Fast Parallel PDR Algorithms for Planning

Marshall Clifton
PhD Candidate

Overview

- Boolean SAT
- Classical Planning
- Encoding Classical Planning as SAT
- Traditional Ramp-Up
- Property Directed Reachability
- Obligation Rescheduling
- Parallel PDR
- Parallel Decompositional PDR
- Results

Boolean SAT - Representation

- Find a model for a Boolean formula
  - Eg: $(a \wedge b) \vee (\neg b \wedge a)$
- Common Representation CNF:
  - $literal$: proposition or its negation
  - $clause$: disjunction of literals
  - CNF: Conjunction of clauses
  - Eg: $(a \vee b) \wedge (b \vee \neg c) \wedge (a \vee \neg b \vee \neg c)$
  - Each clause is an additional constraint

Boolean SAT - Solving

- Stochastic Local Search (SLS)
  - Treated as an optimization problem
  - Over space of assignments: minimize number of UNSAT clauses - stop if 0
  - Sound but incomplete - cannot prove UNSAT
- Conflict Driven Clause Learning (CDCL)
  - Decision Procedure
  - Sound and complete

Classical Planning

- Safety checking in bounded model checking
- Can a finite transition system reach a state

STRIPS - Common Representation

- Fluent set F
  - Example: $\{Light\_On, Window\_Open, \ldots\}$
- State: Boolean valuation over F
- Example of a state:
  - Lights_On $= \top$
  - Window_Open$= \bot$
  - ...
- Action: Transition between states
- Can I act from I to G

STRIPS - Common Representation

- Initial State I
- Goal Condition G - i.e. partial valuation over F
- Action set A where each $a \in A$ given by :
  - $f \in \mathrm{Pre}(a)$ satisfied in predecessor
  - $f \in \mathrm{Add}(a)$ satisfied in successor
  - $f \in \mathrm{Del}(a)$ not satisfied in successor

Planning via SAT

- Step: state, set of actions executing on that state.
- Action set must be non-conflicting - can be executed in any order

Planning via SAT

- Bound the number of steps with horizon h
- SAT encoding of problem at horizon h - in CNF
- Satisfiable iff plan of length h or less exists

Planning via SAT

- Direct Encoding - plan can be efficiently extracted
- SAT proposition for fluent/action for most/all steps
  - $f@t$: fluent f True at step t.
    For every step
  - $a@t$: action a executed at step t.
    For all but the last step
- Clauses enforce problem constraints

Clauses

- Initial State: $\bigwedge_{i \in I} i@0$
- Goal Condition: $\bigwedge_{g \in G} g@h$
- Action Set: $a \in A, t \in \{0, \ldots, h-1\}$:

$$\bigwedge_{f \in Pre(a)} a@t \quad \rightarrow \quad f@t \quad \wedge$$

$$\bigwedge_{f \in Add(a)} a@t \quad \rightarrow \quad f@(t+1) \quad \wedge$$

$$\bigwedge_{f \in Del(a)} a@t \quad \rightarrow \quad \neg f@(t+1)$$

More Clauses

- Explanatory Frame Axiom:
  $f \in F, t \in \{0, \ldots, h-1\}$:

  $$(\neg f@t \wedge f@(t+1)) \rightarrow \bigvee_{a \in A \text{ s.t. } f \in Add(a)} a@t$$

  $$(f@t \wedge \neg f@(t+1)) \rightarrow \bigvee_{a \in A \text{ s.t. } f \in Del(a)} a@t$$

- Interference Mutex Axiom:
  $a, a' \in A, t \in \{0, \ldots, h-1\}$:
  $\exists f \text{ s.t. } f \in Del(a) \text{ and } f \in Pre(a')$
  $\rightarrow \neg(a@t \wedge a'@t)$

Invariants

- Usually binary mutex clauses
- Eg. $\neg ON(A, B) \vee \neg ON(B, A)$
- Usually provided by preprocessing

Rintanen, J. 2012. Planning as Satisfiability: Heuristics.

Ramp-Up

- Usually incomplete
- Series of bounded instances in a query strategy
  - Geometric sequence (eg $[1, 2, 4, 8, 16, \ldots]$)
  - Multiple horizons in parallel
  - Use result from one to help another
    - i.e. incremental SAT

Completeness Thresholds - Upper Bounds

- Makes SAT-based Planning **Complete**
- Completeness Threshold: plan exist at this horizon iff one exist at all

Completeness Thresholds - Upper Bounds

- Makes SAT-based Planning **Complete**
- Completeness Threshold: plan exist at this horizon iff one exist at all
- Trivial bound: $2^{|F|} - 1$
- Methods exist for finding smaller thresholds

Property Directed Reachability

- Sound and **complete** SAT planning procedure
- Maintains reachability information compactly
- Iteratively tightens that by restricting via nogoods
- Calls a SAT solver **a lot** on small subproblems

Bradley, A. R. 2011. SAT-Based Model Checking Without Unrolling.
Suda, M. 2014. Property Directed Reachability for Automated Planning.

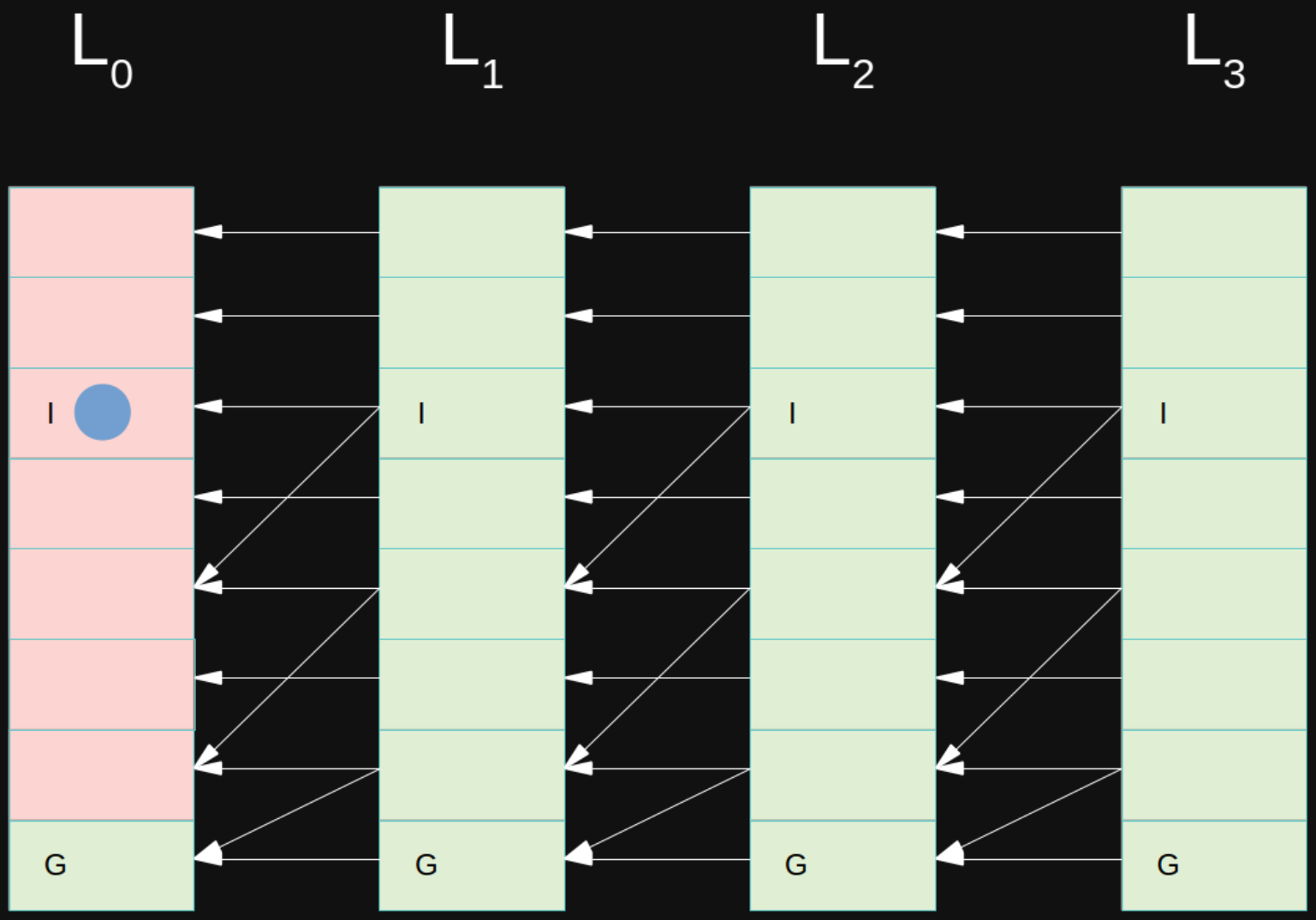
Property Directed Reachability

- For each time step N from the goal maintain:
  - Layer information: - overapproximation of what can reach the goal
  - States which may be able to reach the goal
- Process states until:
  - Reach a goal state
  - Show goal is unreachable

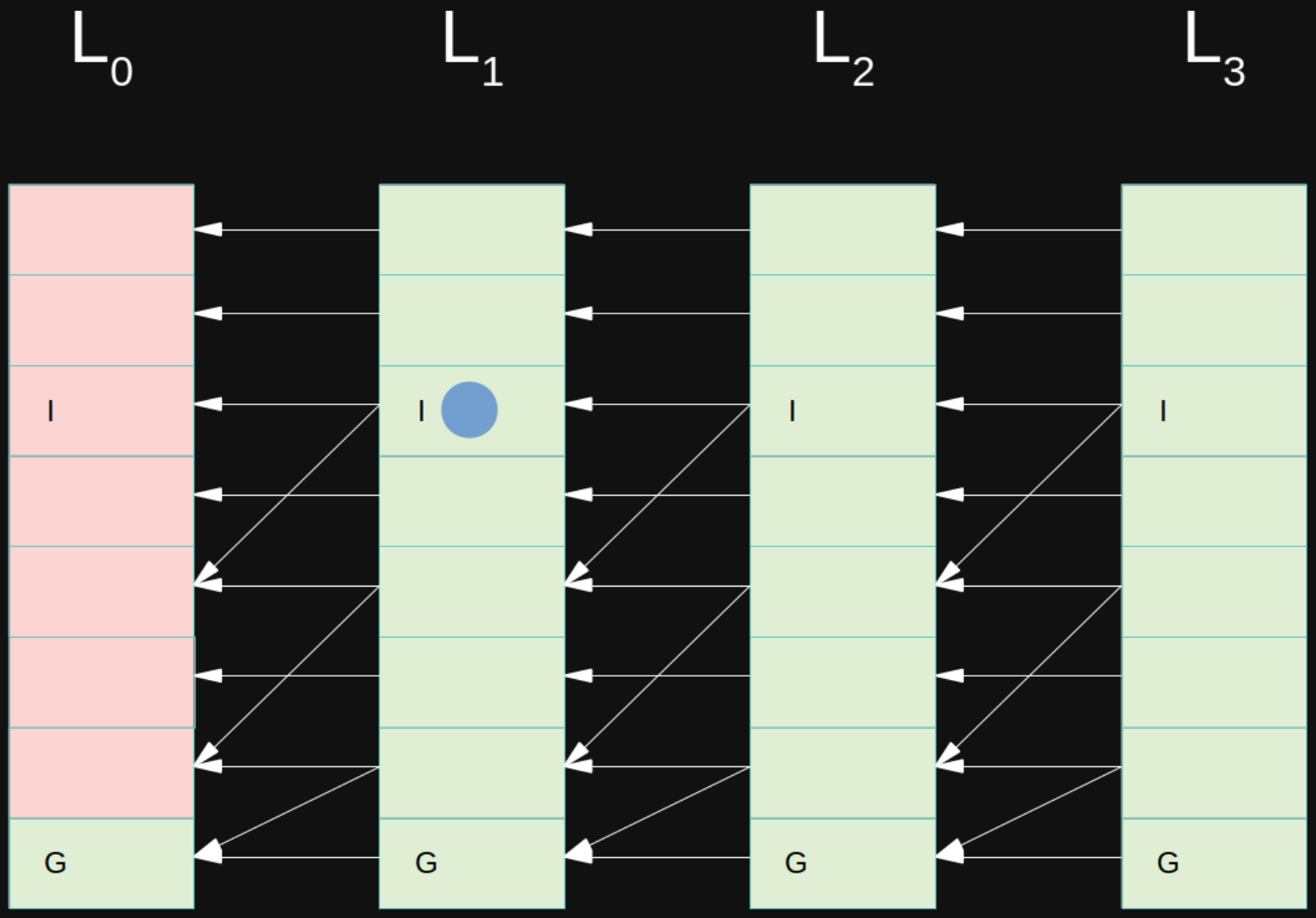
PDR - Reasoning

- Robot trying to move to a room
- State:
  - ¬Window_Open
  - ¬Door_Open
  - ¬Have_Key
  - Light_On
- Reason:
  - ¬Door_Open
  - ¬Have_Key

