Progress-Space Tradeoffs in Single-Writer Memory Implementations

Damien Imbs[1,2], Petr Kuznetsov[3] and Thibault Rieutord[3]

[1] LIF, Aix-Marseille Université & CNRS, France
[2] Bremen University, Germany
[3] LTCI, Télécom ParisTech, Université Paris Saclay

18^{th} of December 2017

The Comparison-Based Model

Processes comes from a large set of potential participants.

- Only n of them may participates.
- Have unique identifiers from a large set (MAC,IP,RFID...).

Can differentiate themselves only by using identifiers relative order.

Not suited for an SWMR memory:

- Possibly requires infinitely many registers.
- Need *a priori* allocation of processes to registers (renaming).

Natural to assume access to a MWMR memory.

Asynchronous SWMR Implementations

Processes are attributed access to two operations:

- *Write*(*Val*): Replace the content of a *private* abstract memory location with value *Val*.
- *Collect*(): Returns the content of the abstract memory location of all participating processes.

- Can be used to execute SWMR higher-level algorithms such as renaming[ABDPR90] or atomic snapshots[AADGMS93].
- A stable storage primitive, not as MWMR registers which may inadvertently overwrite other processes written values.

Progress Conditions

Wait-freedom[H91]

Every correct process makes progress.

k-lock-freedom[BG15] (k-non-blocking)

At least k out of the correct processes make progress.

k-obstruction-freedom[HLM03, Tau09]

If there are at most k correct processes, all of them make progress.

k-lock-freedom is strictly stronger than k-obstruction-freedom.

SWMR Implementations in MWMR memory

Related Work[DFGR15,DFGL13]:

- Space optimal lock-free SWMR implementation (n registers).
- Wait-free SWMR implementation ($2n-1$ registers).
- Adaptive SWMR implemenations ($\Theta(p)$ registers).

Contributions:

- k-lock-free SWMR implementation ($n+k-1$ registers).
- Impossibility of a 2-obstruction-free SWMR implementation using n registers.

$\Rightarrow$ Space optimal 2-OF/LF SWMR implementation (n+1 registers).

Lock-Free SWMR Implementation[DFGR15] (1/2)

Register content stability:

- Write operations as triplets (Val, Id, $Counter$).
- Full information: Update registers with all known triplets.
- Maintain visibility: Take a snapshot before each update.

$\Rightarrow$ The content of a register is persistent if not poised to be written.

Read SWMR operations:

- Returns most recent write values present in a snapshot.

$\Rightarrow$ Returns last persistent write value or a more recent one.

Lock-Free SWMR Implementation[DFGR15] (2/2)

Write SWMR operations:

- Update all registers in a round robin fashion.
- Terminate when the value is present in all n registers

Safety.

Other processes can cover at most $n - 1$ registers at once:
$\Rightarrow$ A terminated write operation is persistent.

Lock-freedom. Assume it is not the case, then:

1. Eventually all registers contains pending write operations.
2. One register is persistent infinitely often.
3. Persistent content is eventually propagated to all n registers.

Generalization to k-Lock-Freedom

k-Lock-free write SWMR operations:

- Count processes with new write operations: *ActiveCount*.
- Upate the first $n - 1 +$ *ActiveCount* registers.
- Reset *ActiveCount* (to 1) at each new write operation.

Generalization to k-Lock-Freedom

k-Lock-free write SWMR operations:

- Count processes with new write operations: *ActiveCount*.
- Upate the first $n-1+$ *ActiveCount* registers.
- Reset *ActiveCount* (to 1) at each new write operation.

Helping:

Write operations seen by live processes are eventually persistent.

k-lock-freedom with $n+k-1$ registers:

1. The $j > 0$ live processes eventually update only the $n+j-1$ first registers.
2. If $j < k$, the $(n+j)^{th}$ register is written infinitely often by and only by correct blocked processes.

$\Rightarrow$ If so, a live process sees such a blocked write operation.

Lower Bound Preliminaries

Indistinguishability:

A set of configurations $\mathcal{D}$ are indistinguishables for a set of processes P, denoted $I(\mathcal{D}, P)$, if the content of all registers and the state of all processes in P are identical in all configurations in $\mathcal{D}$.

$\rightarrow$ Processes behave identically in indistinguishable executions.

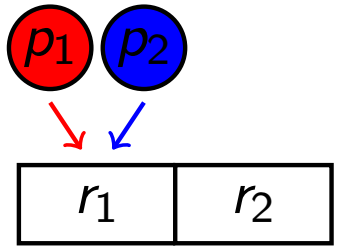
Covering:

A set of registers S is covered by a set of processes P in a configuration C, denoted $Cover(S, P, C)$, if, for all register $r \in S$, there is a process $p \in P$ poised in C to execute a write on r.

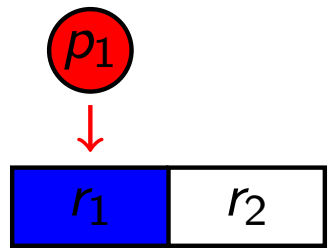
$\rightarrow$ Processes must write an uncovered register for persistence.

Impossibility of 2-process Wait-free SWMR Implementation with 2 Registers

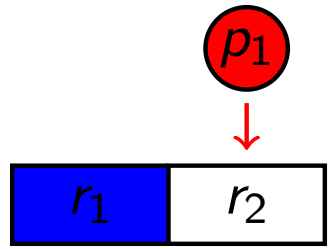
In the comparison-based model, processes write first to the same register. Let them be poised to write to this register r_1:



1. Let C_1 be the configuration reached by letting p_2 write r_1:



2. Let C_2 be the configuration reached by letting p_1 run until it becomes poised to write to r_2 and then we let p_2 write r_1:



$\rightarrow$ p_2 cannot distinguish C_1 and C_2. In a solo-execution p_2 must write r_1 and r_2 and thus can hide steps of p_1.

Confusion

P is *Confused* on S in $\mathcal{D}$, denoted $\mathit{Confused}(P, S, \mathcal{D})$, if:

1. All configurations in $\mathcal{D}$ are indistinguishable to P: $I(\mathcal{D}, P)$.
2. $|S| + |P| = n + 1$.
3. Processes not in P might cover 2 registers in $\mathcal{D}$ "independantly of other processes":

$$\forall p \in \Pi \setminus P, \exists r_p, r'_p \in S, \forall D \in \mathcal{D}, \exists r \in \{r_p, r'_p\} : \mathit{Cover}(\{r\}, \{p\}, D) \wedge$$

$$(\exists D' \in \mathcal{D}, I(\{D, D'\}, \Pi \setminus \{p\}) \wedge \mathit{Cover}(\{r_p, r'_p\} \setminus \{r\}, \{p\}, D')).$$

4. Any strict subset of S is covered by $\Pi \setminus P$ in an execution of $\mathcal{D}$:

$$\forall R \subsetneq S, \exists D \in \mathcal{D} : \mathit{Cover}(R, \Pi \setminus P, D).$$

Simple Results

A confusion on $\mathcal{R}$ implies the impossibility of 2-OF SWMR implementation.

1. Any strict subset of $\mathcal{R}$ migth be covered, thus the confused process must write to all registers for a new SWMR write.
2. Hence, steps of other processes can be hidden infinetely often.

Simple Results

A confusion on $\mathcal{R}$ implies the impossibility of 2-OF SWMR implementation.

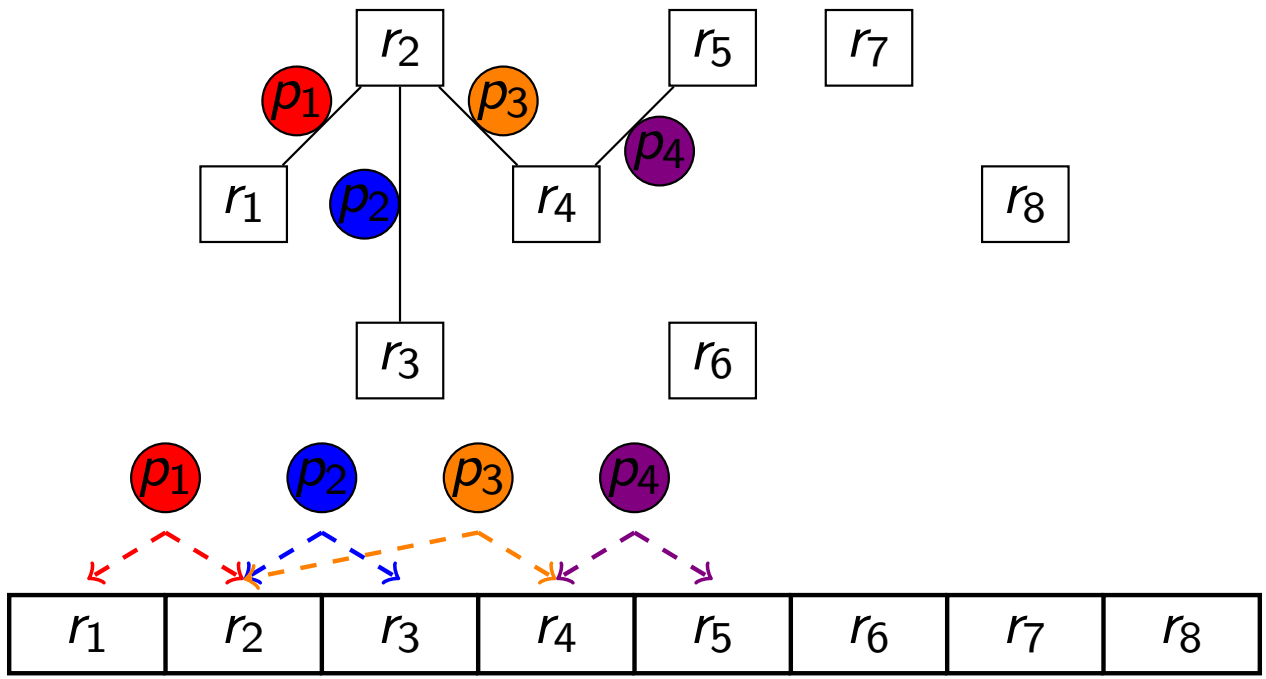
1. Any strict subset of $\mathcal{R}$ migth be covered, thus the confused process must write to all registers for a new SWMR write.
2. Hence, steps of other processes can be hidden infinetely often.

There is a reachable confusion on two registers.

1. Processes first write to the same register.
2. A process can be hidden in an execution leading to cover another register.

