Introduction to self-stabilization

„Self-stabilization in spite of distributed control", Dijkstra 1973

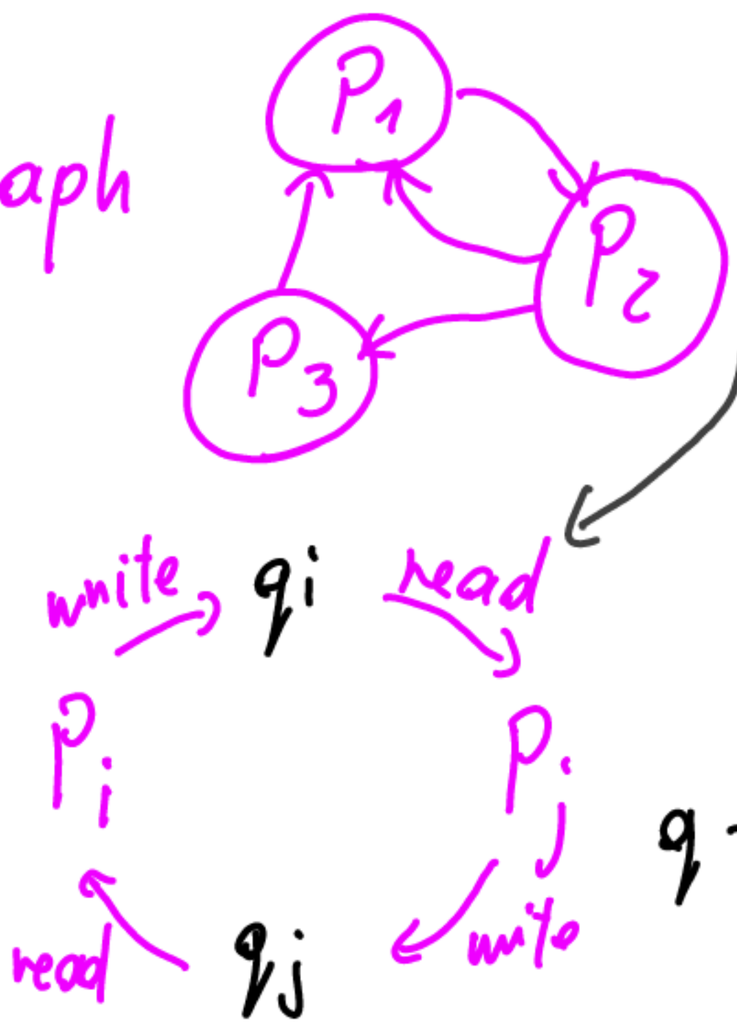
Idea: Do not consider all possible sources errors, but construct an algorithm that from any state will automatically return to a legal state.

- In '90 a lot of papers
- 2000 „Self-stabilization" book by Shlomi Dolev, Neger Dasent
- 2002 - on PODC (conference about distributed computing) They announced that Dijkstra paper most influenced paper.
- Lemiesz recomends: „Introduction to distributed algorithms" G. Tel, 2012, Combridge.

Model (short summary)

1. Distributed system consisted of independent machines/processes.
   $P_1, P_2, \dots P_n$.

2. Communication graph



3. Configuration ≡ a set of all variables, registers, queques of all processes in a given moment (snapshot of the system)
   e.g. $C = (s_1, s_2, \dots s_n, r_1, r_2, \dots r_n)$. s_i - variables of P_i.

4. Execution ≡ a sequence of configurations, e.g. $e = (c_1, c_2, \dots)$

5. Specification S ≡ desired behaviour of algorithm on the sequence of configuration.
   e.g. in each configuration at most one proces can have access to the critical section.

6. Asynchronus model ≡ processes computation time and messages delivening time are non-deterministic
   (from one C we can expect many Cs)

7. Central Deamon ≡ adversary who maliciously decides which of possibilities will occur to prevent reaching specification S.