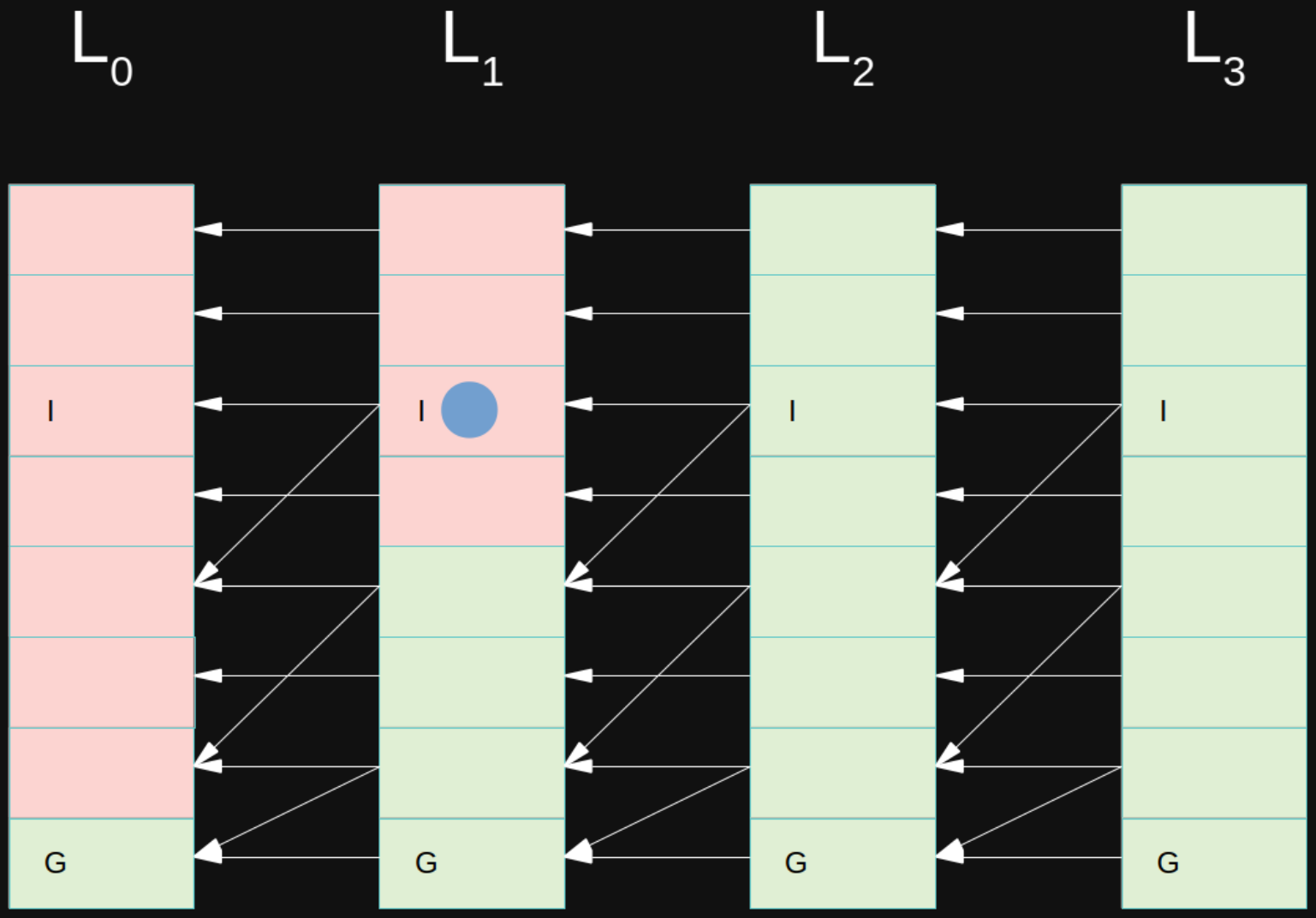
PDR - Example



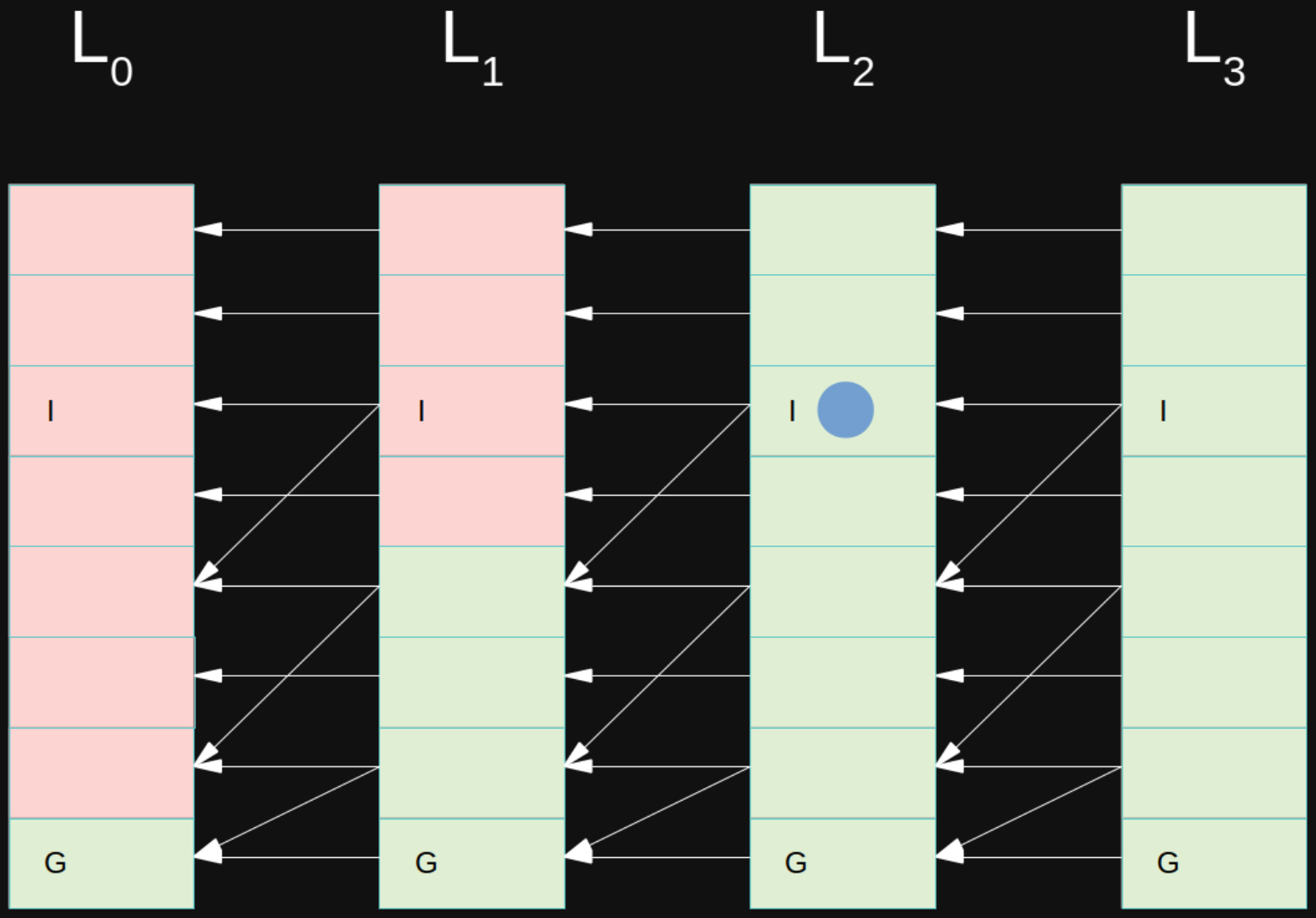
PDR - Example



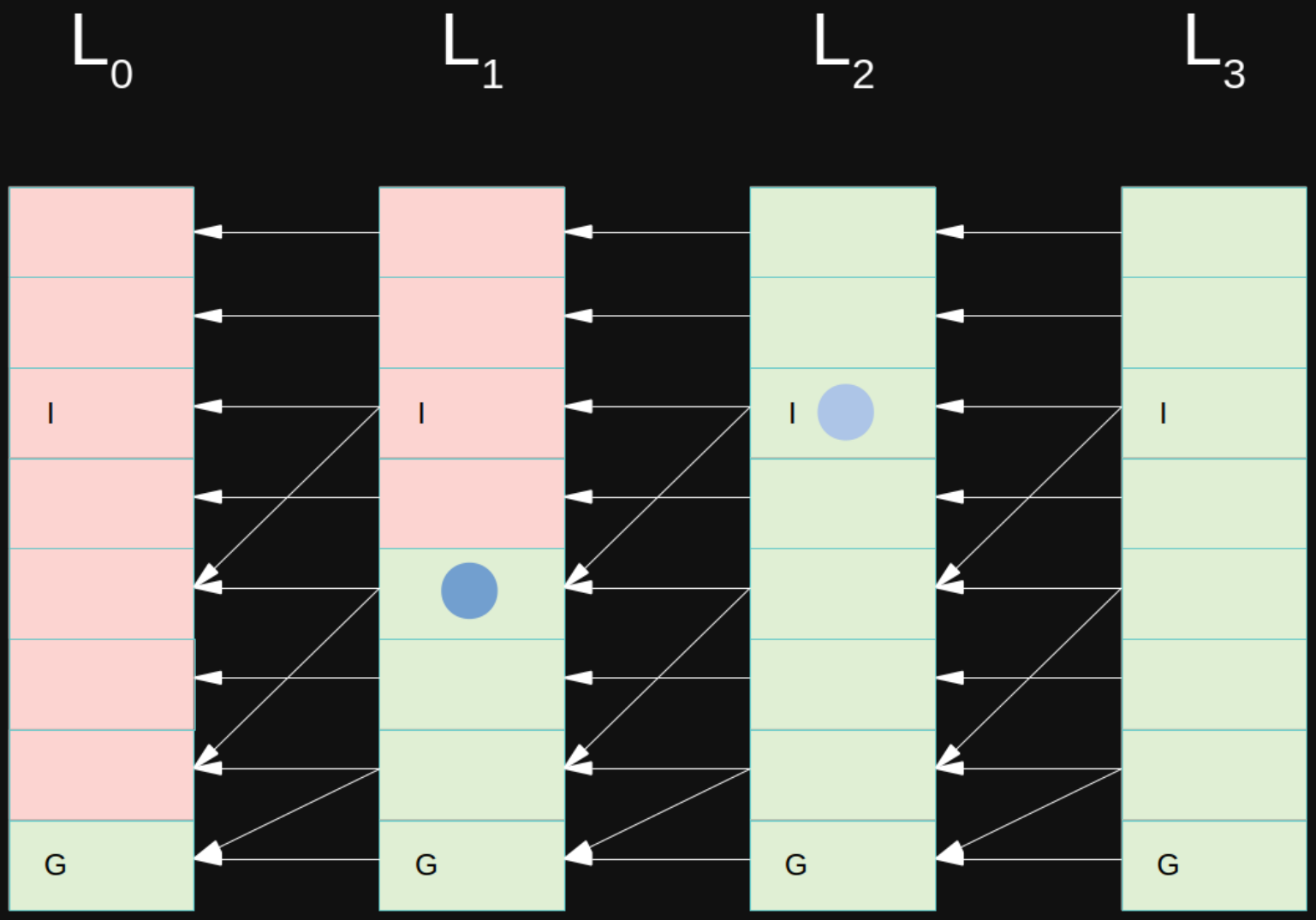
PDR - Example



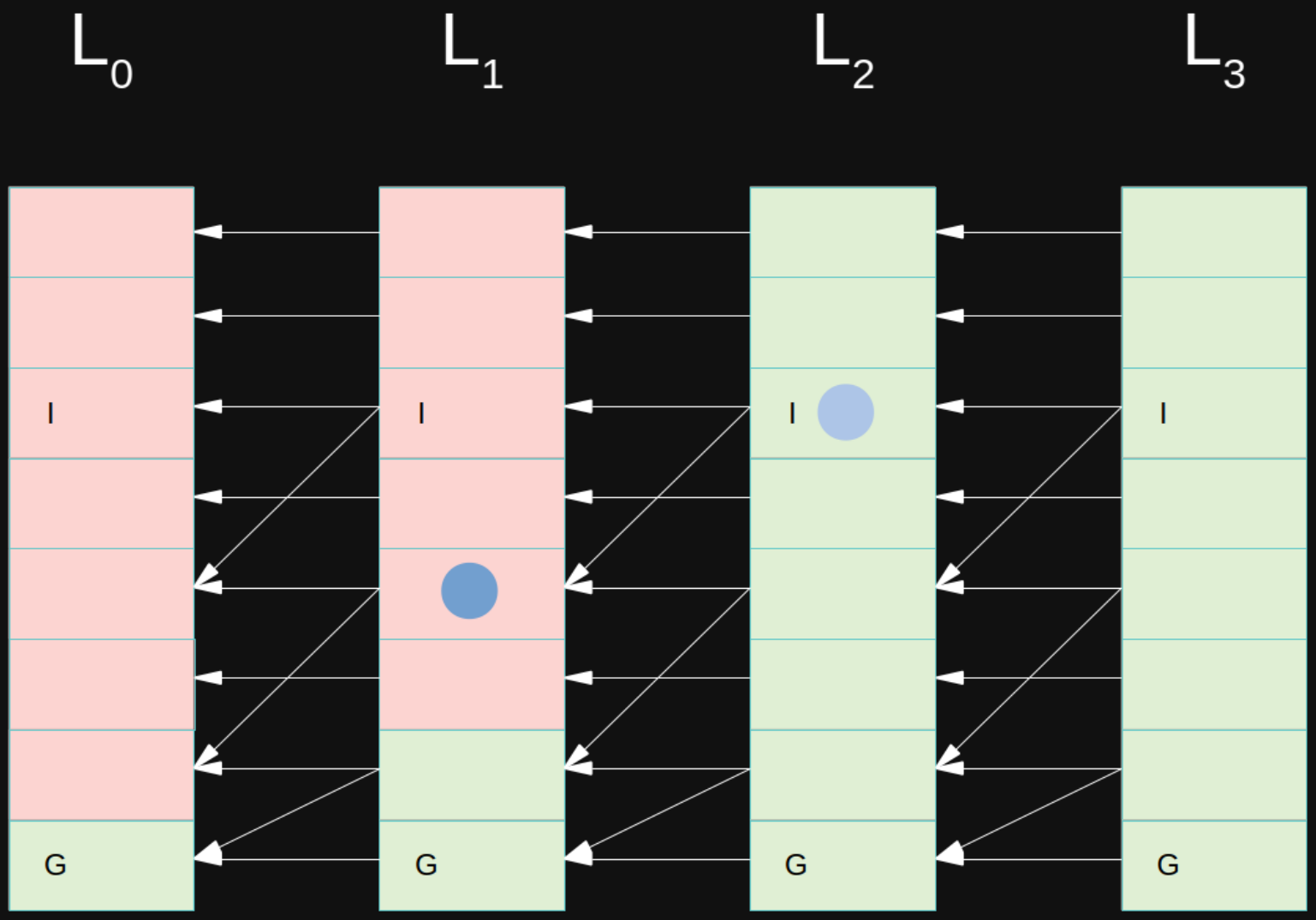
PDR - Example



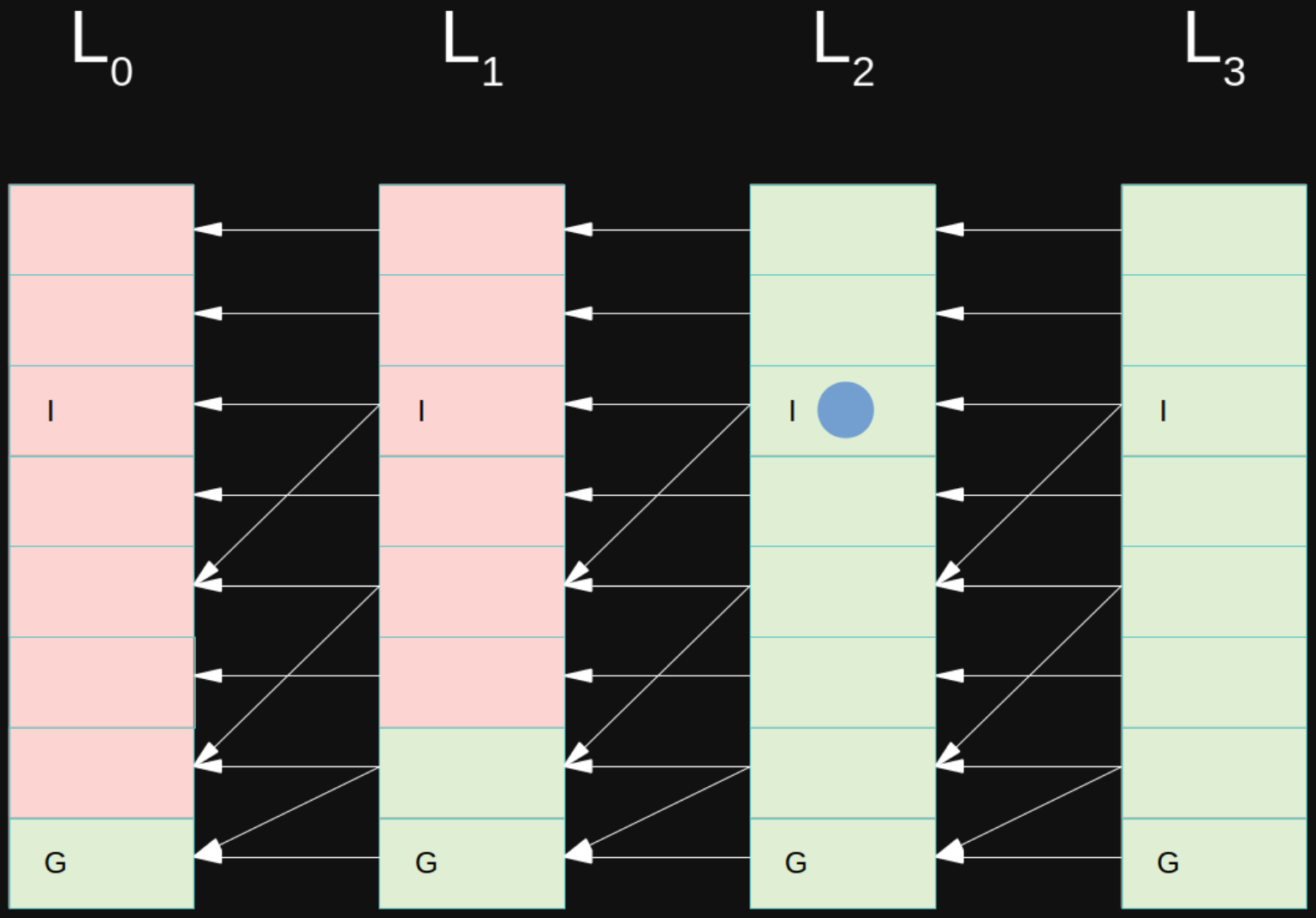
PDR - Example



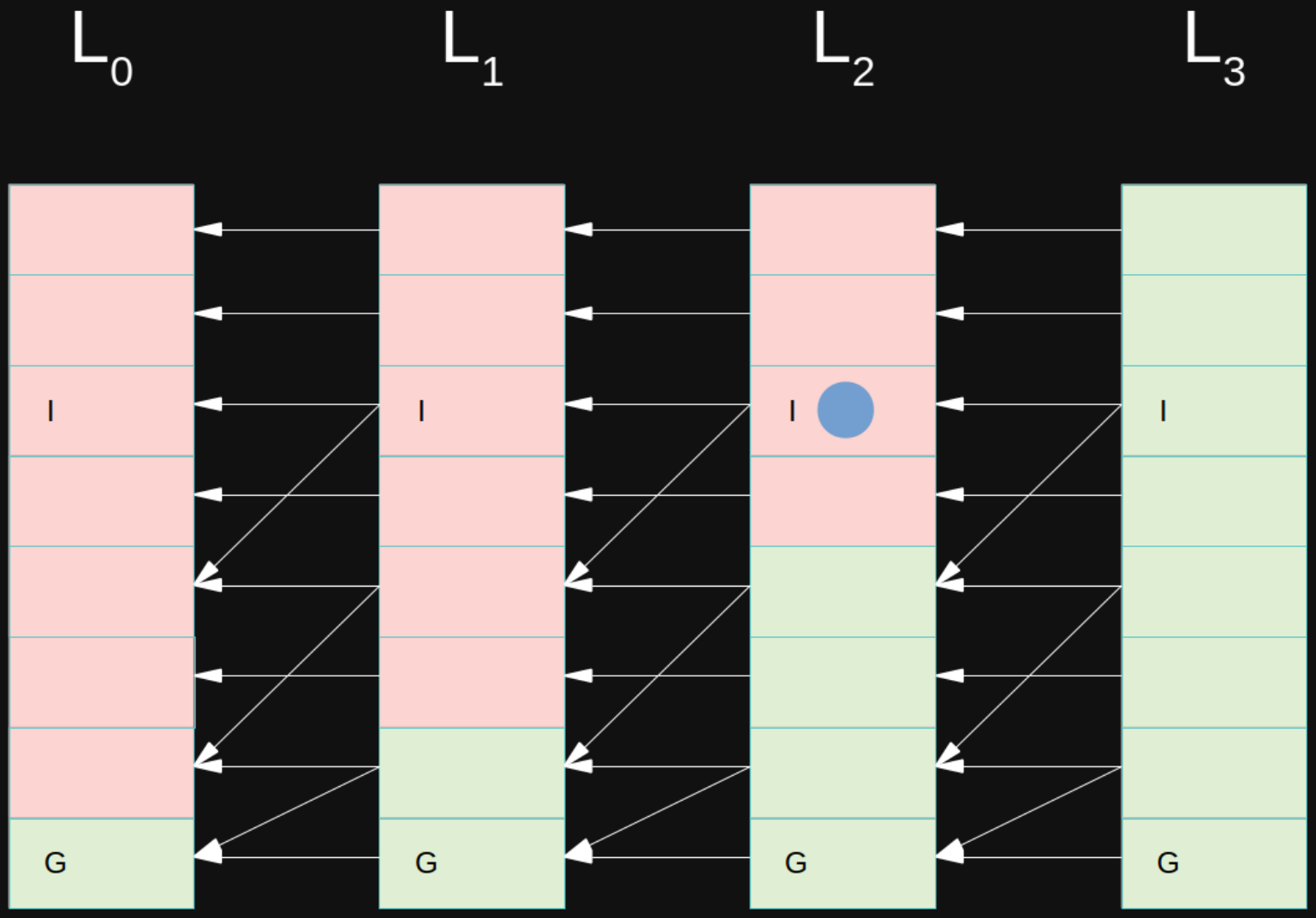
PDR - Example



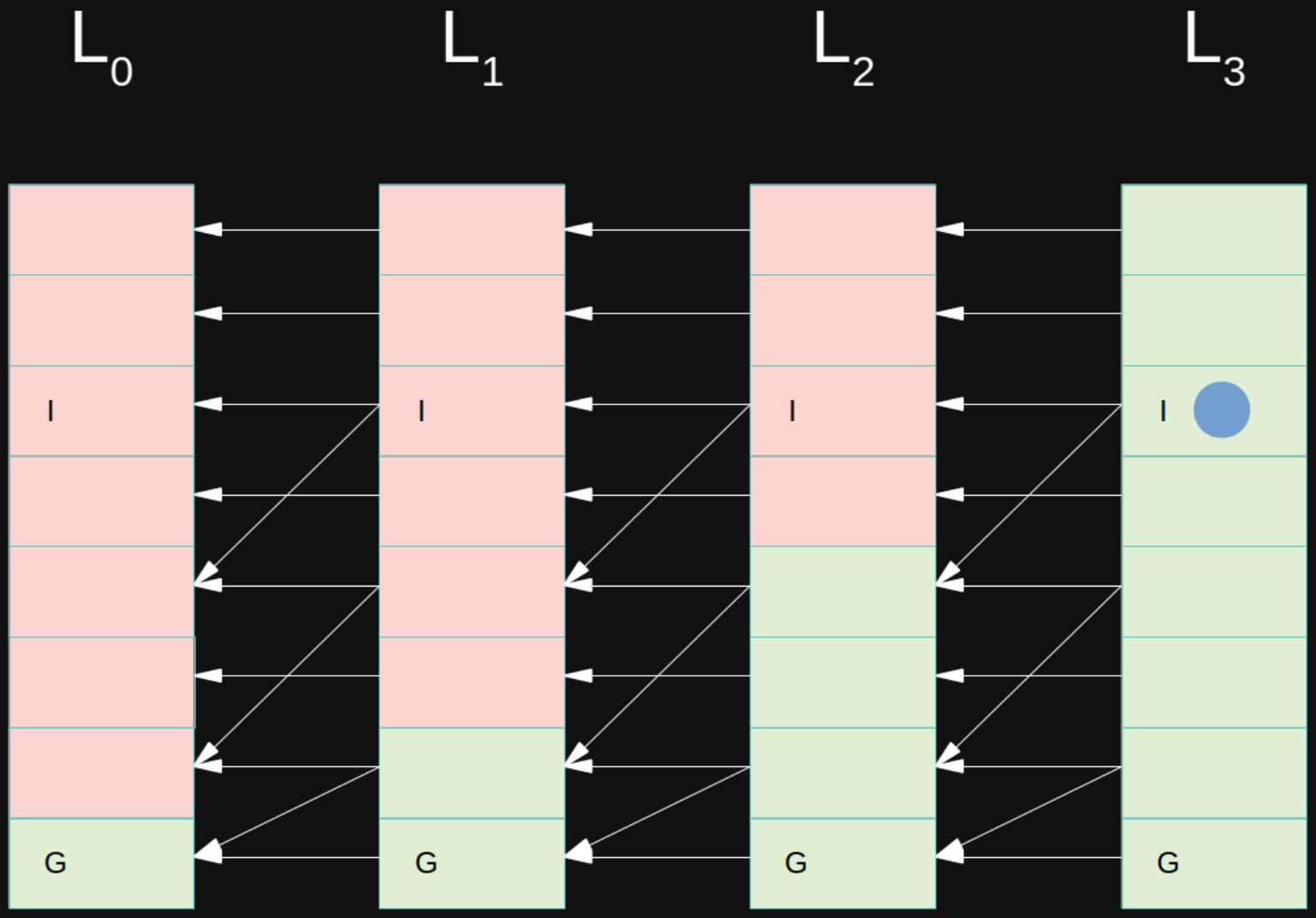
PDR - Example