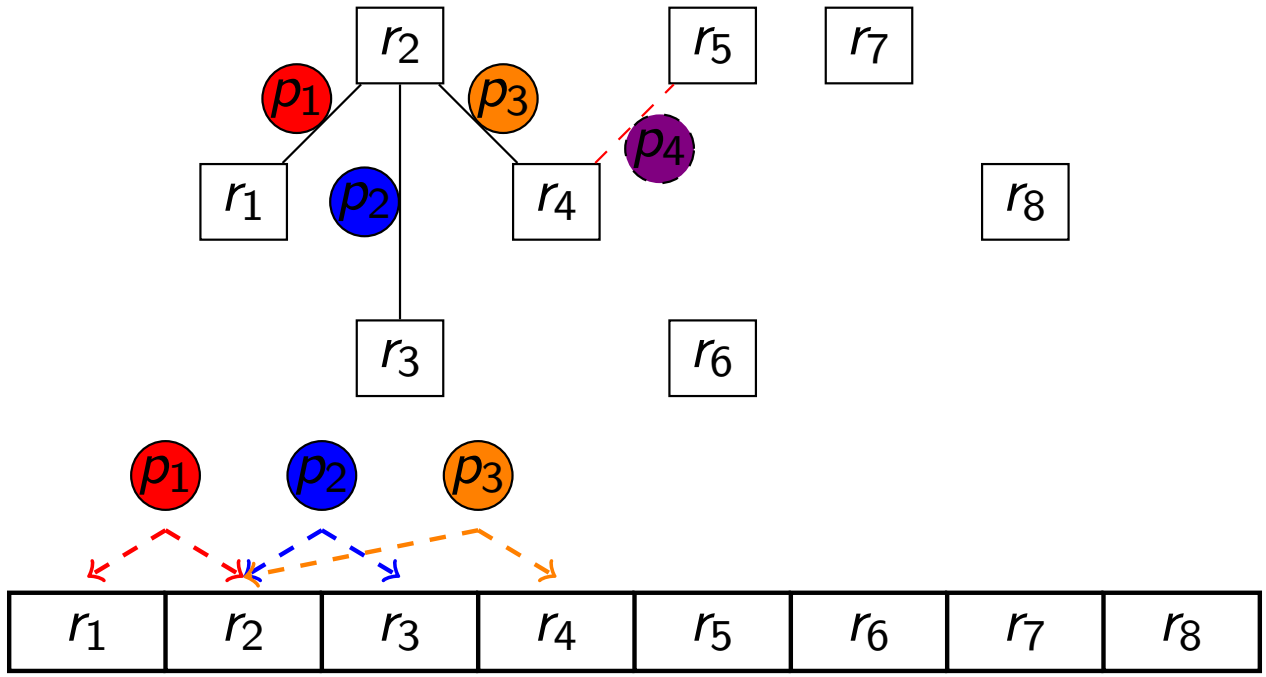
Graph Representation of a Confusion

- Registers are represented as nodes.
- Processes in $\Pi \setminus P$ which may cover two registers are represented as edges.



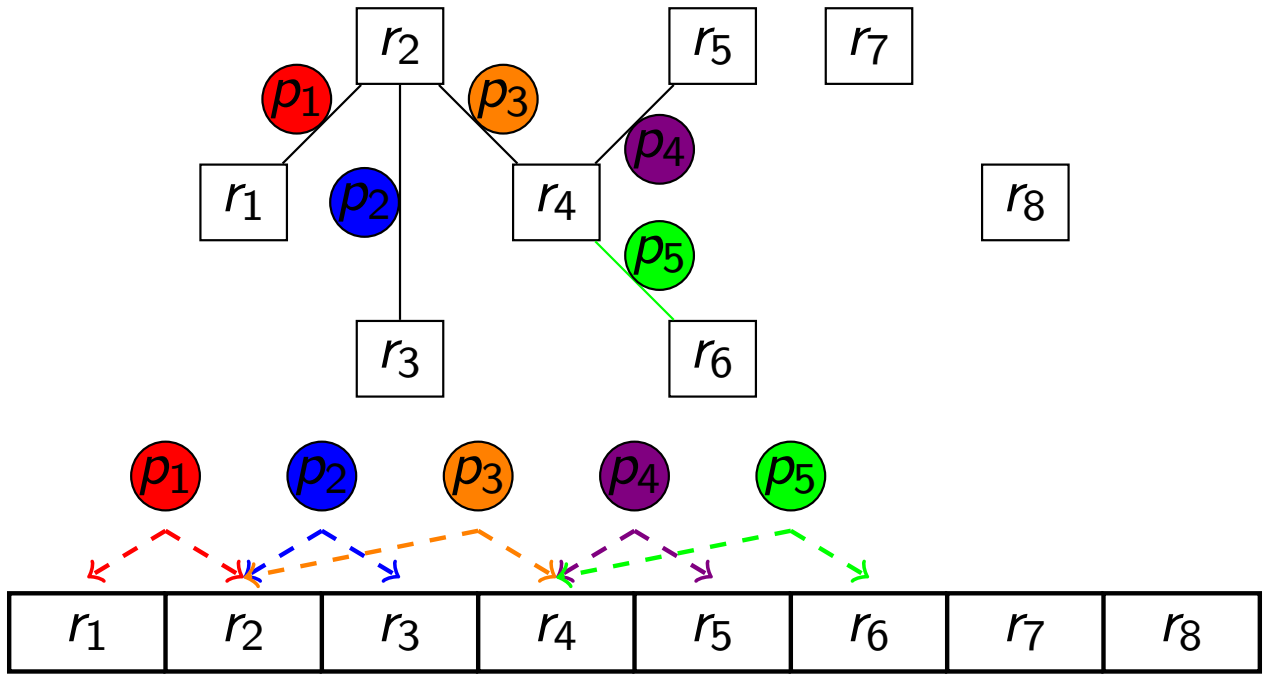
A confusion on S is a spanning tree over S (with some reciprocity).

Downgrading Confusion



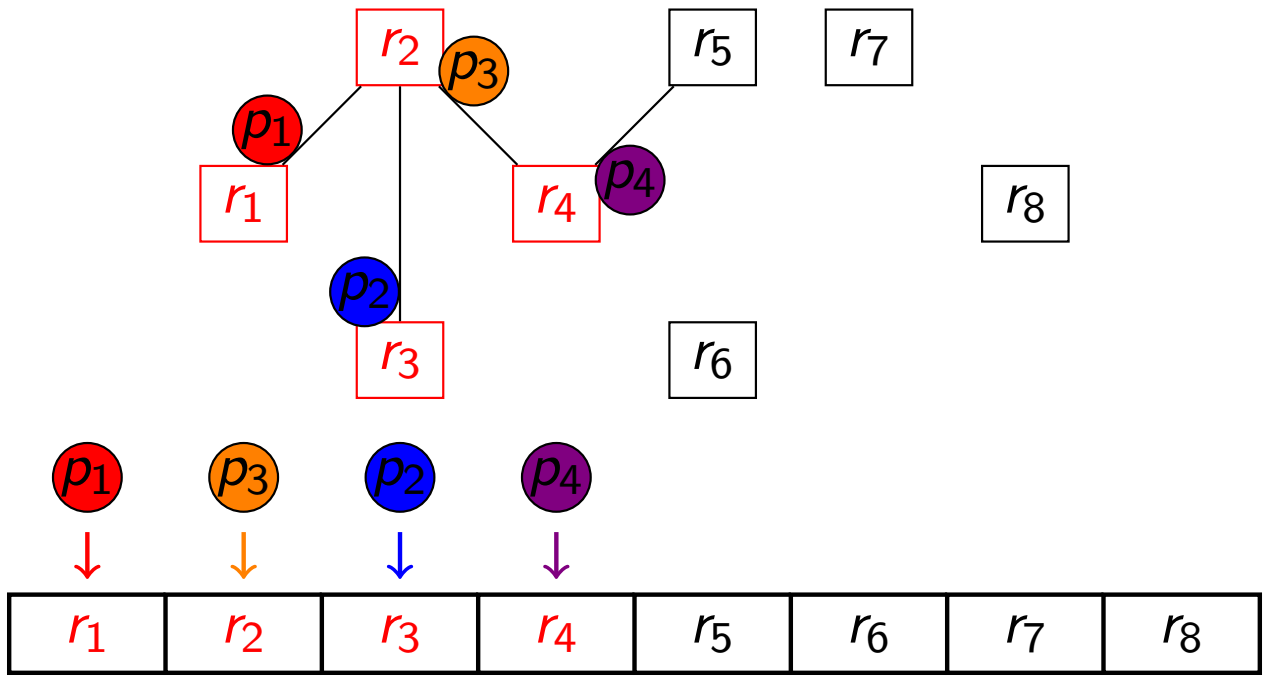
Upgrading Confusion

Given two $\{p_5, p_6, p_7, p_6\}$-only executions, applicable to any confusion's configuration, indistinguishable to $\Pi \setminus \{p_5\}$ in which p_5 covers r_4 and r_6 respectively.



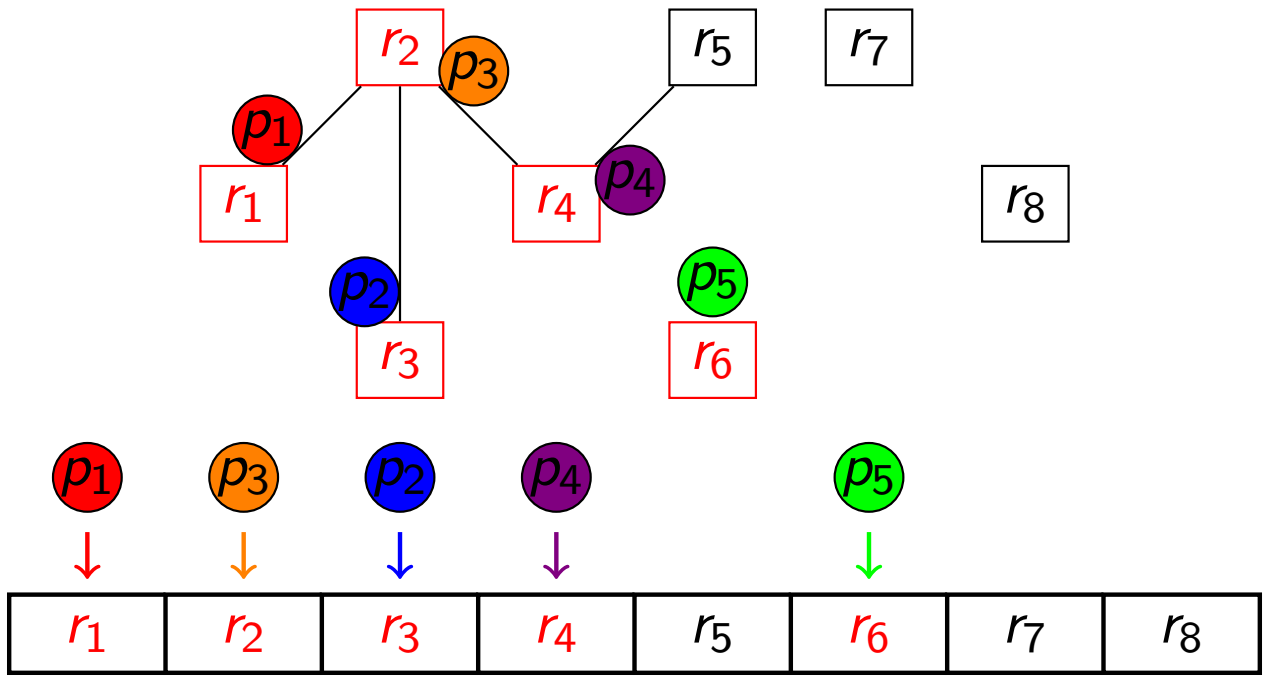
Induction Step (1^{st} Phase): Covering the Rest