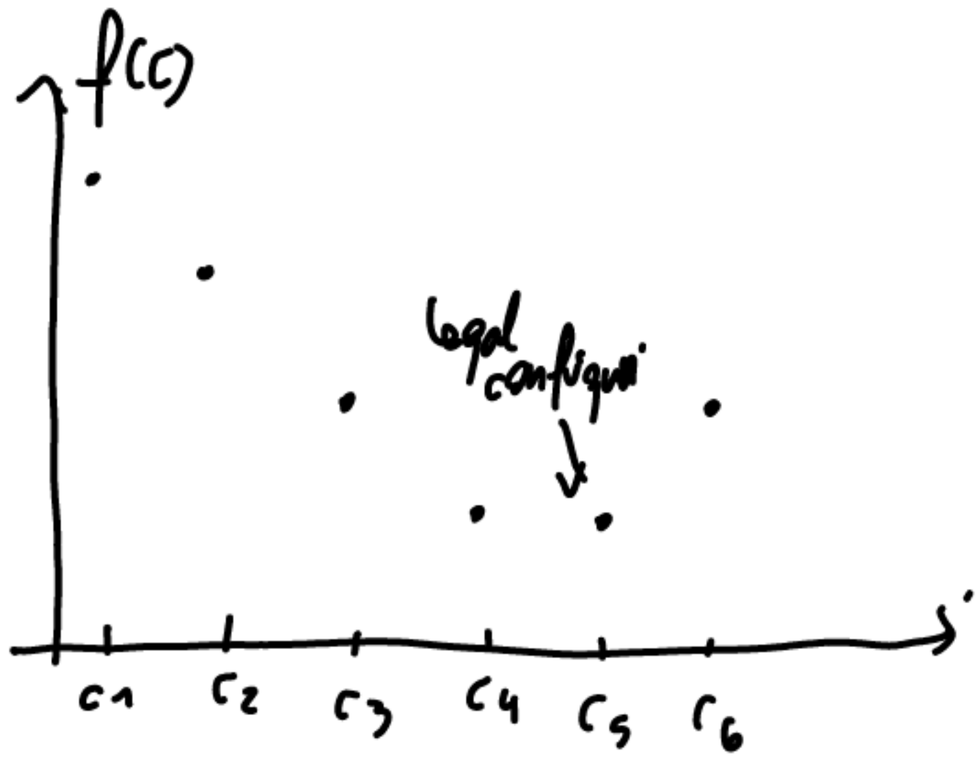
Def (self-stabilization) S - specification, C - set of all possible configuration.
L ⊆ C - set of legal configuration

We say that algorithm self-stabilizies to S if:

1. Connectness: each execution that stants in $c \in L$ meets specification S.
2. Convergence: each execution finaly reaches some legal configuration $c \in L$.

A typical way of proving self-stabilization.

1. Connectness: we need to check that $(\forall c \in L)(\forall e = (c, \ldots))(\text{„}e \text{ meets } S\text{"})$
2. Convergence: there is a potential function $f : C \to W$, where W is well-ordered set (linear order + the smallest element in each subset)
   a) for each move $c_i \to c_j$ we have $f(c_i) > f(c_j)$ or $c_j \in L$
   b) if c_i is terminal (i.e. no moves from c_i) then $c_i \in L$.



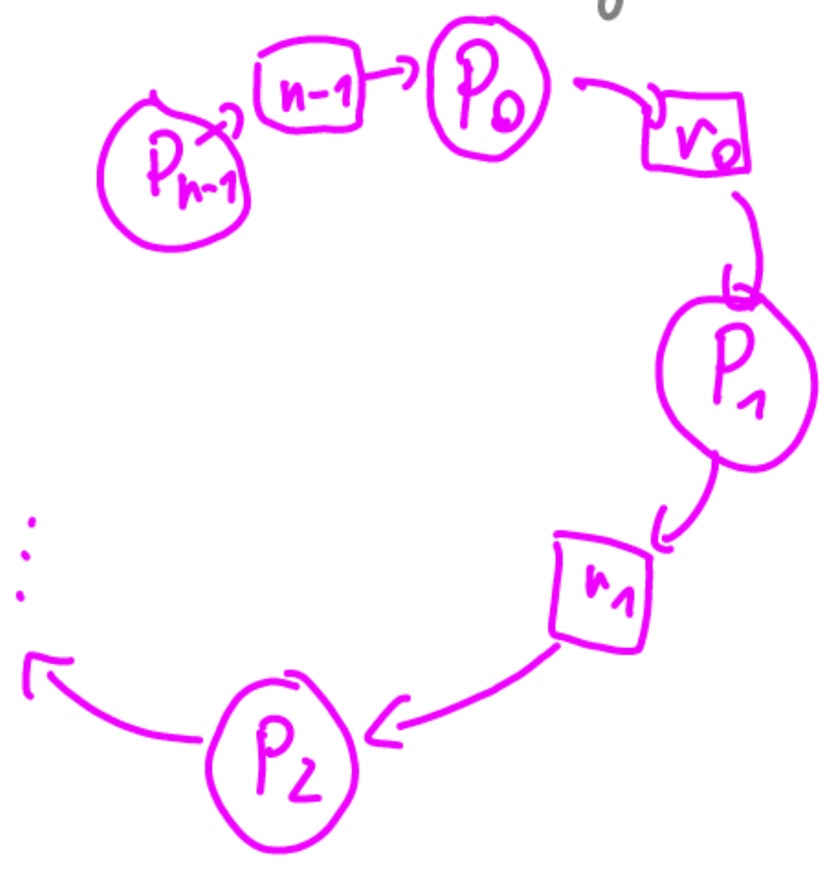
„Mutual Exclusion" by Dijkstra

Specification S

(S1) In each configuration at most one process can be in critical section (CS)

(S2) each process is in CS infinitely many times.