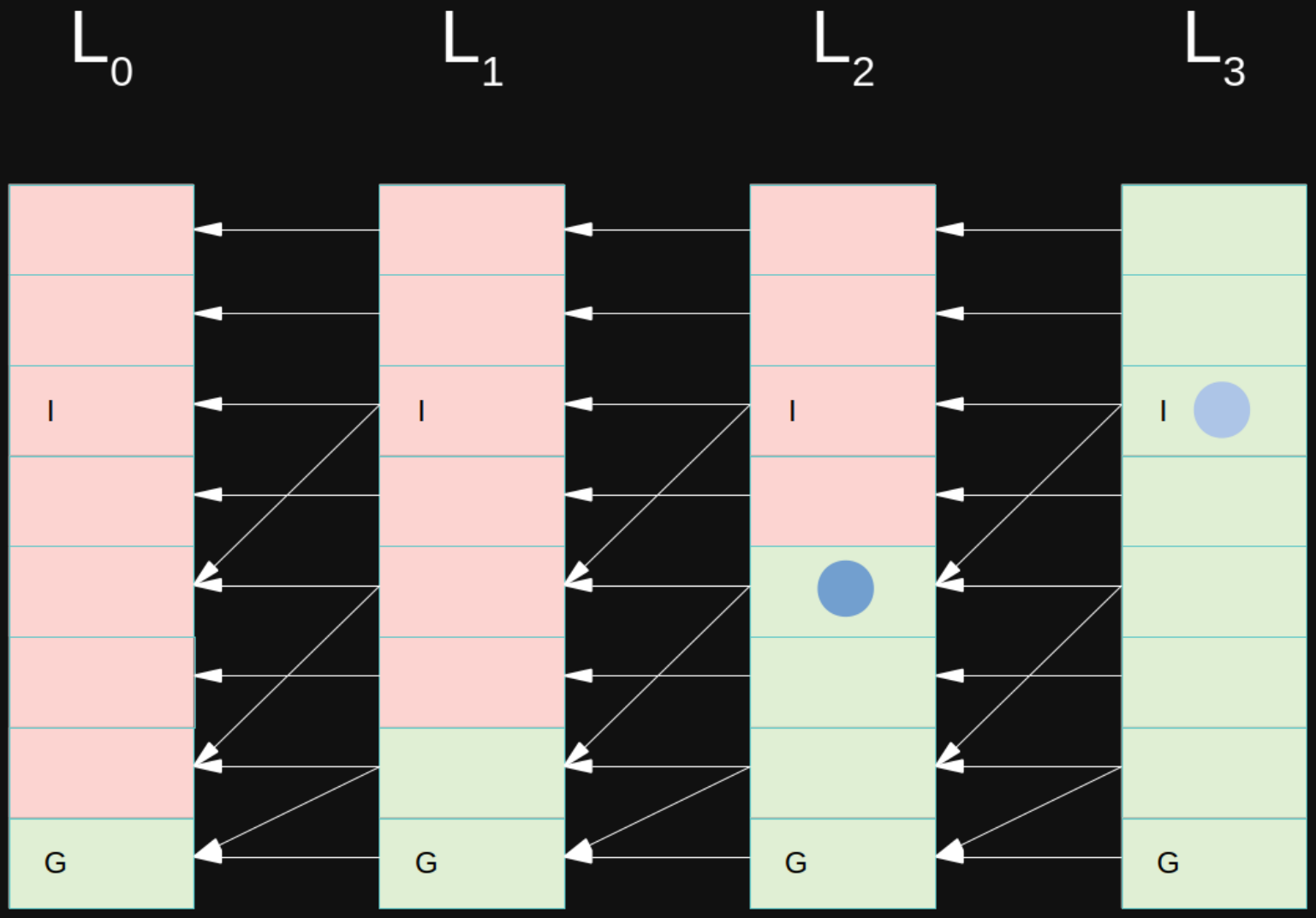
PDR - Example



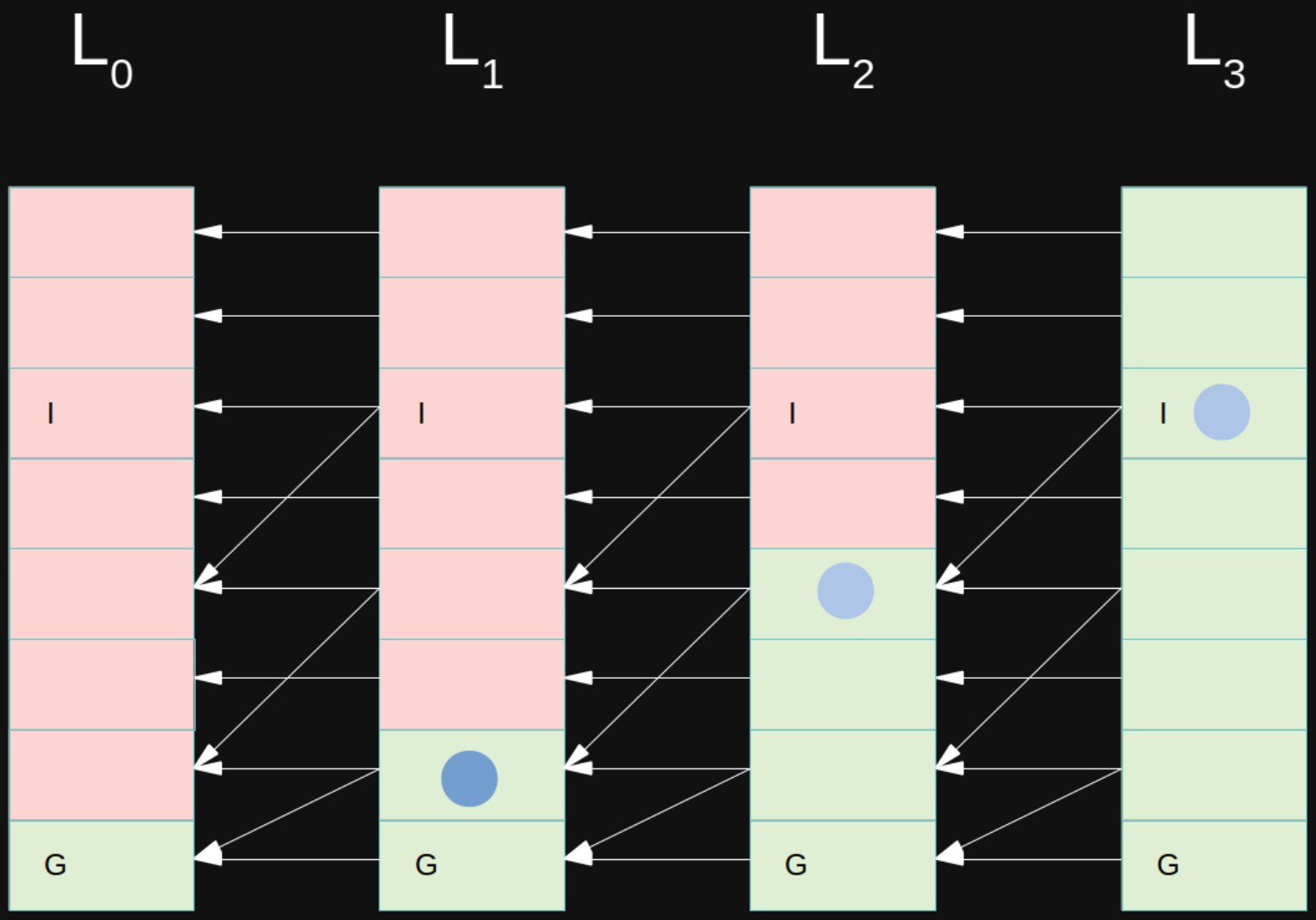
PDR - Example



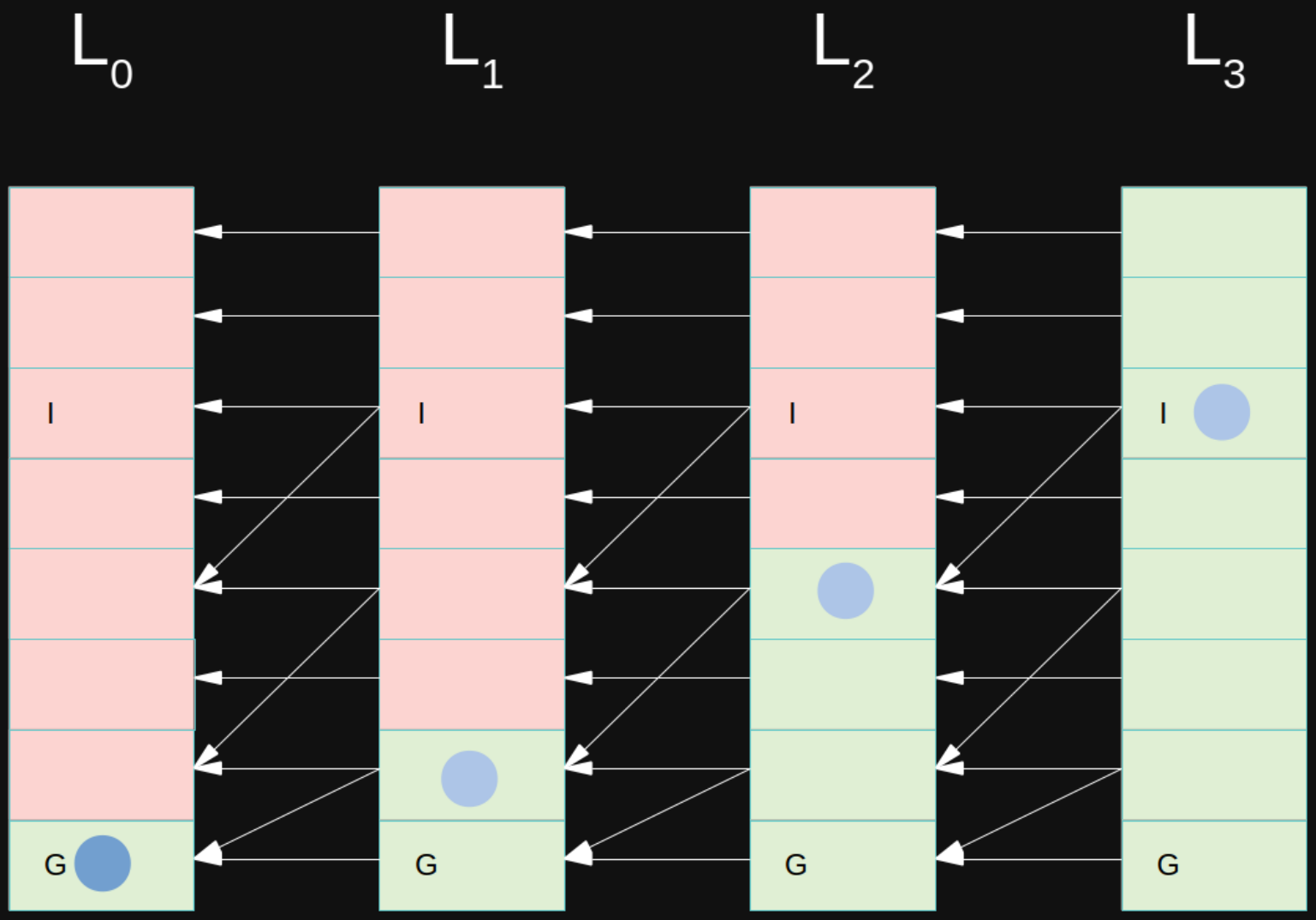
PDR - Example



PDR - Example



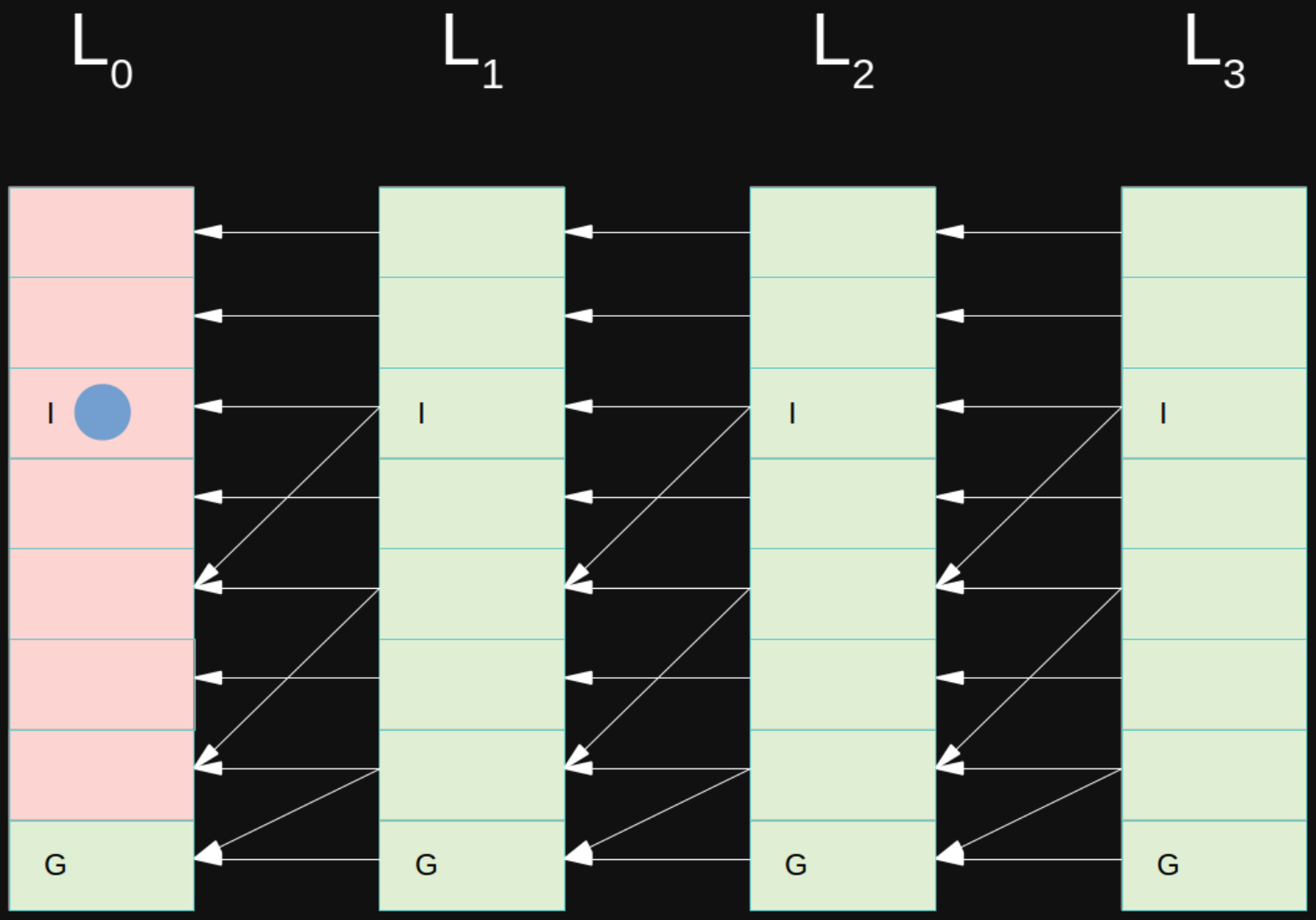
PDR - Example



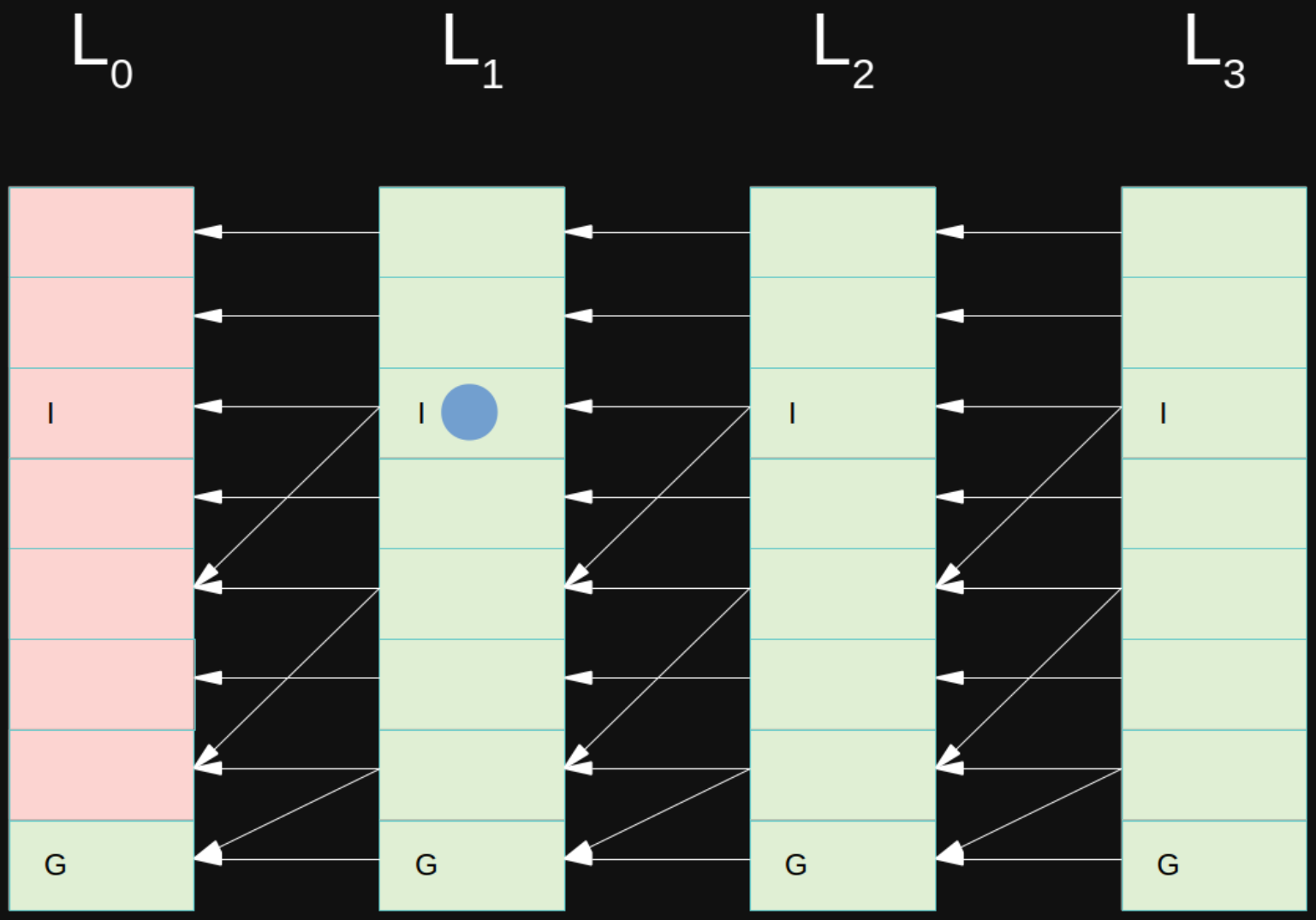
Obligation Rescheduling

- Reprocess failed obligations at the next layer
- Not optimal length plans
- Generally faster in practice