1. Consider some covering of S less one register.



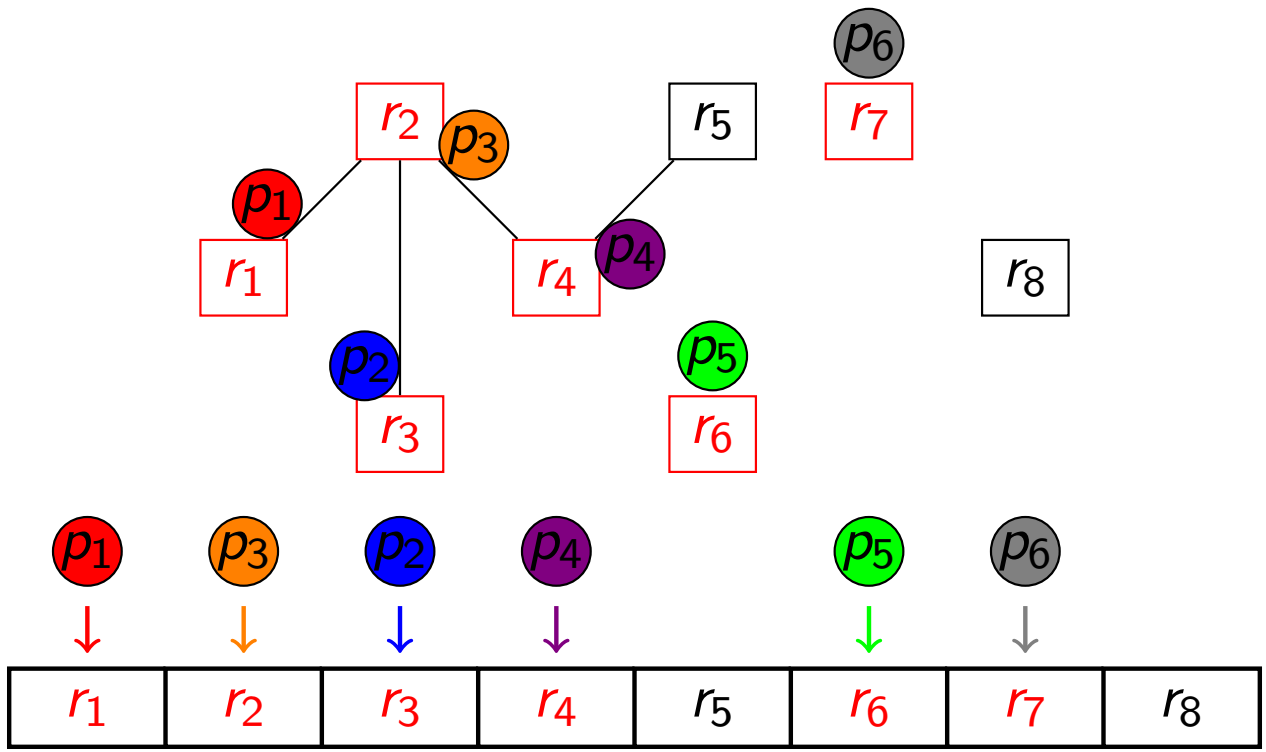
Induction Step (1^{st} Phase): Covering the Rest

1. Consider some covering of S less one register.
2. Let confused processes cover an uncovered register one by one.



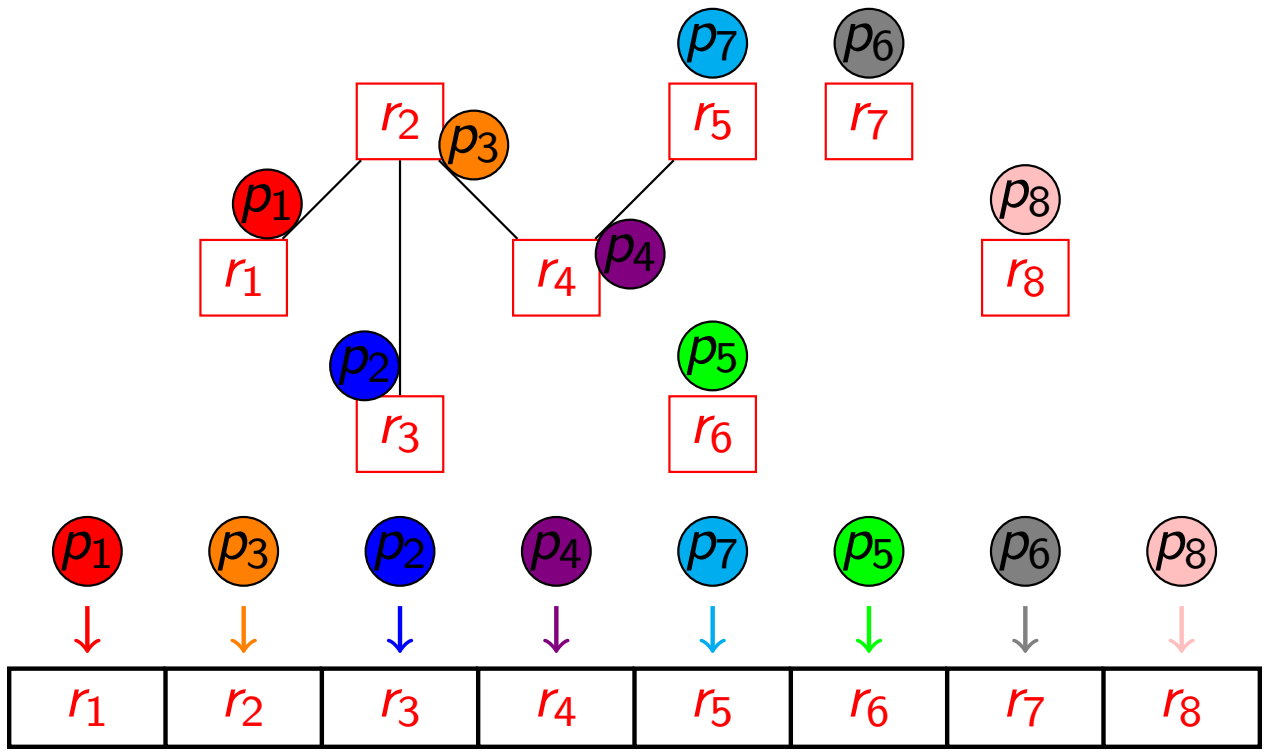
Induction Step (1^{st} Phase): Covering the Rest