Dijkstra solution: „token ring" (1973, p-f in 1992)



when you are in CS you have access to the printer

P_0:

do forever

| if $r_0 == r_{n-1}$:

| | // access to CS

| | $r_0 \leftarrow (r_0 + 1) \bmod (n+1)$

$P_{i \neq 0}$:

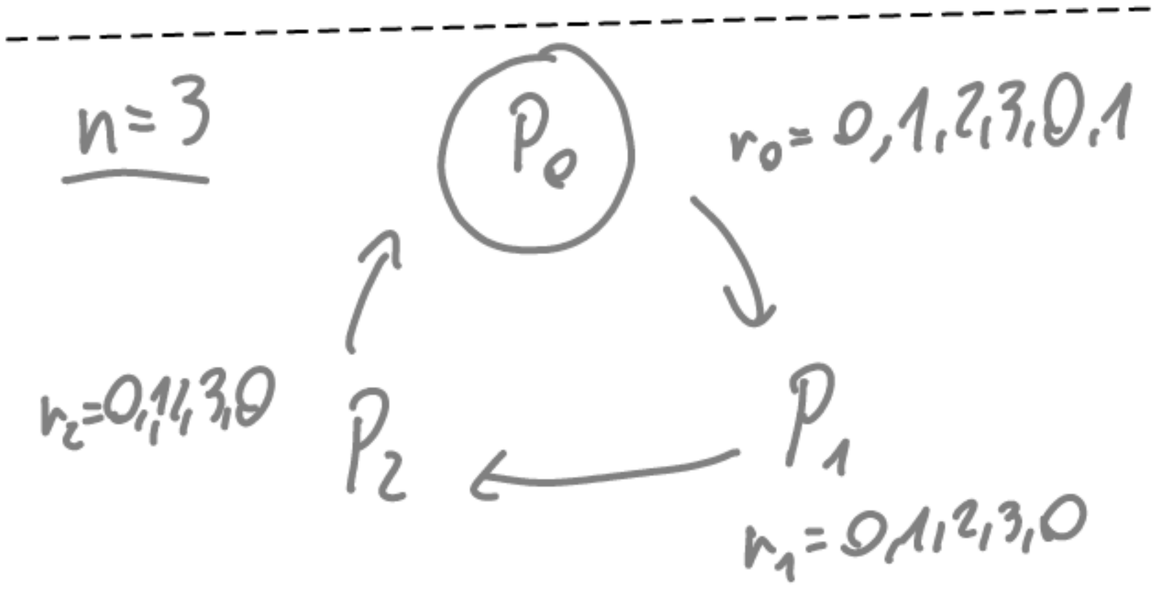
do forever

| if $r_i \neq r_{i-1}$:

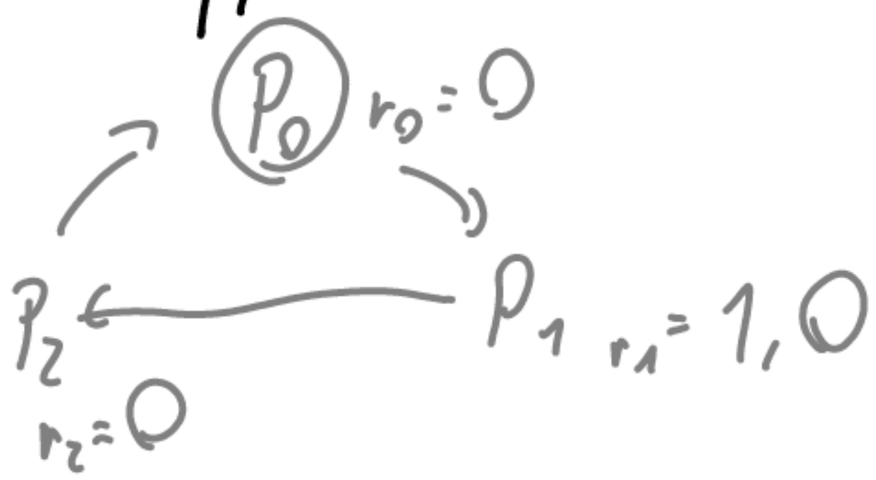
| | // ...

| | $r_i \leftarrow r_{i-1}$

Example $n = 3$



Something bad happens



Lemma 1 $(\forall_{C \in C})(\exists_i)(P_i$ has access to CS)

There is always someone who can move, no terminal configuration.

P-f Assume indirectly that none of the processes can access CS.

Then $r_0 == r_1 == r_2 == r_3 \ldots == r_{n-1} \wedge$ (P_0 can not access $\Rightarrow$) $r_0 \neq r_{n-1}$

↯

Lemma 2 No step of the algorithm increases the number of processes in CS.

P-f A process can get access to critical section only when its predecessor changes its register, and thus looses the access. □

Lemma 3 (Connectness in definition of self-stabilization)

L - set of legal configurations (i.e. only 1 process in the CS)

Then $(\forall_{c \in L})(\forall_{e = (C_1 \ldots)})($„e meets S")

P-f S1: Let in $c_1 \in L$ only 1 process in CS. (in def we have at most) but from Lema 1

From lemma 1 and 2 we know that in config c_2 access to CS.

has P_{i+1} - Ef$_{C.}$ $c_2 \in L$ (P_{i+1} in CS) $\rightarrow$ $c_3 \in L$ (P_{i+2} in CS) $\rightarrow$

S2: From S1 we know that each process is in CS once in n consecutive configurations.