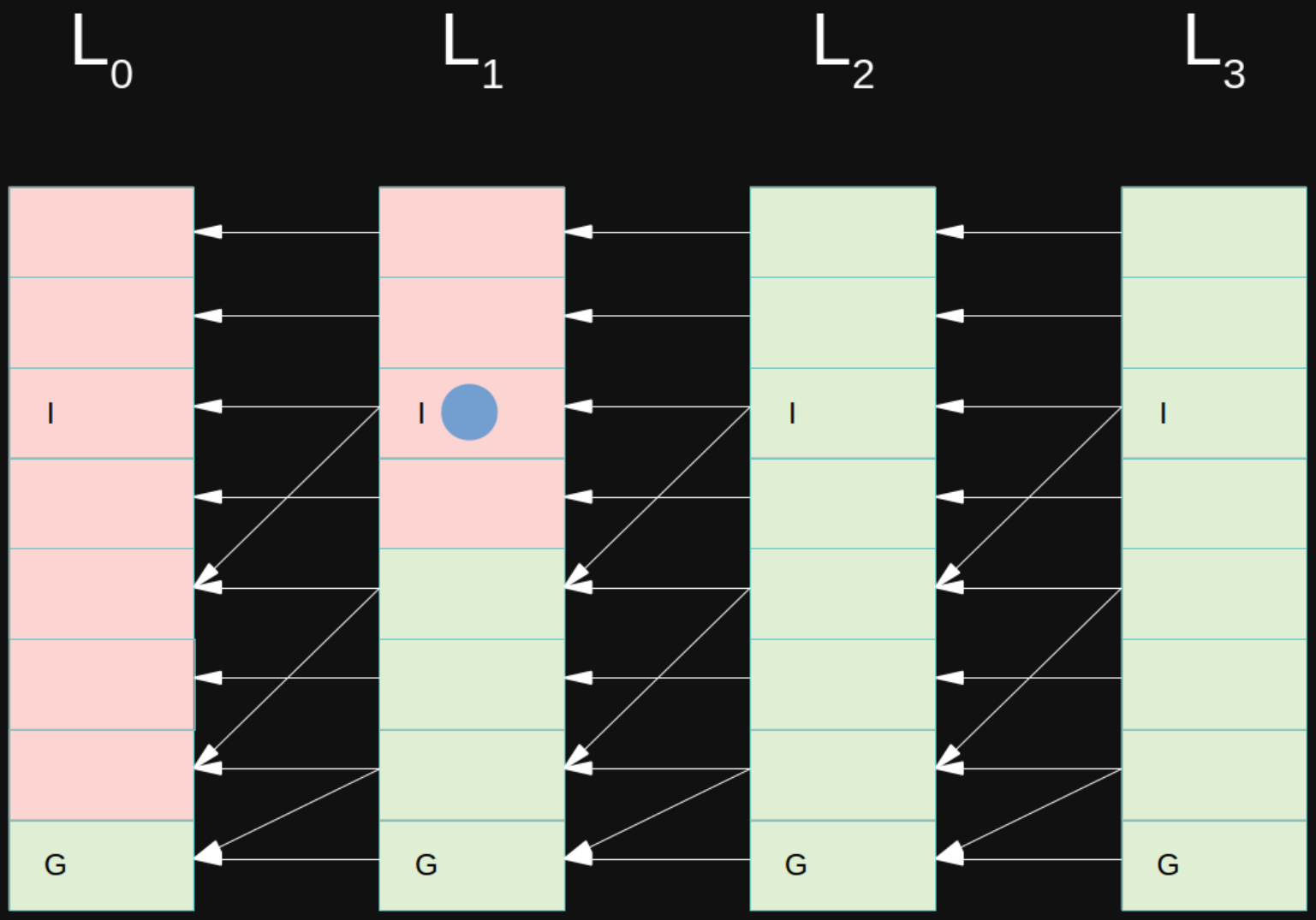
OR - Example



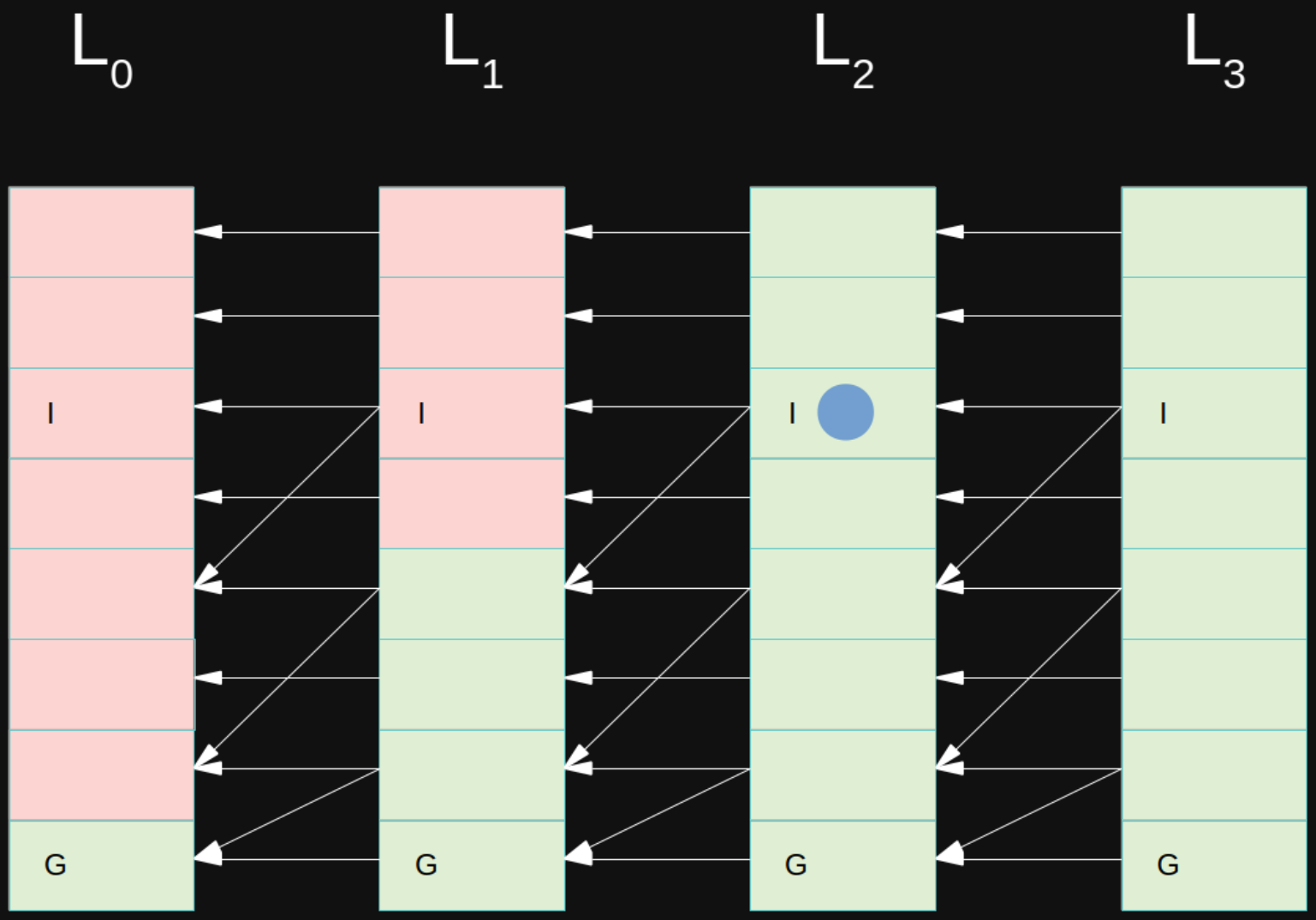
OR - Example



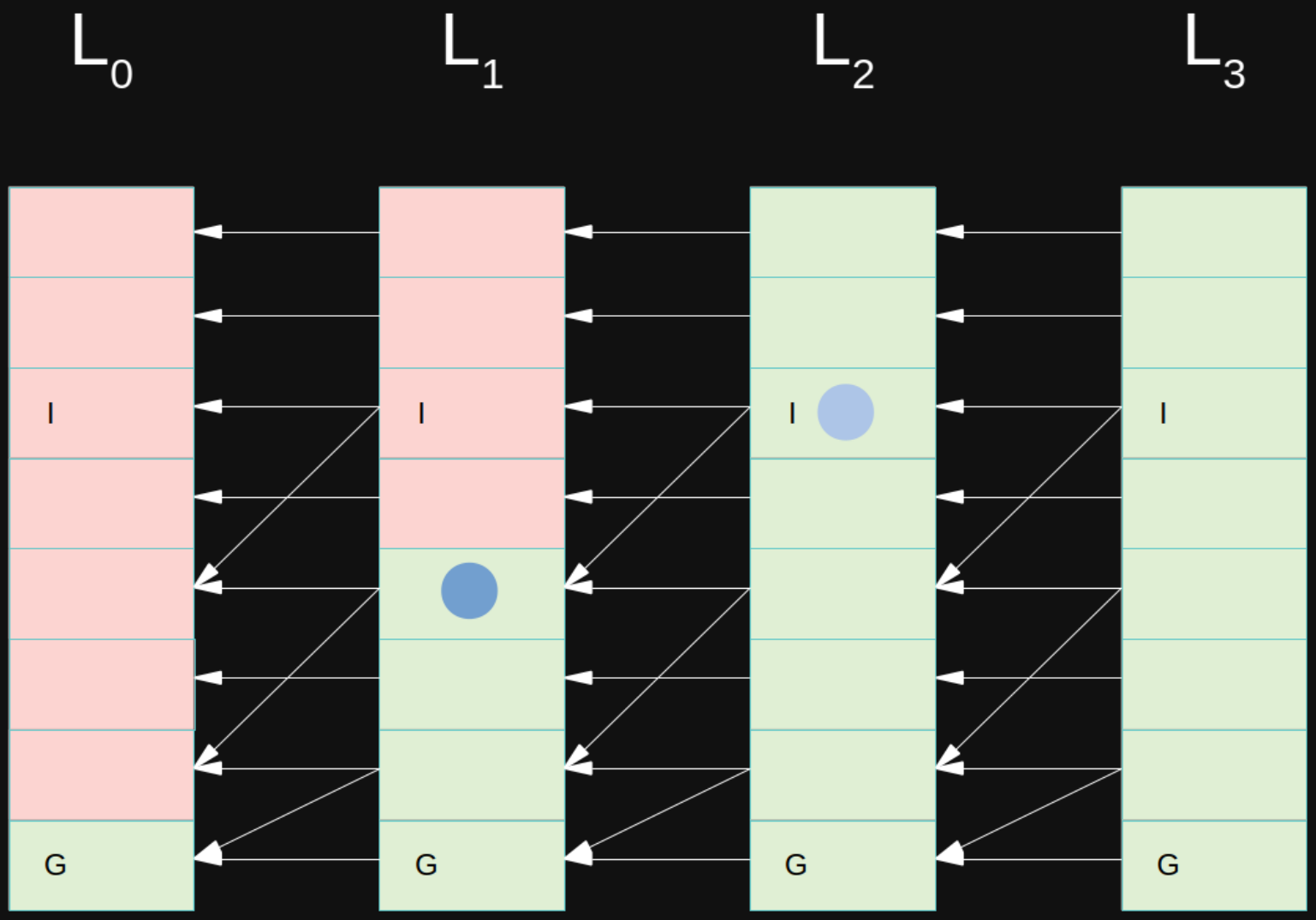
OR - Example



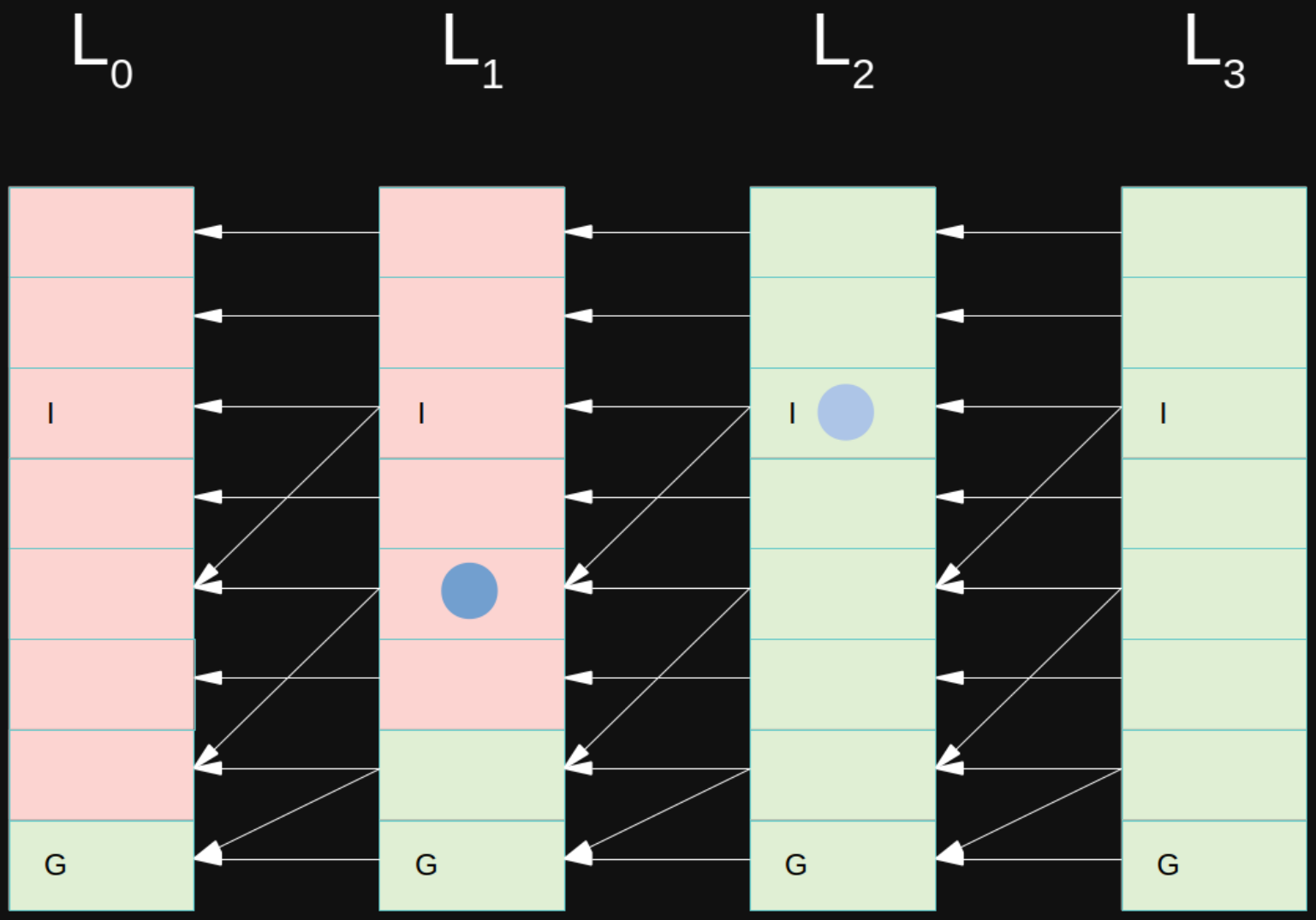
OR - Example



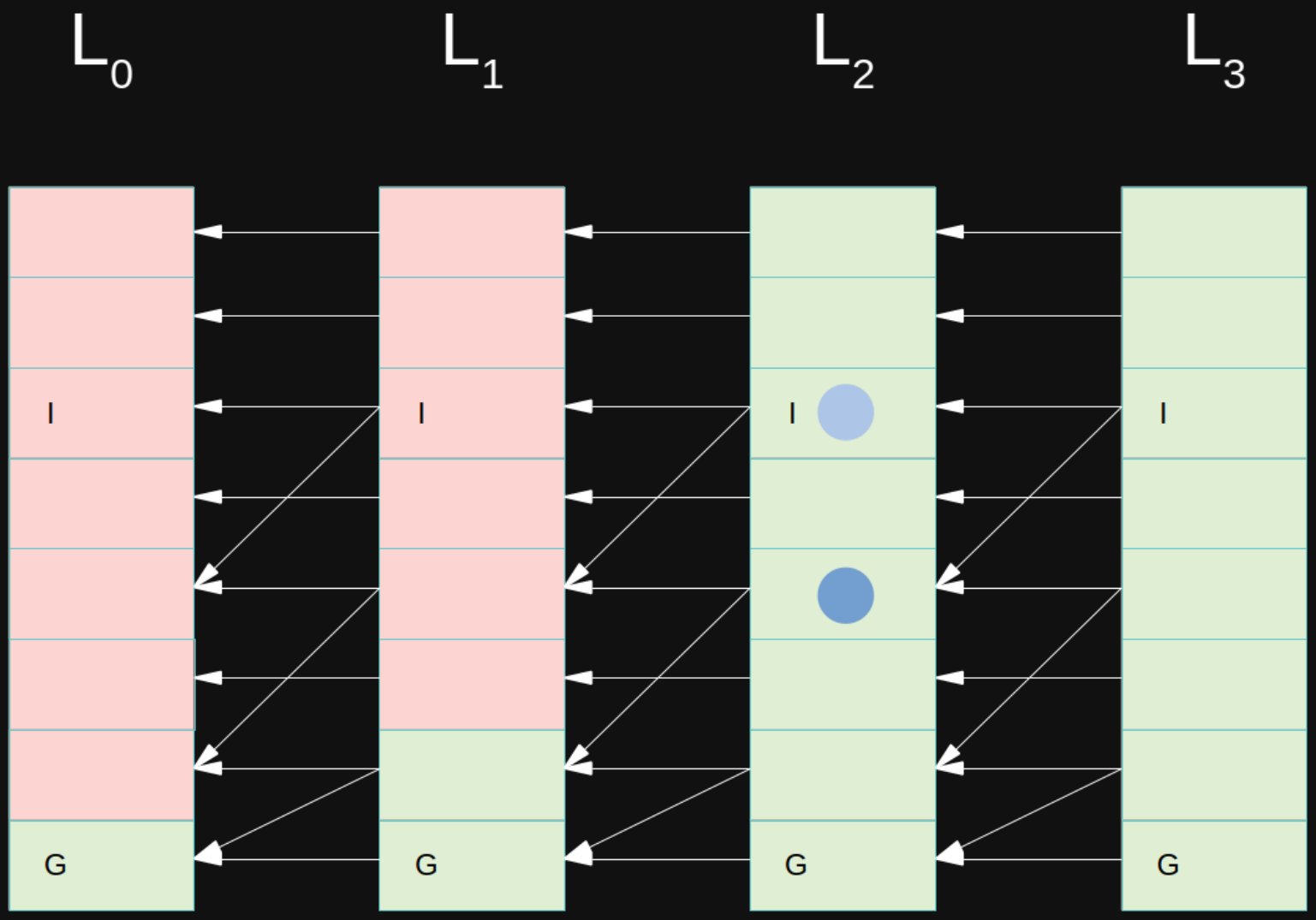
OR - Example



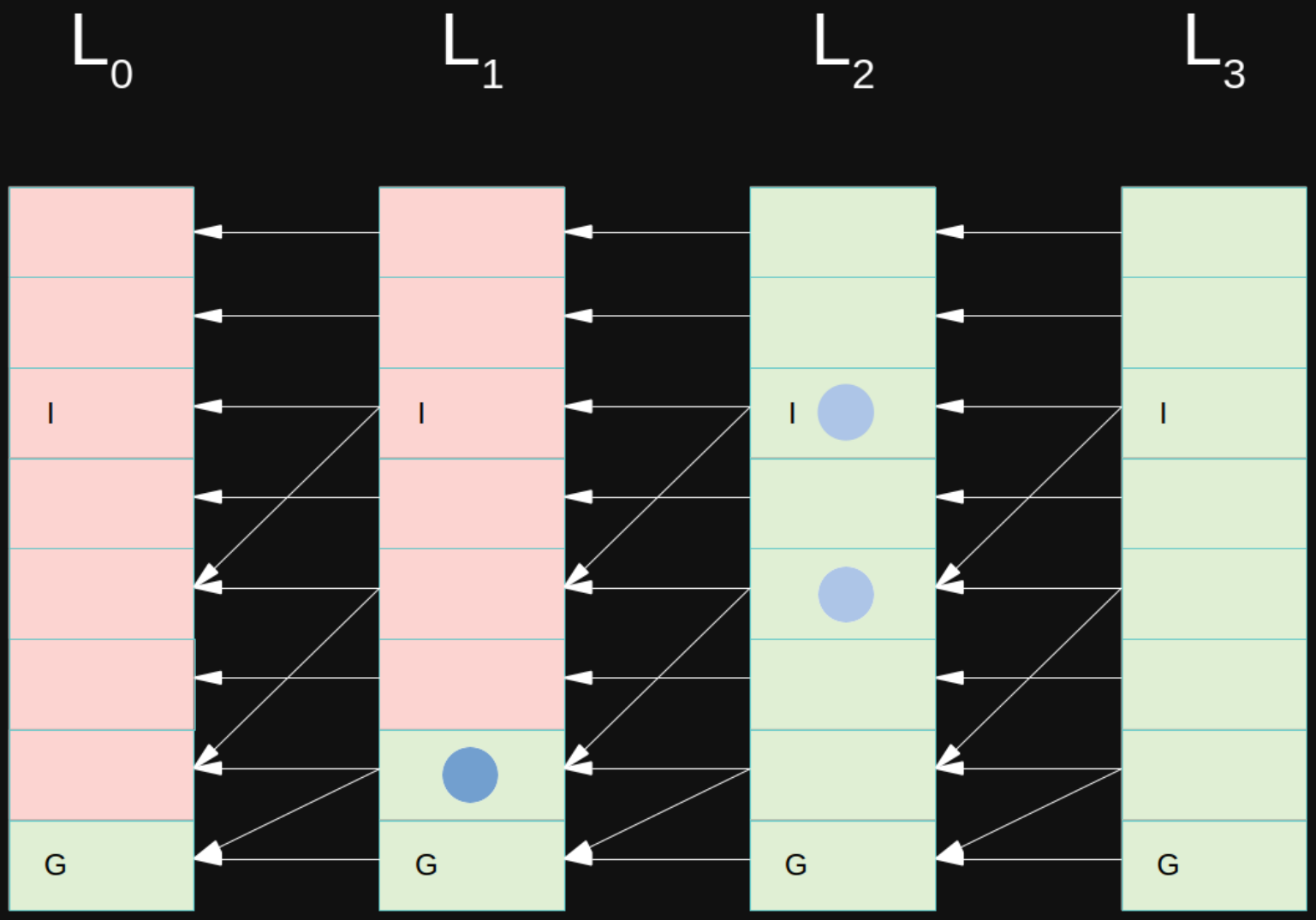
OR - Example



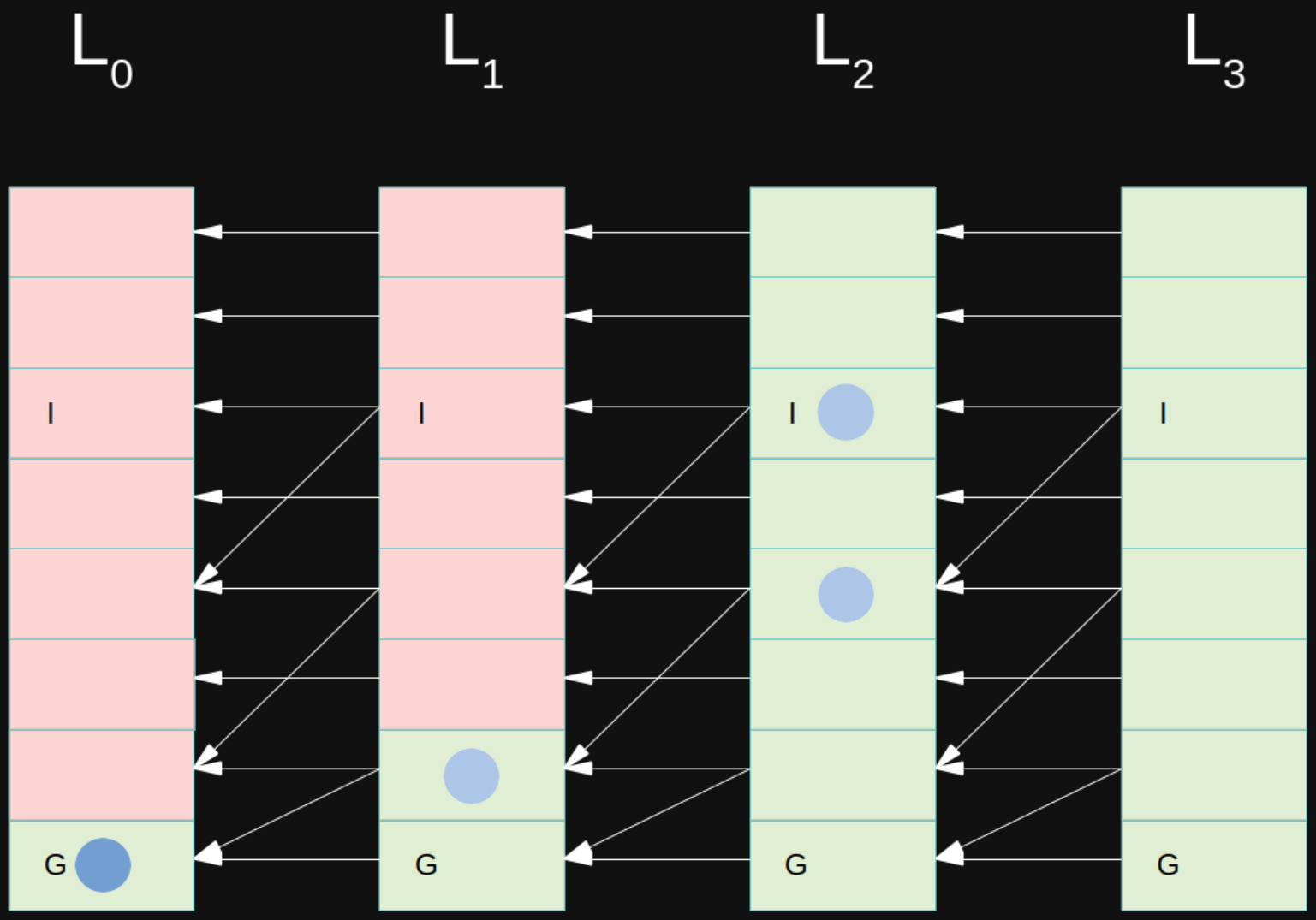
OR - Example



OR - Example



OR - Example



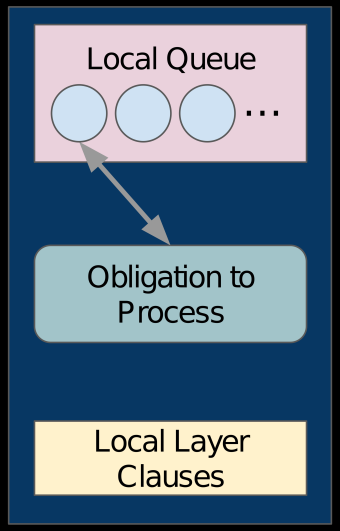
PDR - Algorithm

Input: A planning problem $\Pi = \langle X, A, I, G \rangle$
Output: *True* if the problem is solvable, *False* otherwise

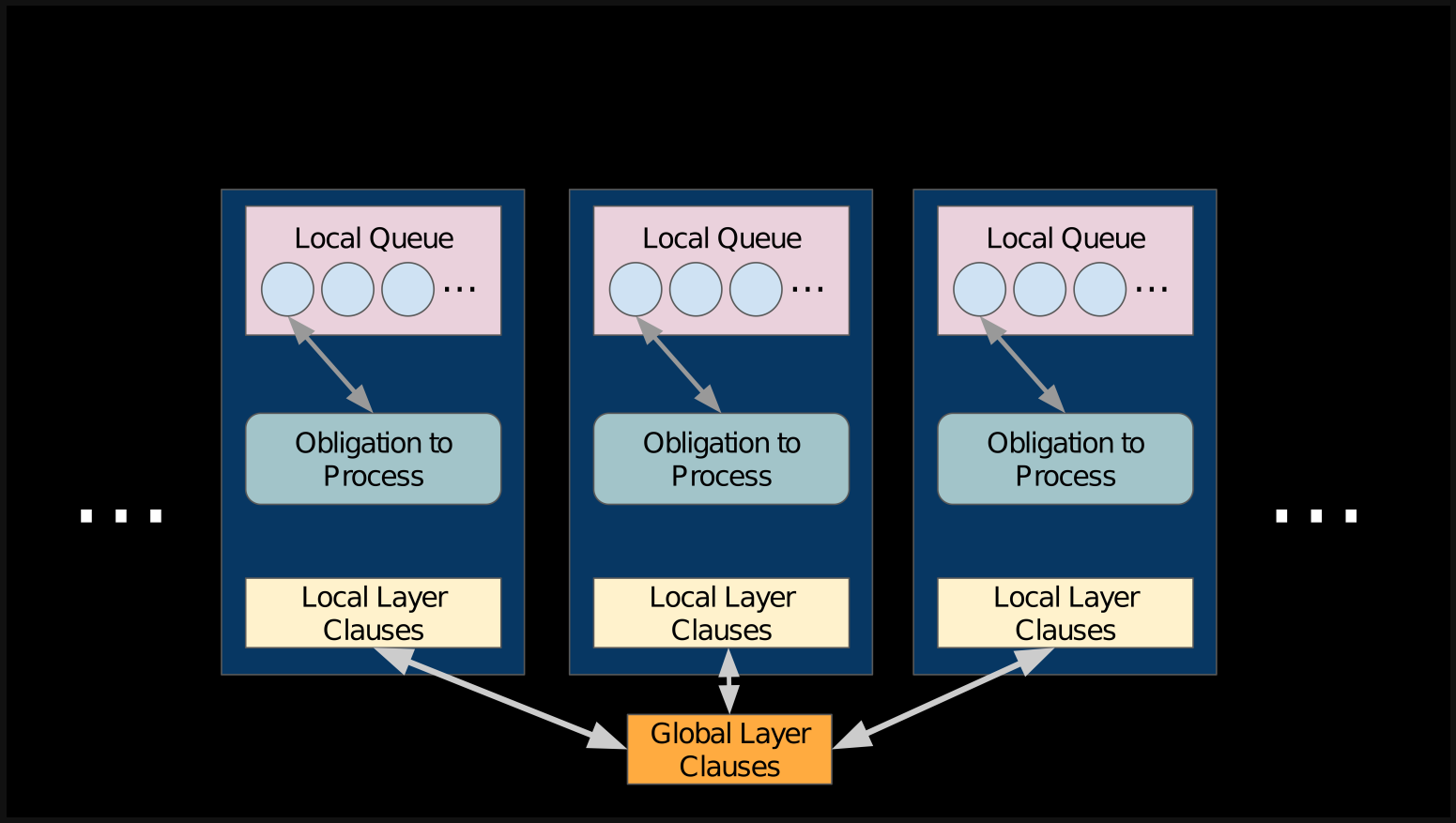
$Q \leftarrow \{\}$, $\mathcal{L}_0 \leftarrow G$, for $j > 0 : \mathcal{L}_j \leftarrow \mathit{True}$
if $I \vdash \mathcal{L}_0$ **then return** True
for $k \in [1, 2, 3, ...]$ **do**
  $Q \leftarrow \{\langle I, k \rangle\}$
  while Q *not empty* **do**