1. Consider some covering of S less one register.
2. Let confused processes cover an uncovered register one by one.

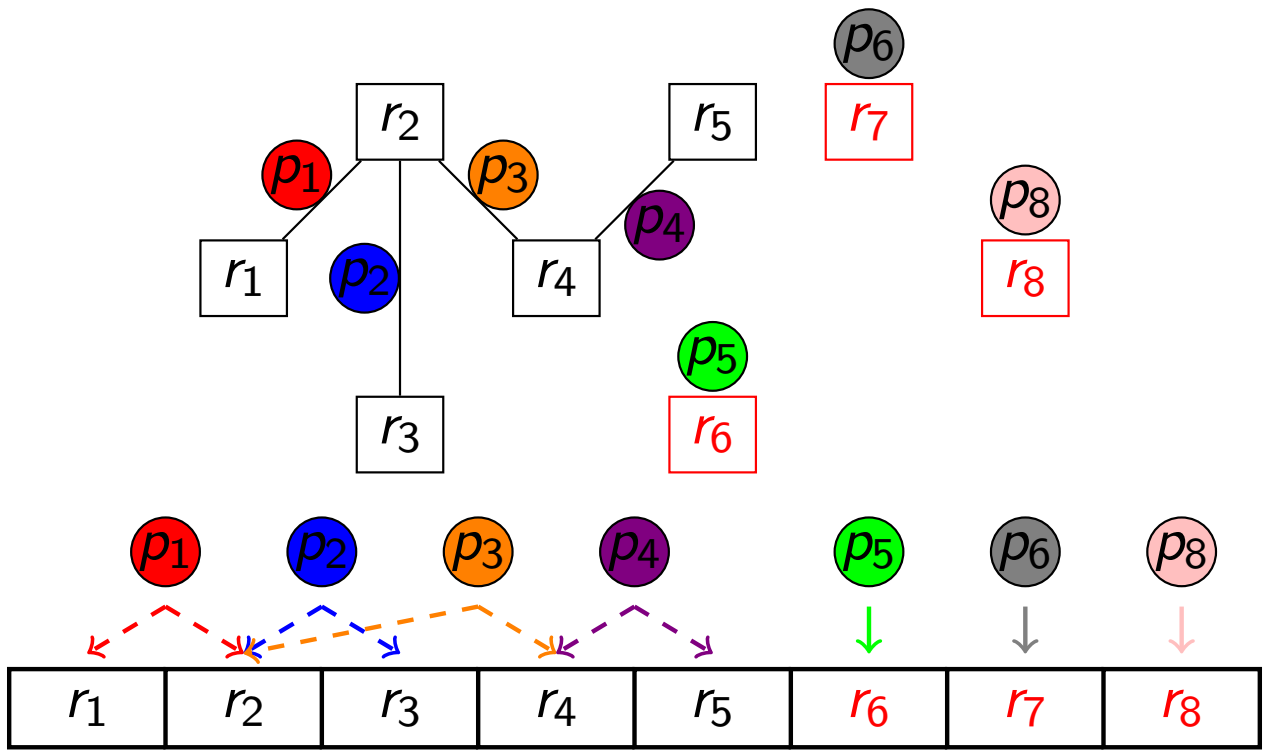


Induction Step (1^{st} Phase): Covering the Rest

1. Consider some covering of S less one register.
2. Let confused processes cover an uncovered register one by one.

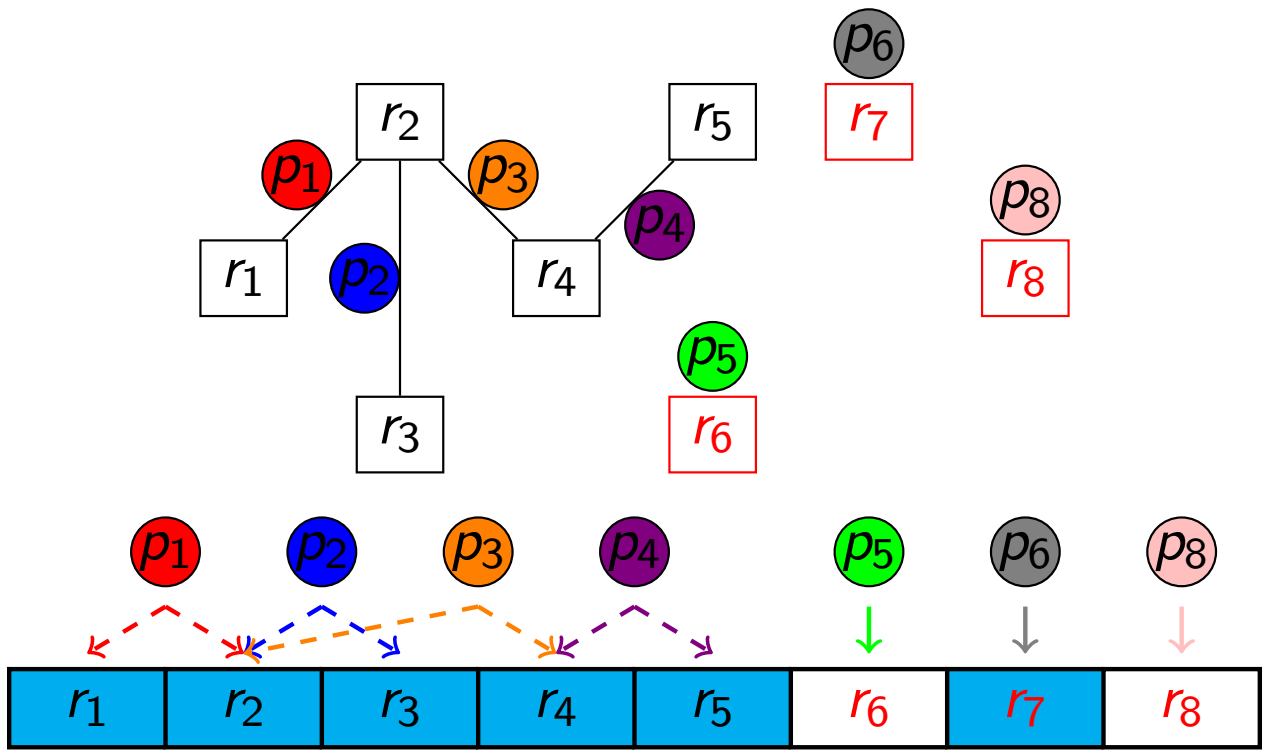


Induction Step (2^{nd} Phase): Alternative Confusion



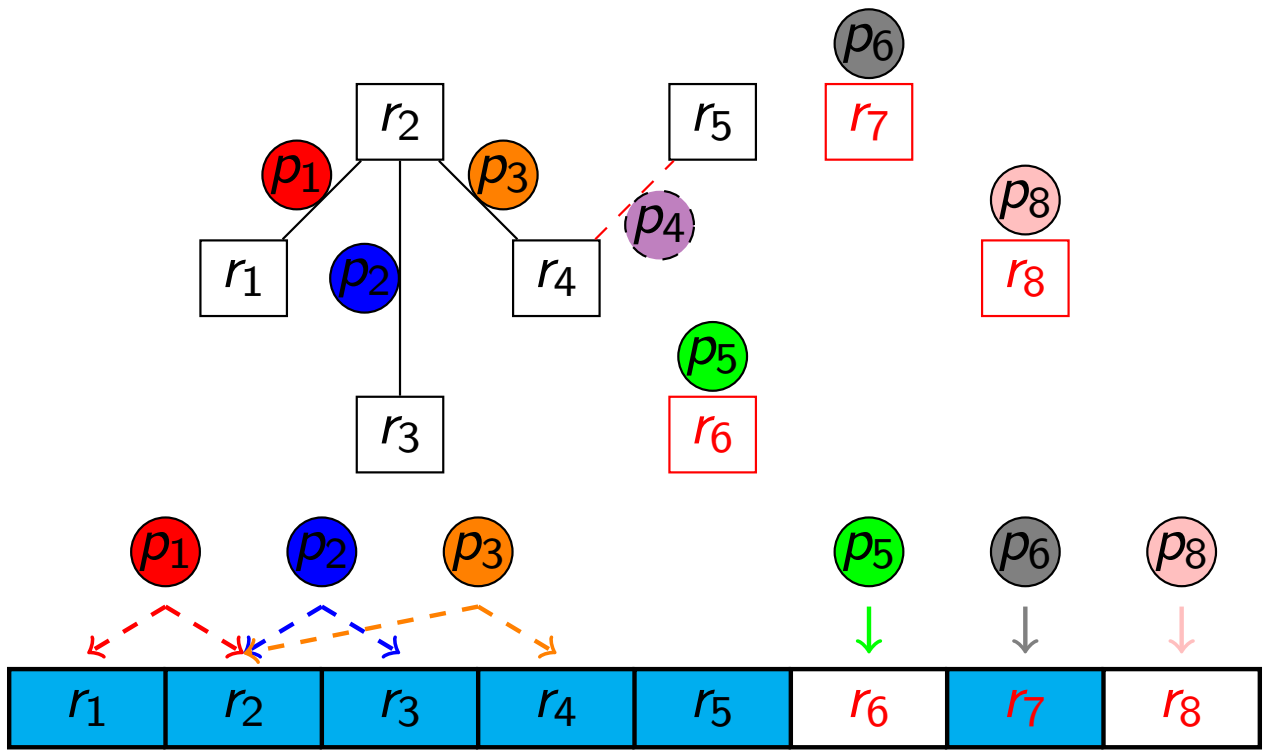
Induction Step (2^{nd} Phase): Alternative Confusion

1. p_7 must write registers in $\{r_1, r_2, r_3, r_4, r_5\}$ infinitely often.



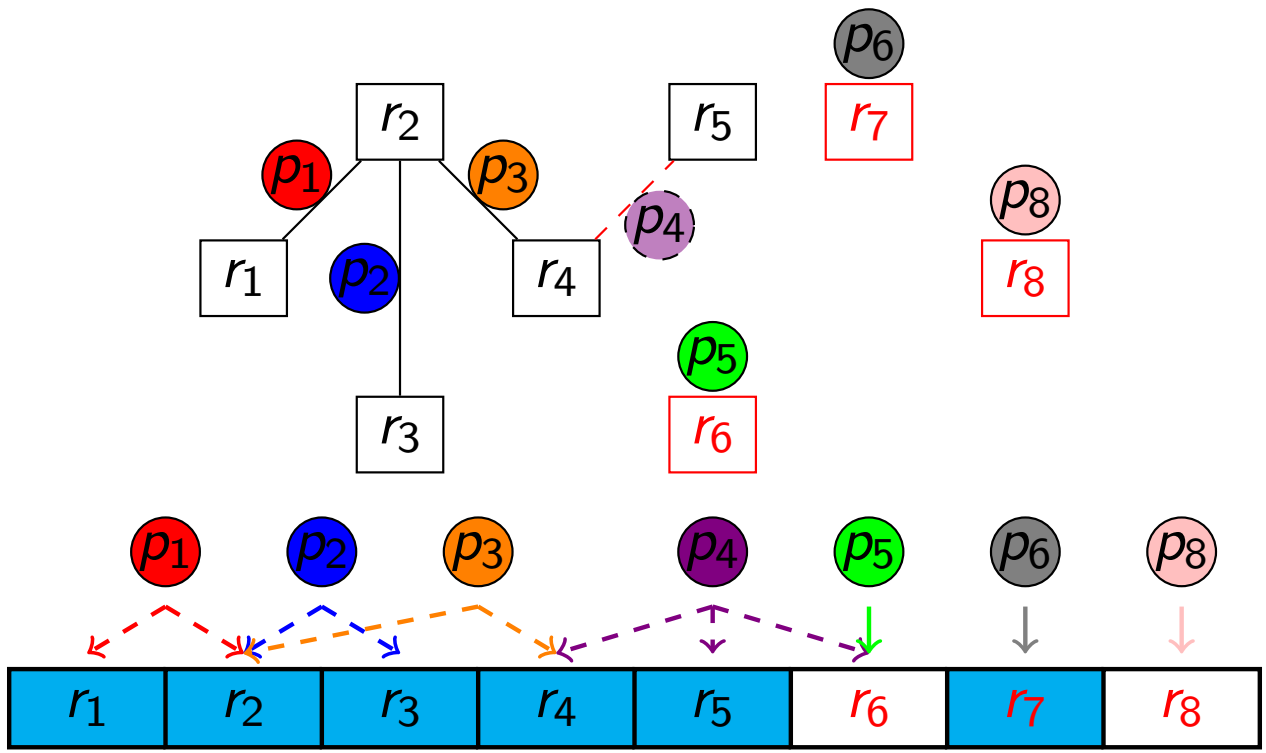
Induction Step (2^{nd} Phase): Alternative Confusion

1. p_7 must write registers in $\{r_1, r_2, r_3, r_4, r_5\}$ infinitely often.
2. Select a degraded confusion (removing r_5 and adding p_4).



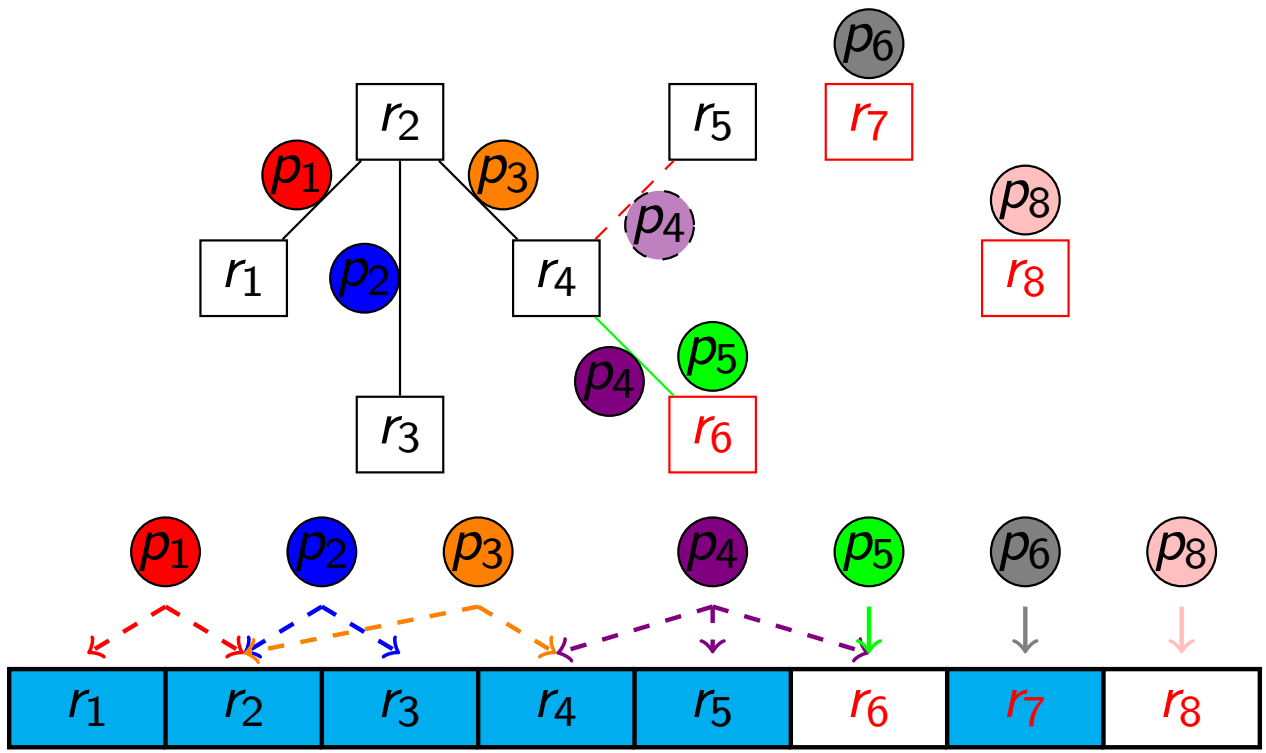
Induction Step (2^{nd} Phase): Alternative Confusion

1. p_7 must write registers in $\{r_1, r_2, r_3, r_4, r_5\}$ infinitely often.
2. Select a degraded confusion (removing r_5 and adding p_4).
3. p_4 executed hidden by p_7 eventually cover a register $\notin S$ (r_6).

