}}(6)$	160/368	4	8	12	16	20	24	28
$rp_{\mathrm{Buff}}(7)$	240/560	4	8	12	16	20	24	28

Complexity function

4n

Recapitulatory graph (three cells)



Final thoughts

- Our results clearly state that Fifo is *the fastest* between the considered implementation. In other words, the following preorder relations holds:

$$\text{Fifo} \sqsupseteq \text{Pipe} \sqsupseteq \text{Buff}$$

- These results are *supported* in part by the qualitative ones. Where it was possible to provide results, the formers are confirmed by the latters (see, for example Pipe and Buff or fifo and Buff).
- What about the other results?

- Our results clearly state that Fifo is *the fastest* between the considered implementation. In other words, the following preorder relations holds:

$$\text{Fifo} \sqsupseteq \text{Pipe} \sqsupseteq \text{Buff}$$

- These results are *supported* in part by the qualitative ones. Where it was possible to provide results, the formers are confirmed by the latters (see, for example Pipe and Buff or fifo and Buff).
- What about the other results?

- Our results clearly state that Fifo is *the fastest* between the considered implementation. In other words, the following preorder relations holds:

$$\text{Fifo} \sqsupseteq \text{Pipe} \sqsupseteq \text{Buff}$$

- These results are *supported* in part by the qualitative ones. Where it was possible to provide results, the formers are confirmed by the latters (see, for example Pipe and Buff or fifo and Buff).
- What about the other results?

Final thoughts

- Our results clearly state that Fifo is *the fastest* between the considered implementation. In other words, the following preorder relations holds:

$$\text{Fifo} \sqsupseteq \text{Pipe} \sqsupseteq \text{Buff}$$

- These results are *supported* in part by the qualitative ones. Where it was possible to provide results, the formers are confirmed by the latters (see, for example Pipe and Buff or fifo and Buff).
- What about the other results?

Final thoughts

- For what concern Fifo and Pipe,qualitative results clearly state that they are *not related* or in other words it is impossible to define if one is faster than the other.
- From theory we have a preposition that clearly relates the qualitative and quantitative:

Given P and Q testable, $P \sqsupseteq Q$ iff for all tests O we have $p(P||O) \leq p(Q||O), i.e. p_P \leq p_Q$

- So...what is the catch??

Final thoughts

- For what concern Fifo and Pipe,qualitative results clearly state that they are *not related* or in other words it is impossible to define if one is faster than the other.
- From theory we have a preposition that clearly relates the qualitative and quantitative:

Given P and Q testable, $P \sqsupseteq Q$ iff for all tests O we have $p(P||O) \leq p(Q||O), i.e. p_P \leq p_Q$

- So...what is the catch??

Final thoughts

- The problem lies in the tests taken into consideration. While qualitative analysis takes into account any kind of test, quantitative measuring examines only *user processes* with a well-known structure.
- This means that not all the available traces in a testing scenario (qualitative one) are available also in the other (quantitative one).
- It is easy to argue that, by narrowing the class of tests in the qualitative theory (and thus redefining the preorder operator), we can obtain the desired results.
- Besides that, all the other results demonstrate the quality of the work done up to now.