    $\langle s, i \rangle \leftarrow$ pop most recently added obligation from Q with minimal i.
    $\langle \mathit{exists}, s', u \rangle \leftarrow \Pi.\mathit{get\_succ}(s, \mathcal{L}_{i-1})$
    if *exists* **then**
      $Q \leftarrow Q \cup \{\langle s, i \rangle, \langle s', i-1 \rangle\}$
      if $i - 1 = 0$ **then return** True
    else
      $r \leftarrow \Pi.\mathit{compute\_reason}(\langle u, i \rangle, \mathcal{L})$
      for $j \in \{0, .., i\}$ **do** $\mathcal{L}_j \leftarrow \mathcal{L}_j \wedge \neg r$
      if $i < k$ **then** $Q \leftarrow Q \cup \{\langle s, i+1 \rangle\}$ /* Obligation Rescheduling */
    end
  end
  for $i \in \{1, .., k+1\}$ **do** /* Clause Pushing */
    foreach $c \in \mathcal{L}_{i-1}$ *and* $c \notin \mathcal{L}_i$ **do**
      if $\neg \Pi.\mathit{has\_succ}(\neg c, \mathcal{L}_{i-1})$ **then** $\mathcal{L}_i \leftarrow \mathcal{L}_i \wedge \neg c$
    end
    if $\mathcal{L}_{i-1} \equiv \mathcal{L}_i$ **then return** False
  end
end