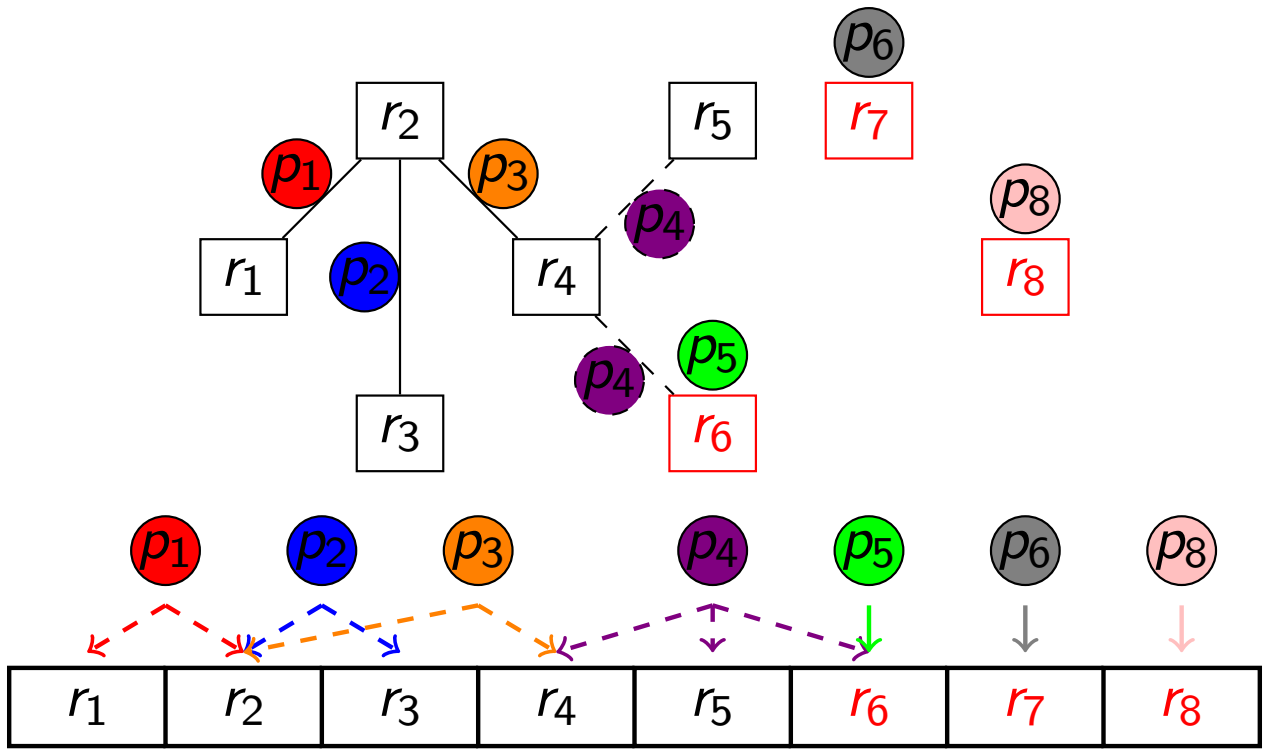


Induction Step (2^{nd} Phase): Alternative Confusion

1. p_7 must write registers in $\{r_1, r_2, r_3, r_4, r_5\}$ infinitely often.
2. Select a degraded confusion (removing r_5 and adding p_4).
3. p_4 executed hidden by p_7 eventually cover a register $\notin S$ (r_6).
4. $\{p_5, p_6, p_7, p_8\}$ are now also confused on $\{r_1, r_2, r_3, r_4, r_6\}$.

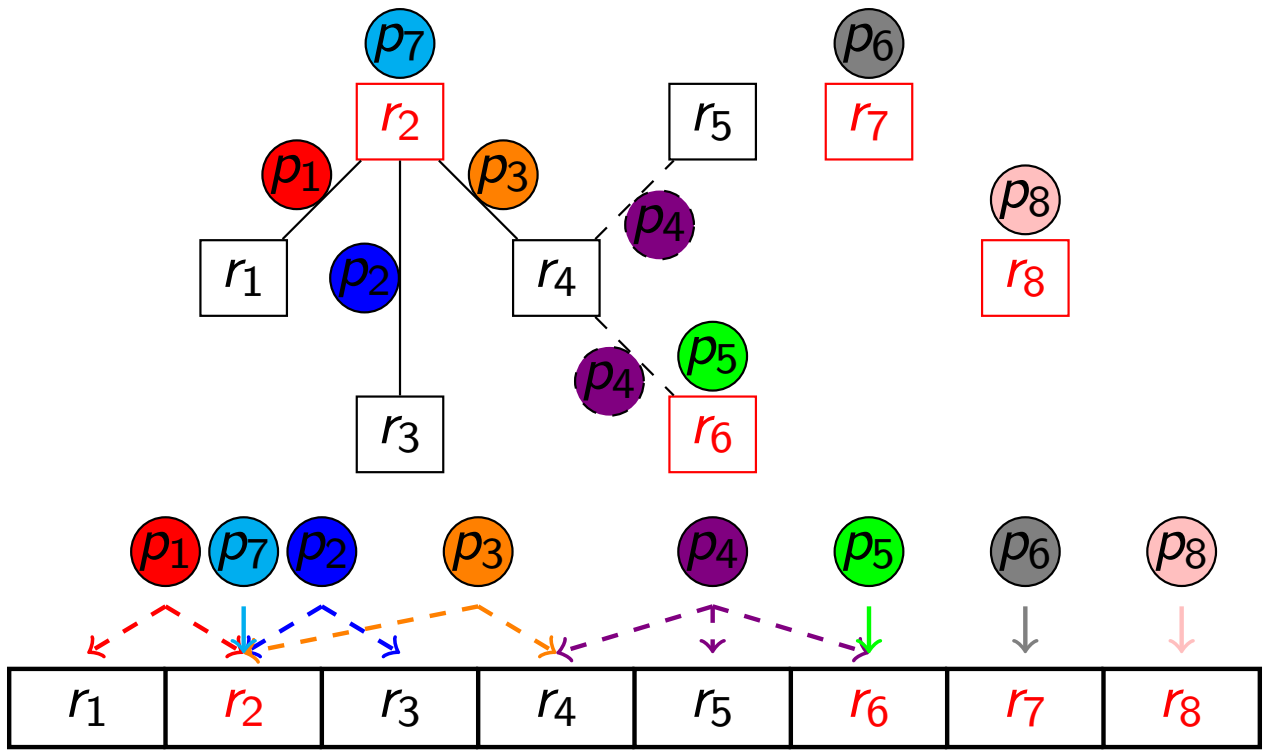


Induction Step (3^{rd} Phase): Increasing Confusion



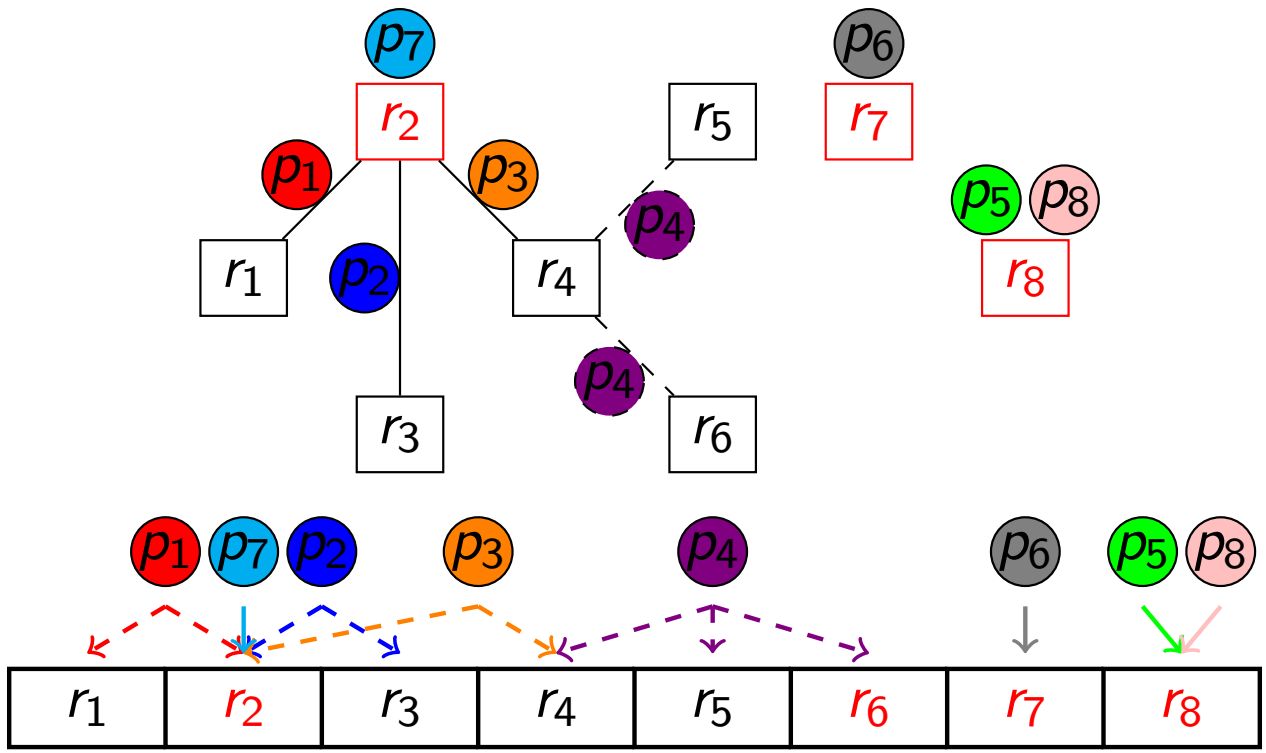
Induction Step (3^{rd} Phase): Increasing Confusion

1. Let p_7 cover some register in $\{r_1, r_2, r_3, r_4\}$.



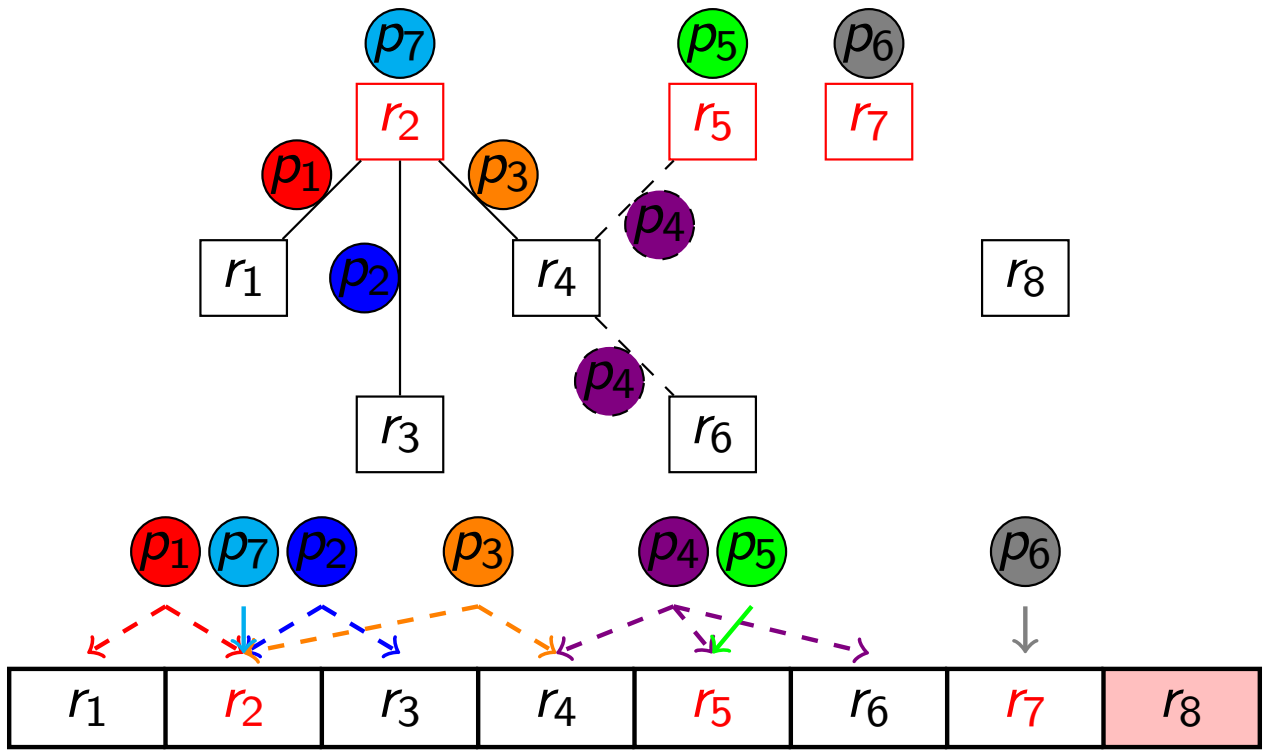
Induction Step (3^{rd} Phase): Increasing Confusion

1. Let p_7 cover some register in $\{r_1, r_2, r_3, r_4\}$.
2. Let p_5 run until it covers some register $\neq r_5$ for the last time:
   1. p_5 covers a register in $\{r_7, r_8\}$.



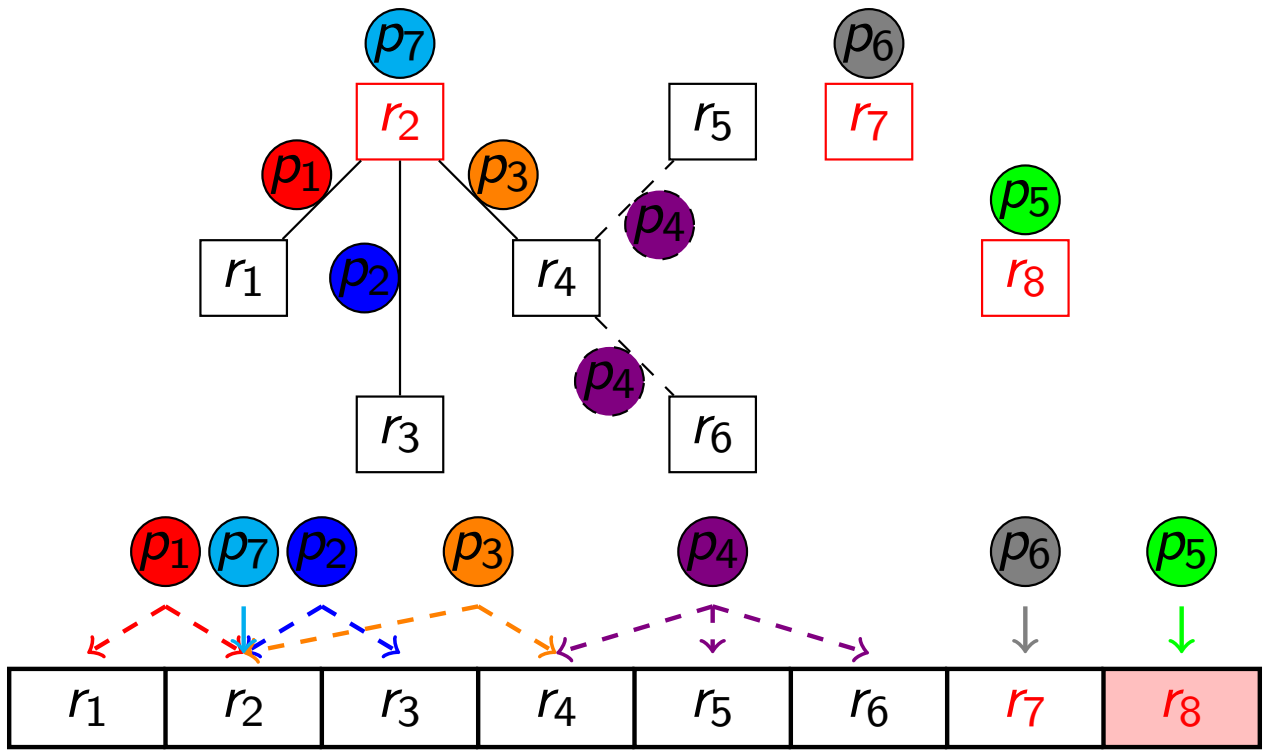
Induction Step (3^{rd} Phase): Increasing Confusion

1. Let p_7 cover some register in $\{r_1, r_2, r_3, r_4\}$.
2. Let p_5 run until it covers some register $\neq r_5$ for the last time:
   1. p_5 covers a register in $\{r_7, r_8\}$.



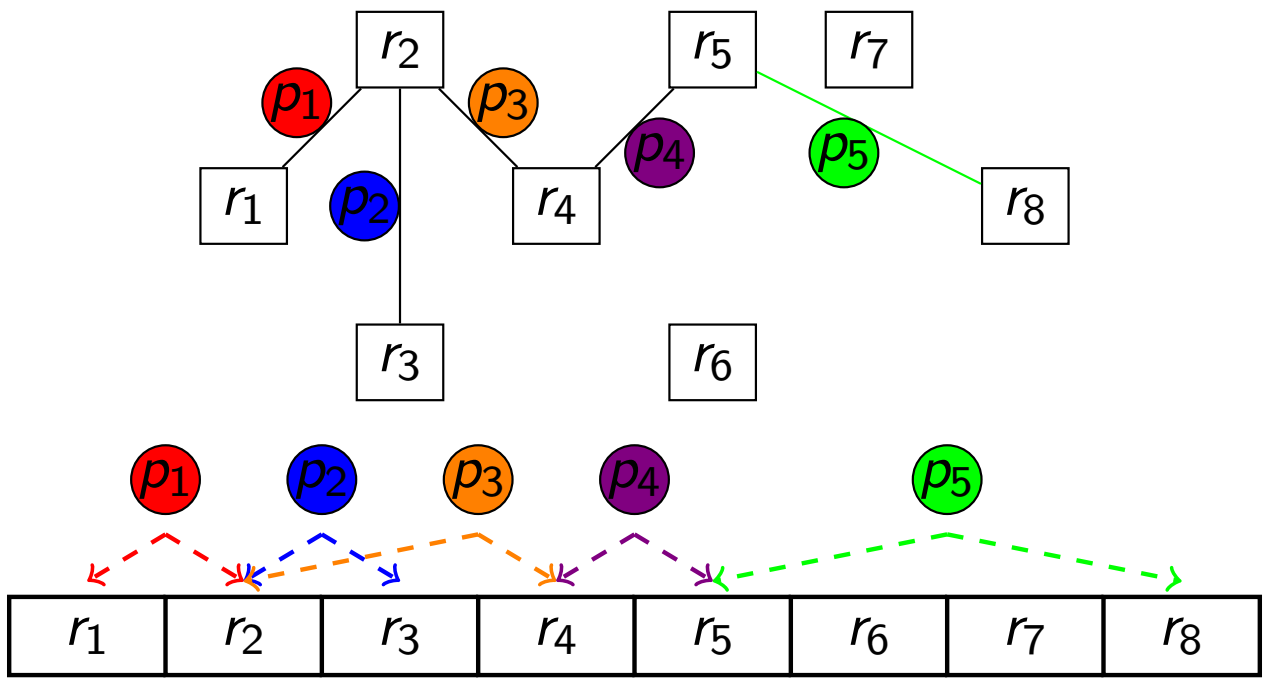
Induction Step (3^{rd} Phase): Increasing Confusion

1. Let p_7 cover some register in $\{r_1, r_2, r_3, r_4\}$.
2. Let p_5 run until it covers some register $\neq r_5$ for the last time:
   1. p_5 covers a register in $\{r_7, r_8\}$.



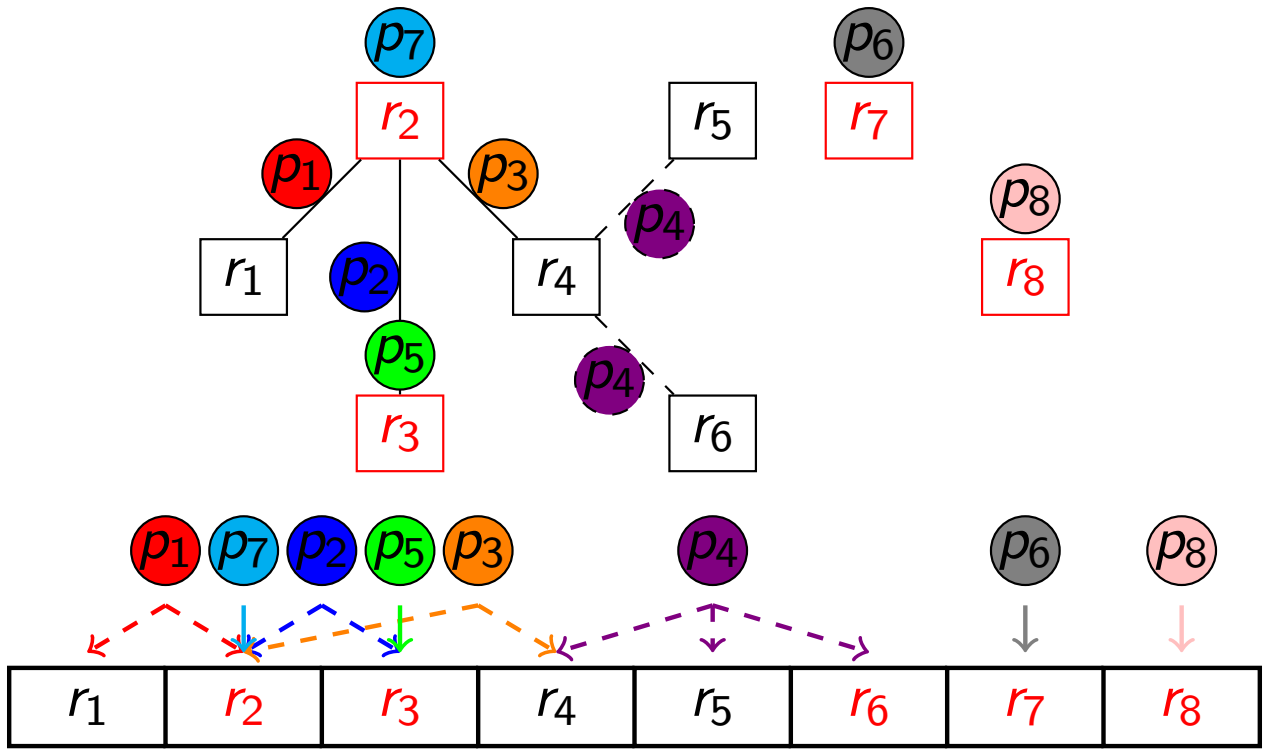
Induction Step (3^{rd} Phase): Increasing Confusion

1. Let p_7 cover some register in $\{r_1, r_2, r_3, r_4\}$.
2. Let p_5 run until it covers some register $\neq r_5$ for the last time:
   1. p_5 covers a register in $\{r_7, r_8\}$.