Algorithm 2 Property Directed Reachability

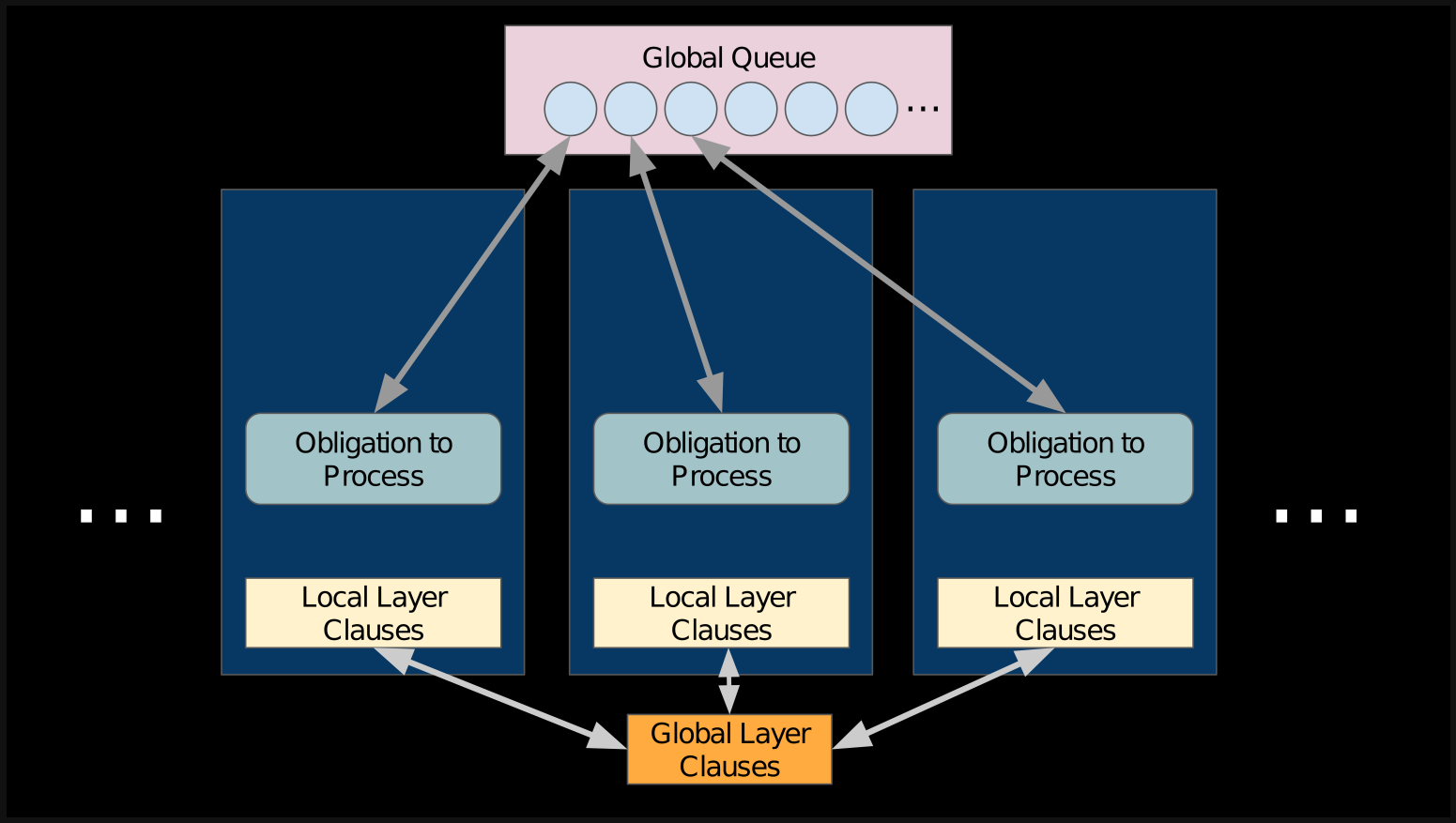
Serial PDR



Existing PDR Parallelizations



Parallel State PDR



Existing Parallelizations

- Portfolio of PDR processes, sharing reasons
- Without obligation rescheduling
- Not in planning
- Reimplemented for benchmarking

PS-PDR

- Single *orchestrator*, multiple *workers*
- Orchestrator manages queue, checks for convergence, shares layer formula
- Workers process obligations using local copy of layer information

PS-PDR Worker

Input: A planning problem $\Pi = \langle X, A, I, G \rangle$

1 **Loop**
2 $\quad \langle clause\_pushing, \mathcal{L}^{local\_copy}, \langle s, i \rangle \rangle \leftarrow get\_obligation()$
3 $\quad \langle exists, s', u \rangle \leftarrow \Pi.get\_succ(s, \mathcal{L}_{i-1}^{local\_copy})$
4 $\quad$ **if** *exists* **then**
5 $\quad\quad$ **if** $\neg clause\_pushing$ **then** $send\_success\_to\_orchestrator(\langle s', s, i \rangle)$
6 $\quad$ **else**
7 $\quad\quad$ **if** *clause_pushing* **then**
8 $\quad\quad\quad send\_failure\_to\_orchestrator(\langle NONE, s, i \rangle)$
9 $\quad\quad$ **else**
10 $\quad\quad\quad r \leftarrow \Pi.compute\_reason(\langle u, i \rangle, \mathcal{L}^{local\_copy})$
11 $\quad\quad\quad send\_failure\_to\_orchestrator(\langle r, s, i \rangle)$
12 $\quad\quad$ **end**
13 $\quad$ **end**
14 **EndLoop**

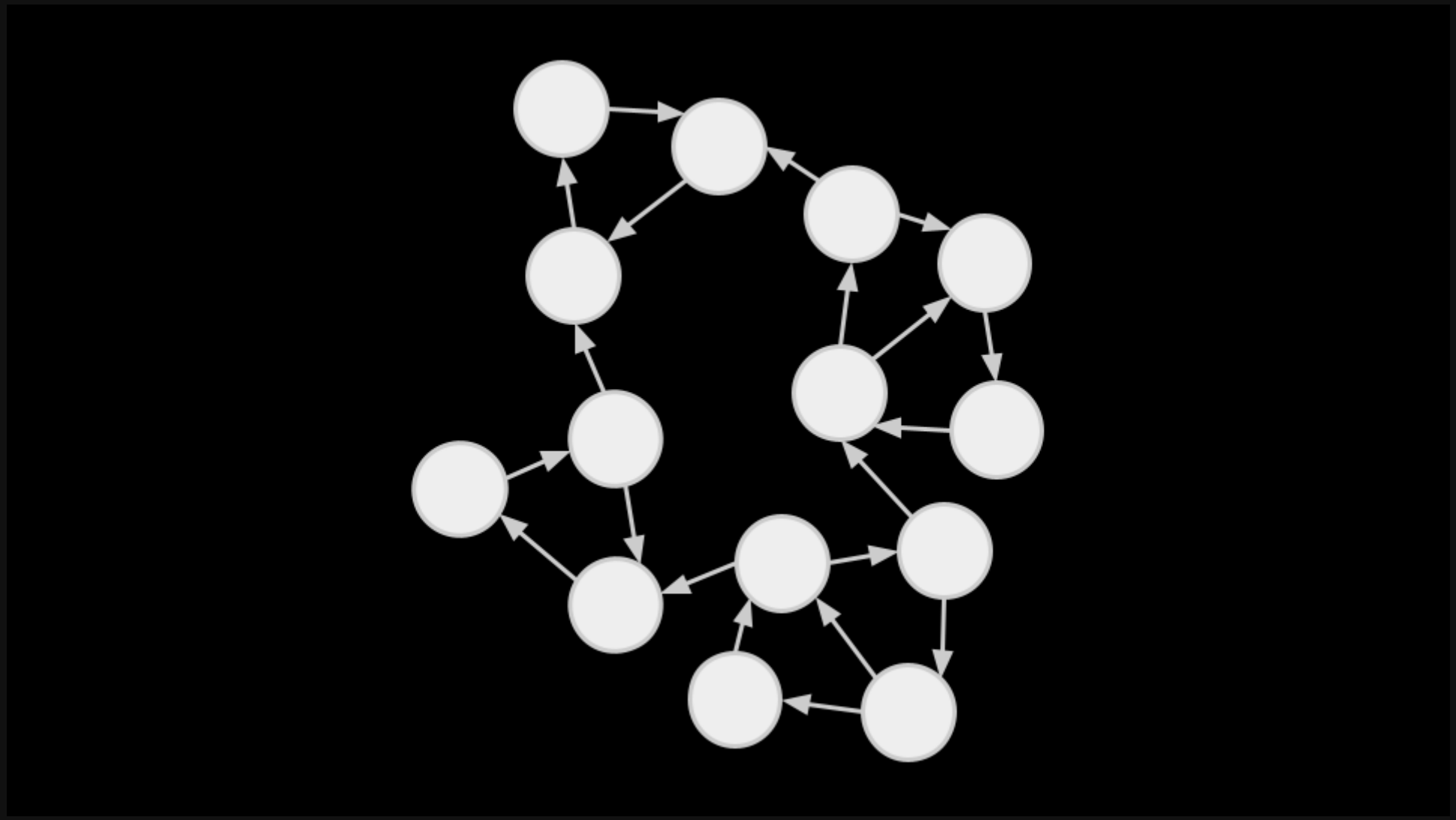
Algorithm 3 PS-PDR Worker

PS-PDR Orchestrator

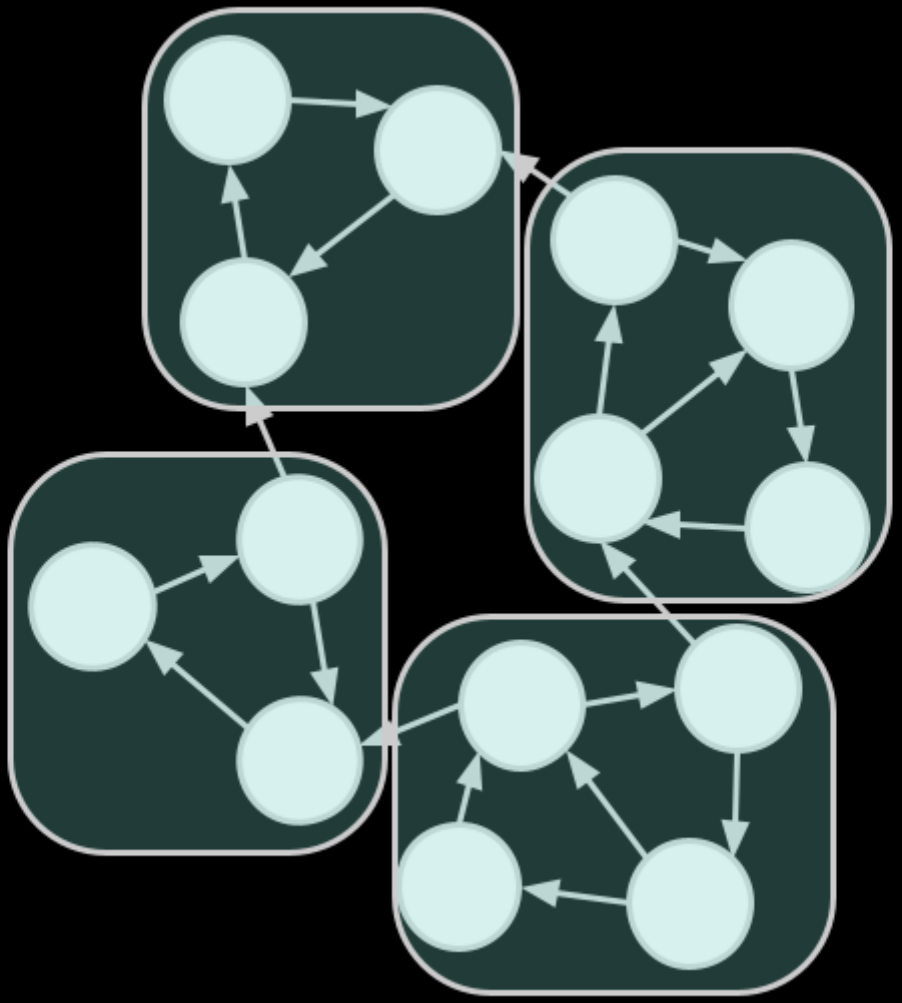
$Q \leftarrow \{\}, \mathcal{L}_0 \leftarrow G$, for $j > 0 : \mathcal{L}_j \leftarrow \mathit{True}$
if $I \vdash \mathcal{L}_0$ **then return** True
for $k \in [1, 2, 3, ...]$ **do**
    $Q \leftarrow \{\langle I, k\rangle\}$
    while $Q \neq \{\}$, *or* $\mathit{workers\_waiting\_for\_obligation}() \neq \{1, .., M\}$ **do**
        foreach $w \in \mathit{workers\_waiting\_for\_obligation}()$ **do**
            if $Q \neq \{\}$ **then**
                $\langle s, i\rangle \leftarrow$ pop most recently added obligation from Q with minimal i.
                $\mathit{send\_obligation\_to\_worker}(\langle \mathit{False}, \mathcal{L}, \langle s, i\rangle\rangle, w)$
            end
        end
        foreach $\langle s', s, i\rangle \in \mathit{get\_successes\_from\_workers}()$ **do**
            $Q \leftarrow Q \cup \{\langle s, i\rangle, \langle s', i-1\rangle\}$
            if $i - 1 = 0$ **then return** True
        end
        foreach $\langle r, s, i\rangle \in \mathit{get\_failures\_from\_workers}()$ **do**
            for $j \in \{0, .., i\}$ **do** $\mathcal{L}_j \leftarrow \mathcal{L}_j \wedge \neg r$
            if $i < k$ **then** $Q \leftarrow Q \cup \{\langle s, i+1\rangle\}$ /* Obligation Rescheduling */
        end
    end
    for $i \in \{1, .., k+1\}$ **do**
        ...
        /* Clause Pushing */
        ...
        if $\mathcal{L}_{i-1} \equiv \mathcal{L}_i$ **then return** False
    end
end

Algorithm 4 PS-PDR Orchestrator

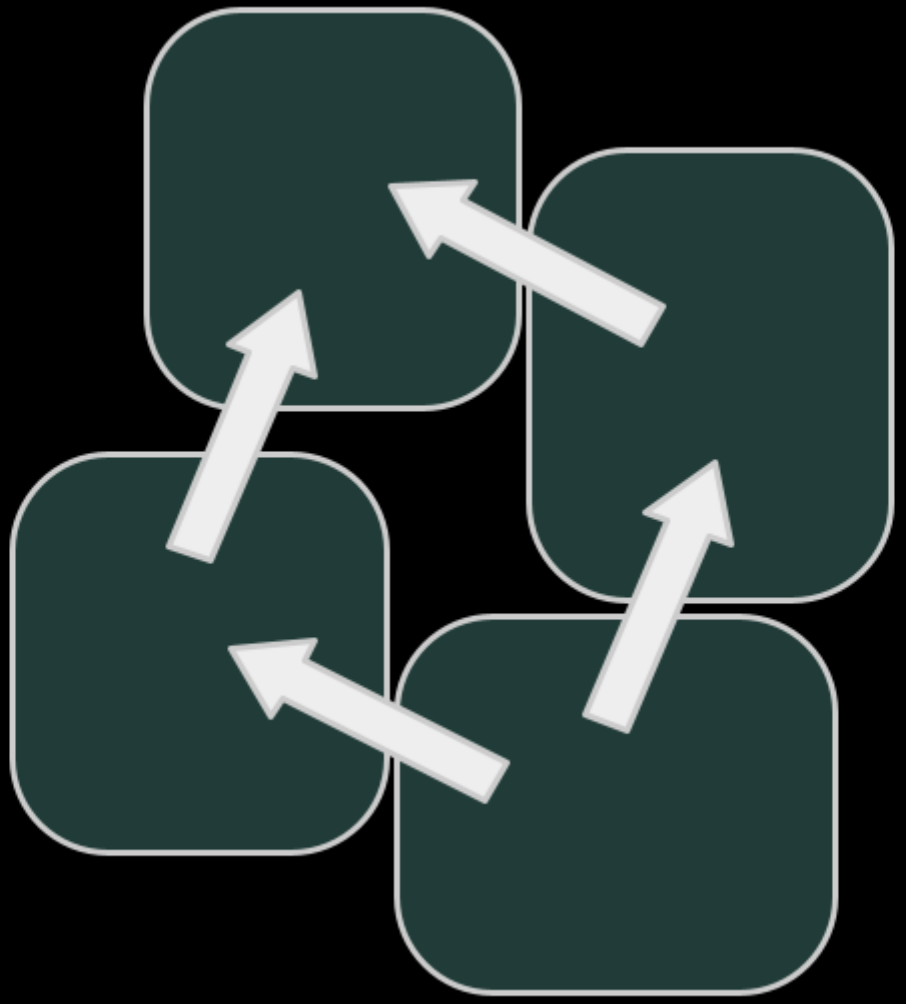
Strongly Connected Components



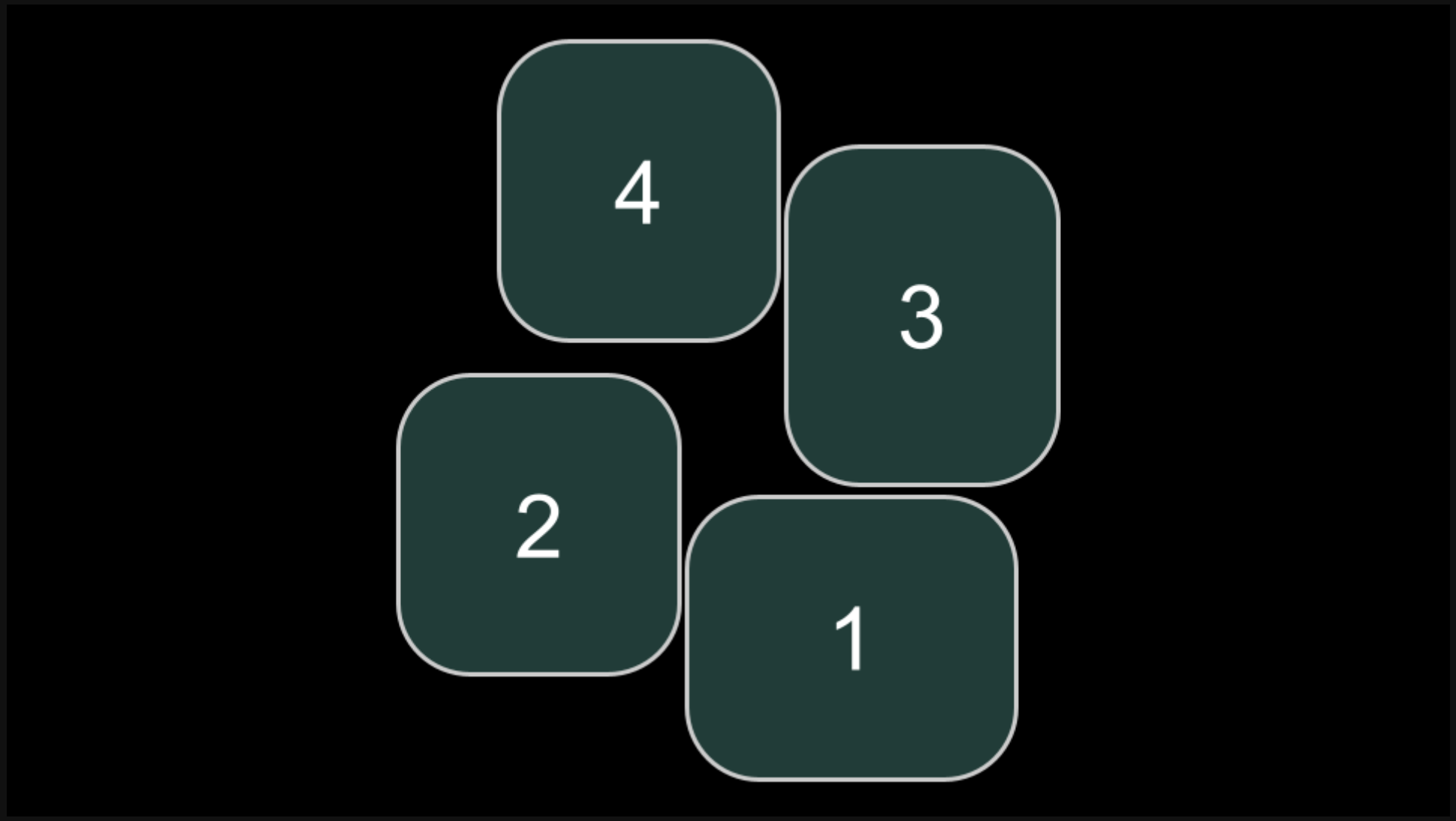
Strongly Connected Components



Strongly Connected Components



Strongly Connected Components


4
3
2
1

Parallel Decompositional PDR

- Project the problem into subproblems
- Solve the projections individually in parallel

Example Logistics Domain

- Two locations: $L1, L2$
- Two packages: $P1, P2$
- One truck: T
- The truck can be at either location.
- Package can be at either location, or in the truck.
- Actions drive truck, or (un)load packages in place.
- What invariants might be found?