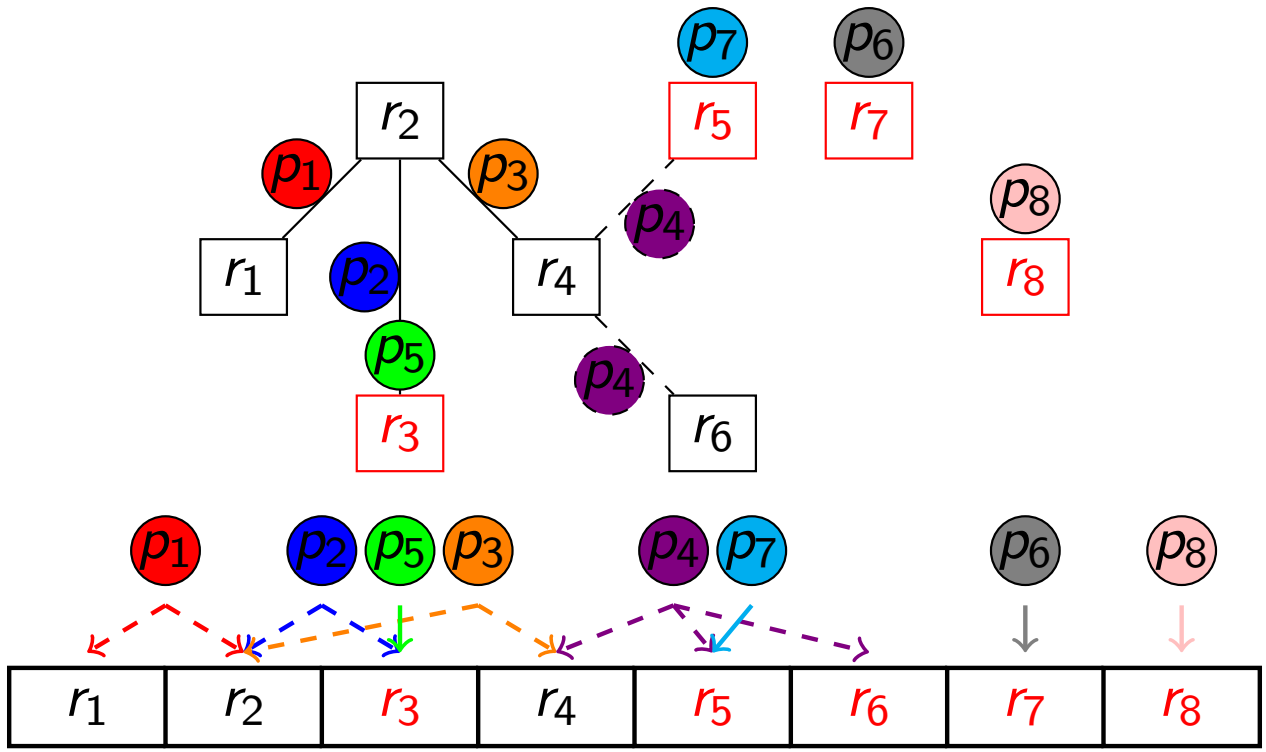
Induction Step (3^{rd} Phase): Increasing Confusion

1. Let p_7 cover some register in $\{r_1, r_2, r_3, r_4\}$.
2. Let p_5 run until it covers some register $\neq r_5$ for the last time:
   1. p_5 covers a register in $\{r_7, r_8\}$.
   2. After running p_7 so it covers r_5, p_5 is still going to write on r_5.



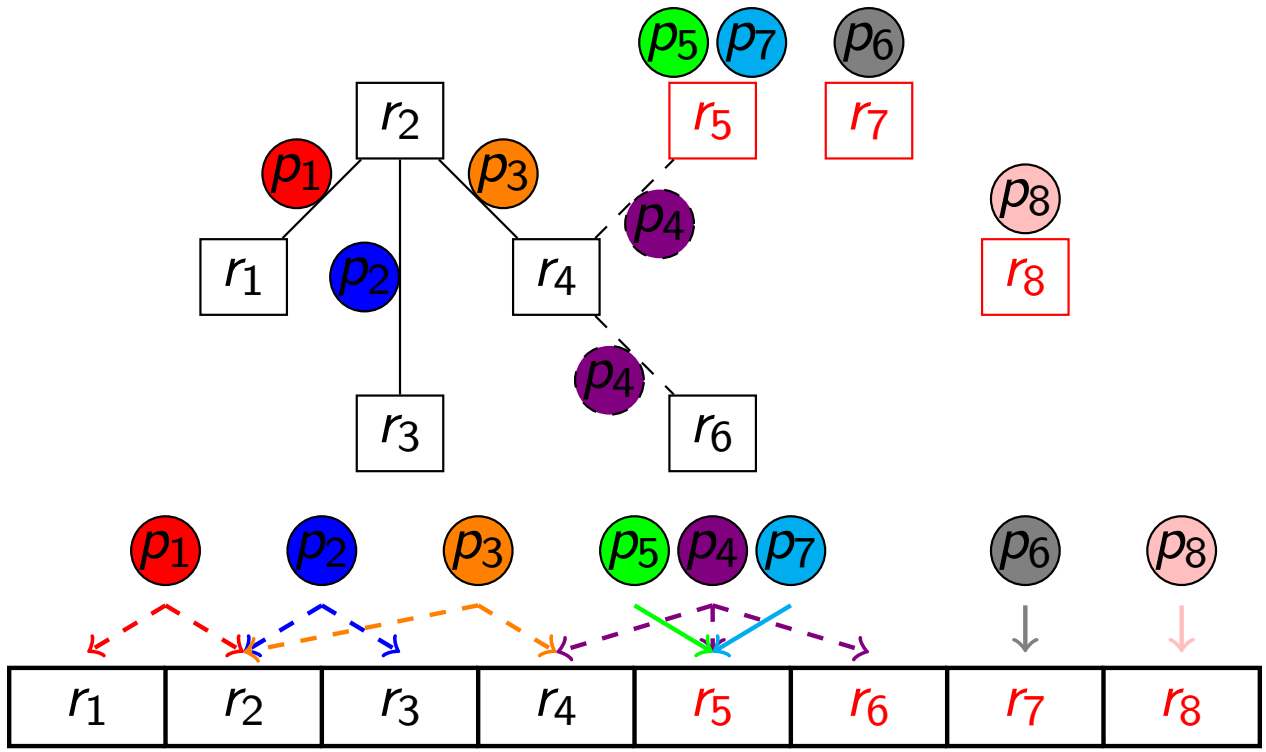
Induction Step (3^{rd} Phase): Increasing Confusion

1. Let p_7 cover some register in $\{r_1, r_2, r_3, r_4\}$.
2. Let p_5 run until it covers some register $\neq r_5$ for the last time:
   1. p_5 covers a register in $\{r_7, r_8\}$.
   2. After running p_7 so it covers r_5, p_5 is still going to write on r_5.



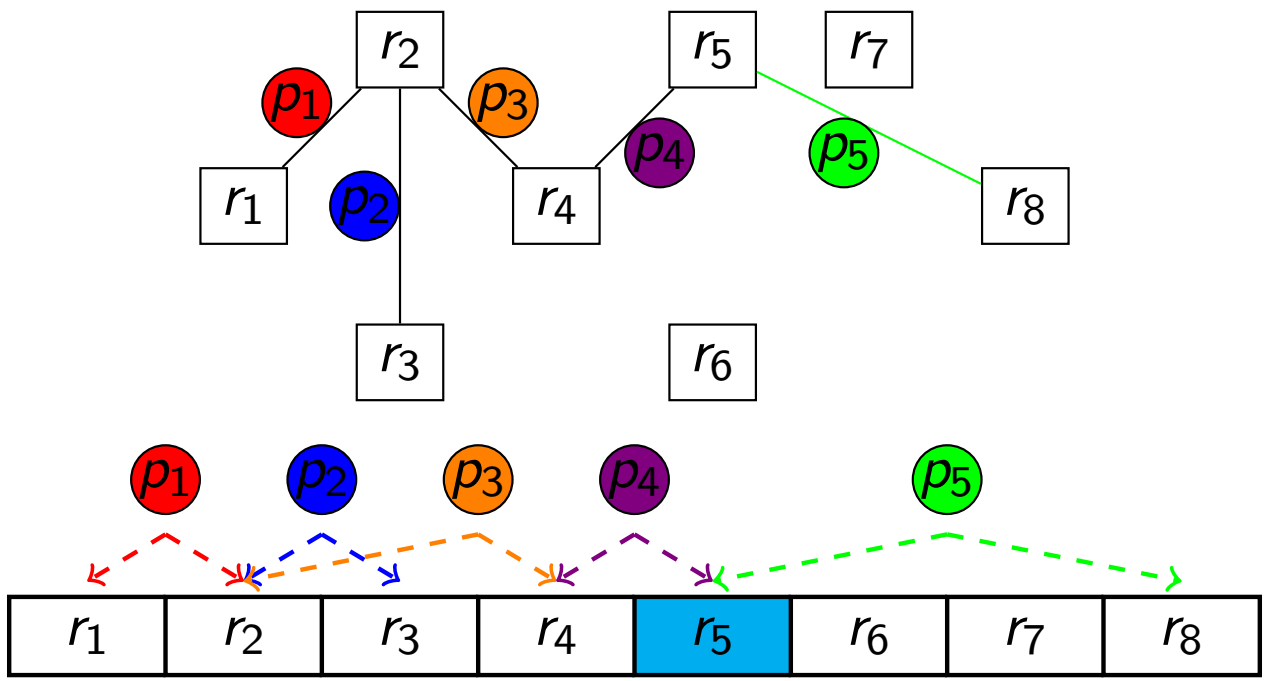
Induction Step (3^{rd} Phase): Increasing Confusion

1. Let p_7 cover some register in $\{r_1, r_2, r_3, r_4\}$.
2. Let p_5 run until it covers some register $\neq r_5$ for the last time:
   1. p_5 covers a register in $\{r_7, r_8\}$.
   2. After running p_7 so it covers r_5, p_5 is still going to write on r_5.