Dependency Graph

- $at(P1, L1) \rightarrow at(T, L1)$
  - Package location *depends* on truck location
  - Not the other way around
- $at(P1, L1) \leftrightarrow at(P1, L2)$
  - Being at A *depends* on not being at B

Knoblock, C. A. 1994. Automatically generating abstractions for planning.
Williams, B. C.; and Nayak, P. P. 1997. A reactive planner for a model-based executive.

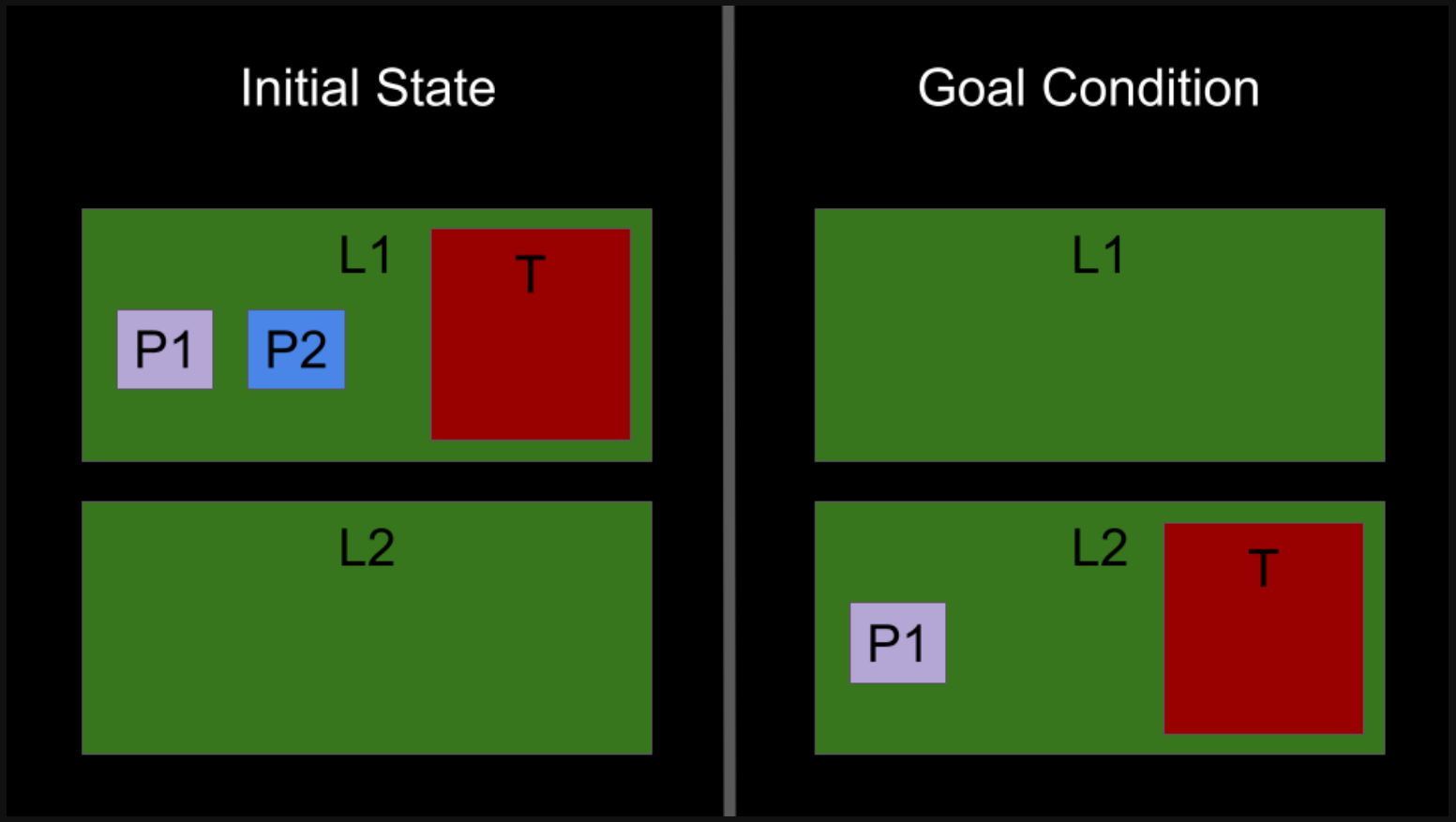
Planning in Abstraction Refinement

- Sort SCCs E.g. $SCC_{P1}, SCC_{P2}, SCC_{T}$
- Produce abstract plans to be refined

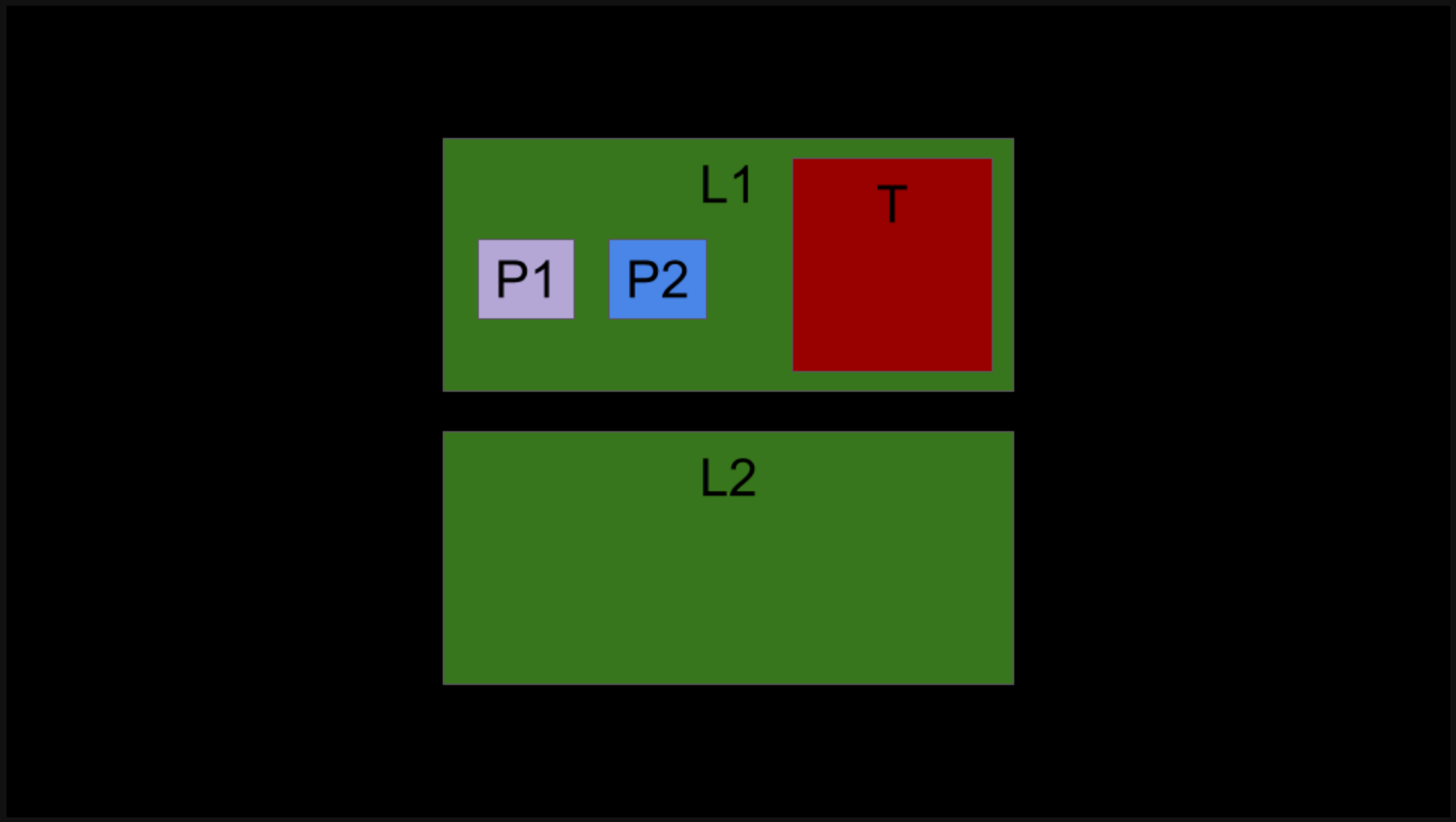
PD-PDR

- Sound but not complete
- Planner interchangeable
- Sort SCCs
- For each SCC:
  - Plan for relevant goals
  - Return everything else to its starting position.
- If all have plans, concatenate to form concrete plan.
- Check concatenation
- Instead plan for each SCC in parallel