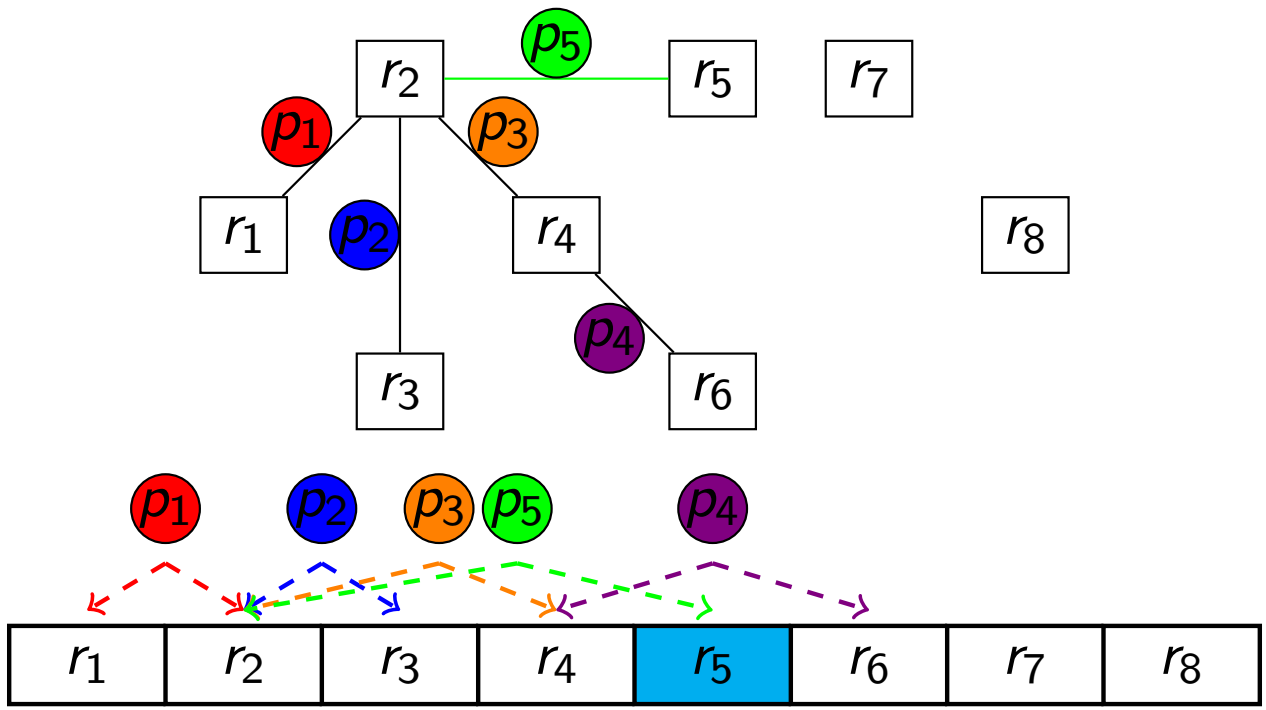
Induction Step (3^{rd} Phase): Increasing Confusion

1. Let p_7 cover some register in $\{r_1, r_2, r_3, r_4\}$.
2. Let p_5 run until it covers some register $\neq r_5$ for the last time:
   1. p_5 covers a register in $\{r_7, r_8\}$.
   2. After running p_7 so it covers r_5, p_5 is still going to write on r_5.



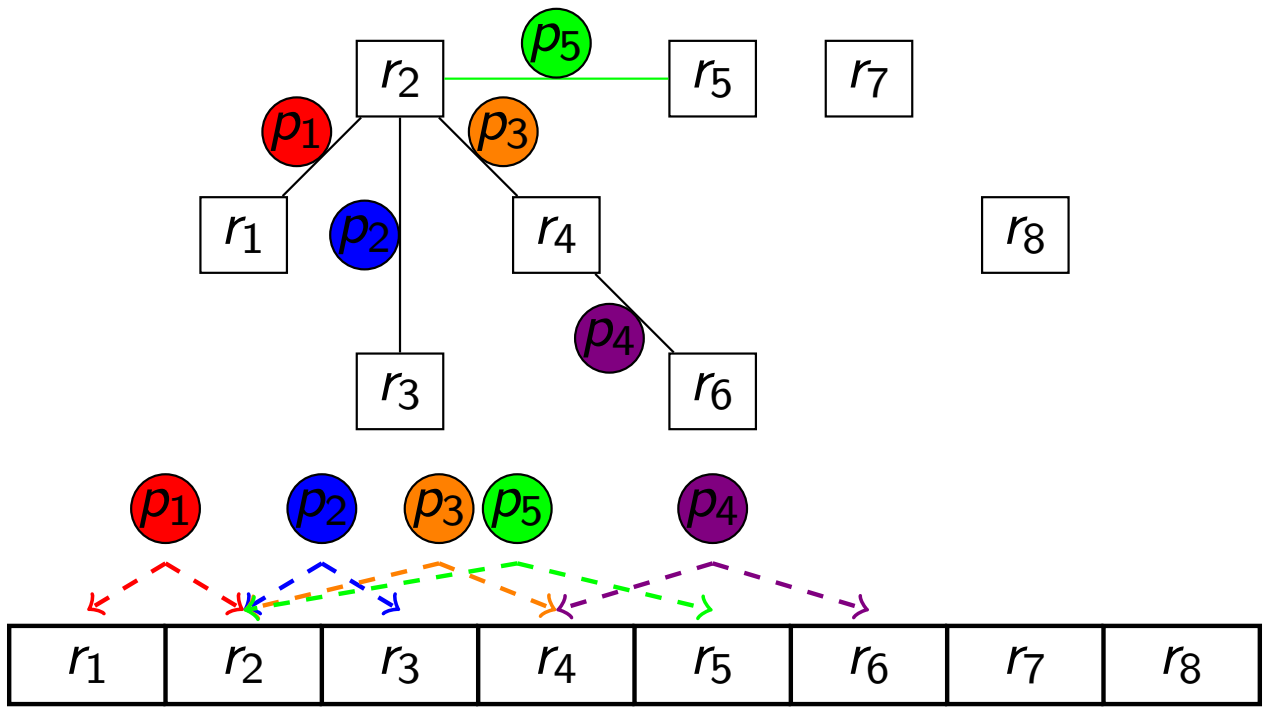
Induction Step (3^{rd} Phase): Increasing Confusion

1. Let p_7 cover some register in $\{r_1, r_2, r_3, r_4\}$.
2. Let p_5 run until it covers some register $\neq r_5$ for the last time:
   1. p_5 covers a register in $\{r_7, r_8\}$.
   2. After running p_7 so it covers r_5, p_5 is still going to write on r_5.



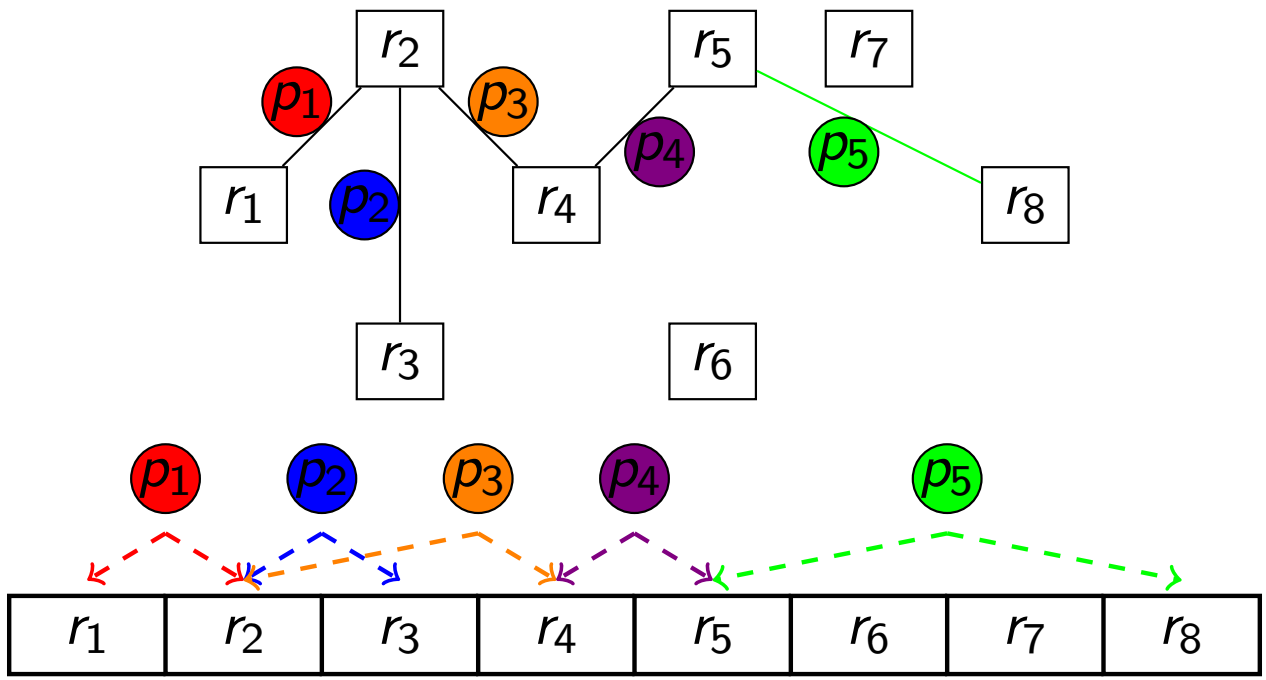
Induction Step (3^{rd} Phase): Increasing Confusion

1. Let p_7 cover some register in $\{r_1, r_2, r_3, r_4\}$.
2. Let p_5 run until it covers some register $\neq r_5$ for the last time:
   1. p_5 covers a register in $\{r_7, r_8\}$.
   2. After running p_7 so it covers r_5, p_5 is still going to write on r_5.
   3. There is a critical step in p_7 execution (a write) after which p_5 no longer writes to r_5: $I(\{e_7 s_c w_5 rds_5, e_7 w_5 rds_5 s_c\}, \Pi \setminus \{p_5\})$.



Induction Step (3^{rd} Phase): Increasing Confusion

1. Let p_7 cover some register in $\{r_1, r_2, r_3, r_4\}$.
2. Let p_5 run until it covers some register $\neq r_5$ for the last time:
   1. p_5 covers a register in $\{r_7, r_8\}$.
   2. After running p_7 so it covers r_5, p_5 is still going to write on r_5.
   3. There is a critical step in p_7 execution (a write) after which p_5 no longer writes to r_5: $I(\{e_7 s_c w_5 rds_5, e_7 w_5 rds_5 s_c\}, \Pi \setminus \{p_5\})$.



Conclusion and Perspectives

Interesting results:

- Differentiating lock-freedom and wait-freedom feasability.
- Impossibility lower bound with more registers than processes.
- New covering-indistinguishability proof technique.

Potential extensions:

- Prove conjecture of algorithm space optimality for any k.
- Generalize and improve notion of *confusion*.
- Anonymous and/or adaptive generalizations.

Questions?

Thank You!