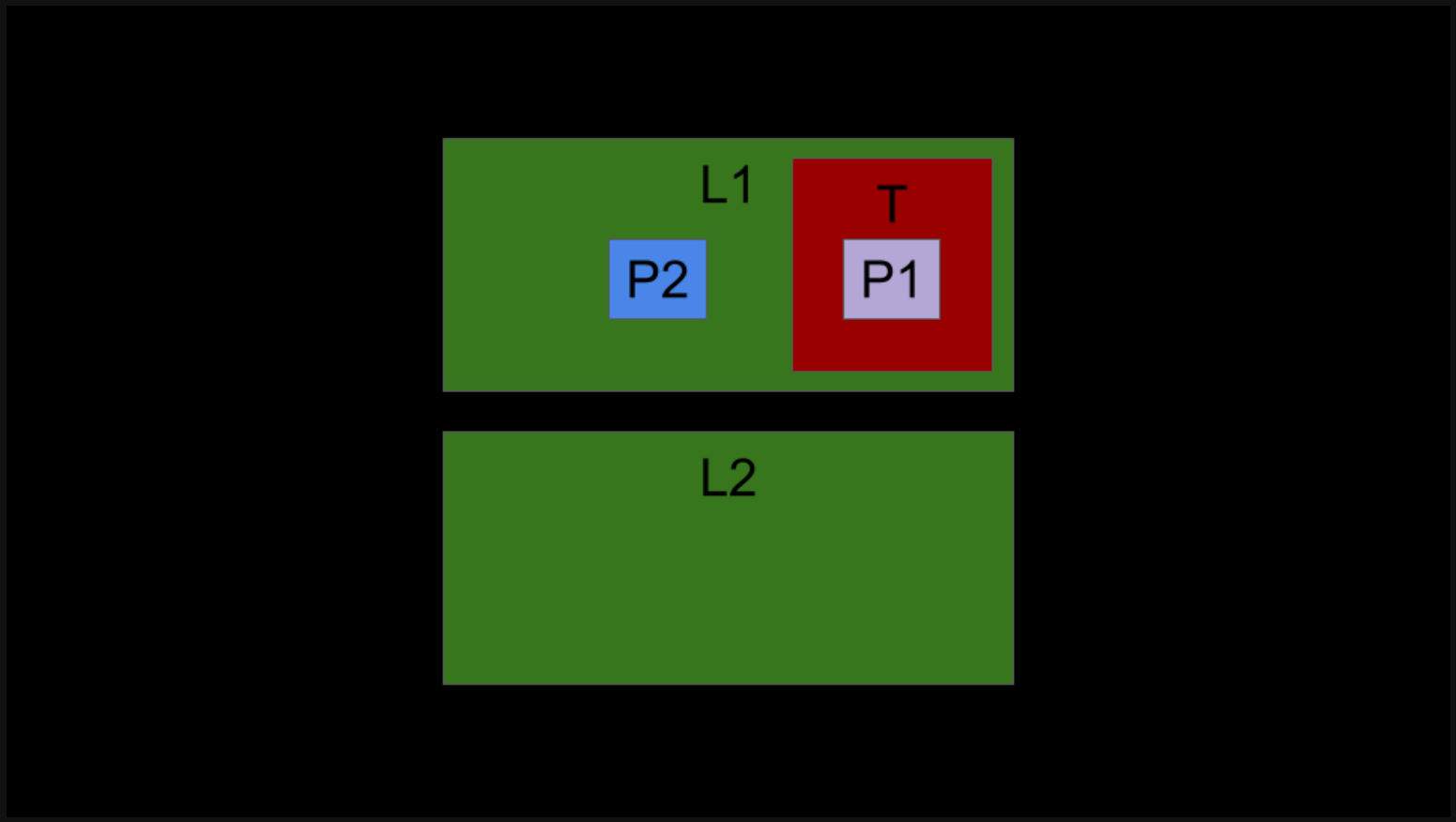
PD-PDR - Example



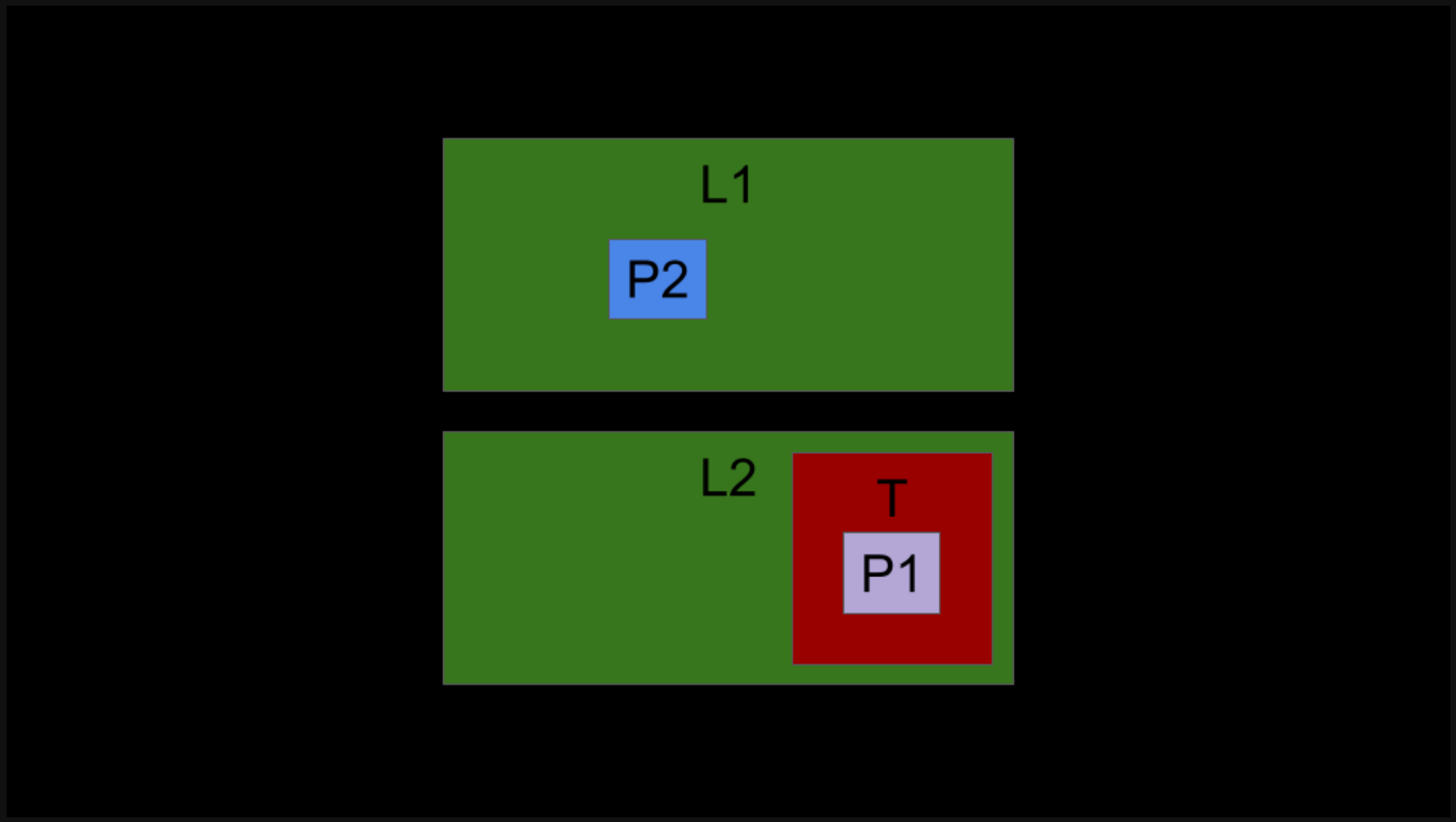
PD-PDR - Example



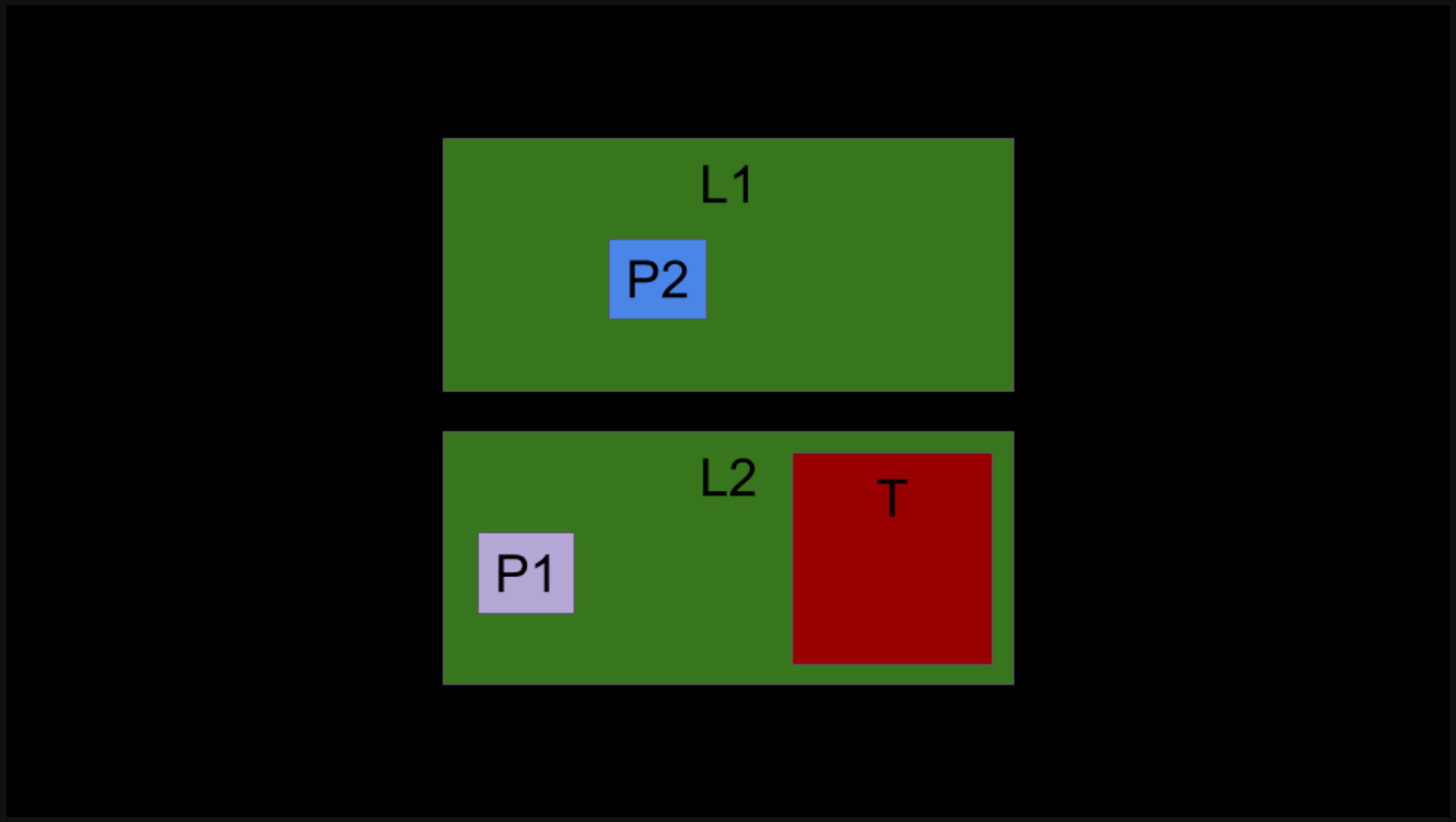
PD-PDR - Example



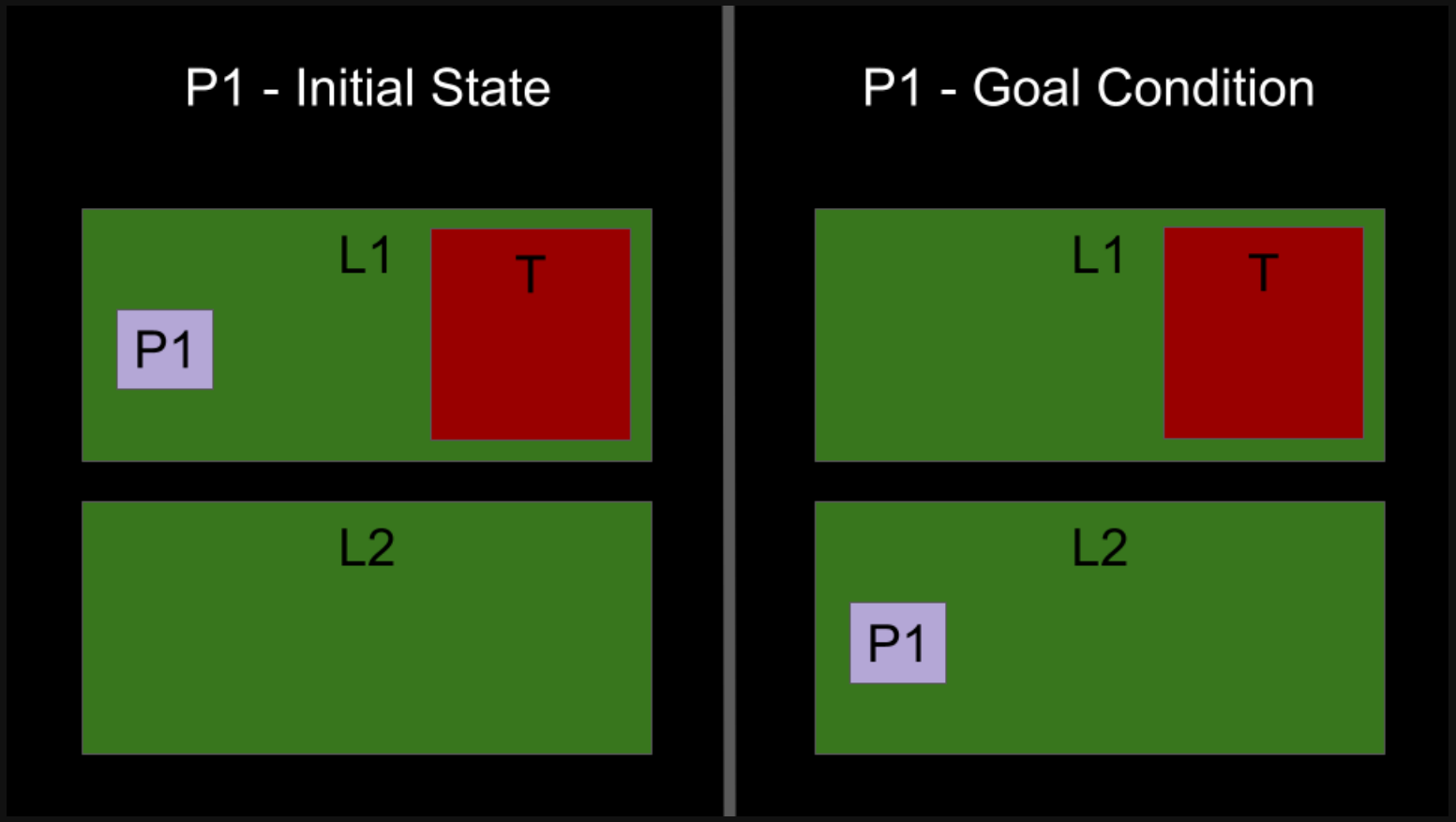
PD-PDR - Example



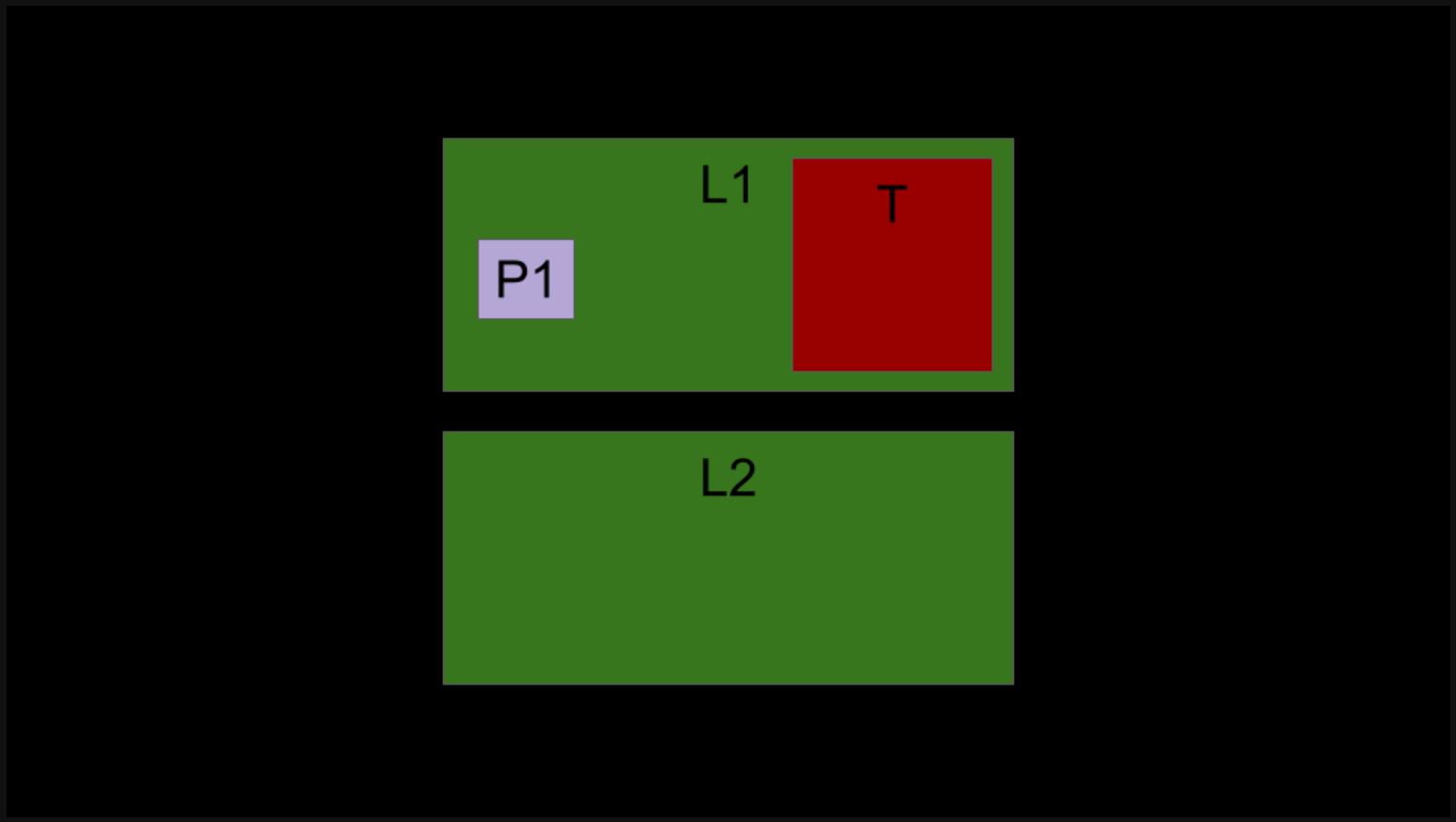
PD-PDR - Example



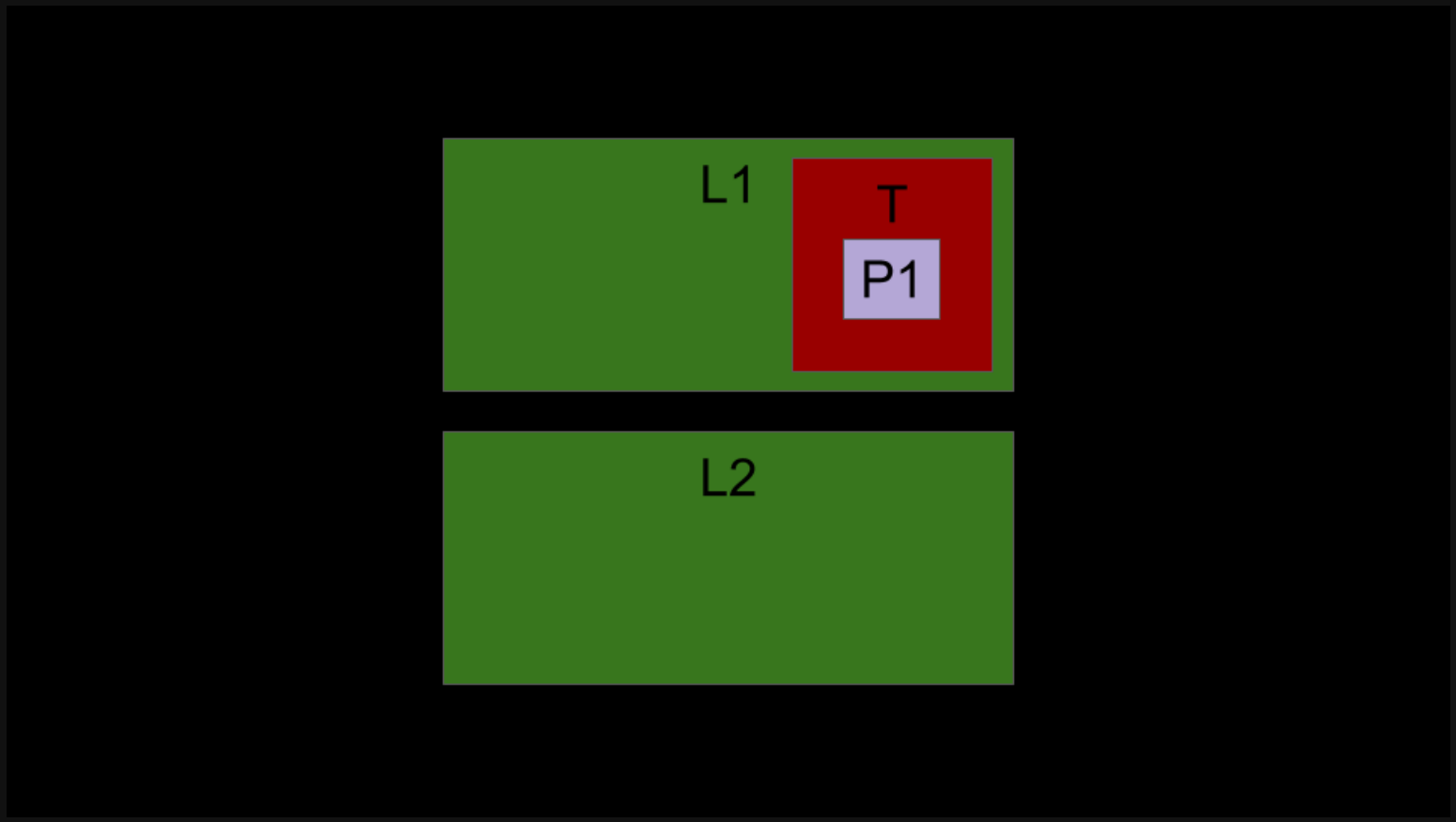
PD-PDR - Example



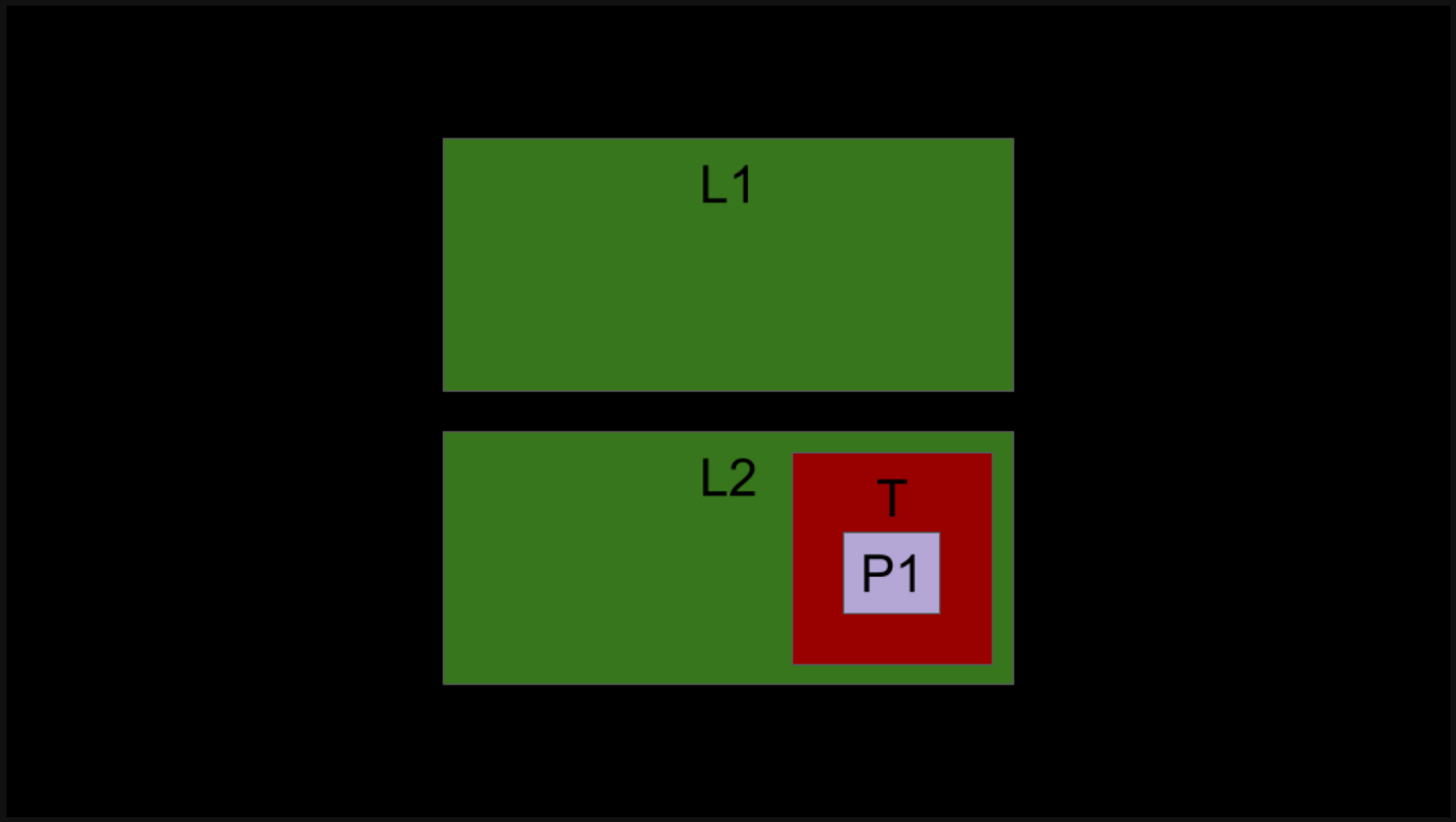
PD-PDR - Example



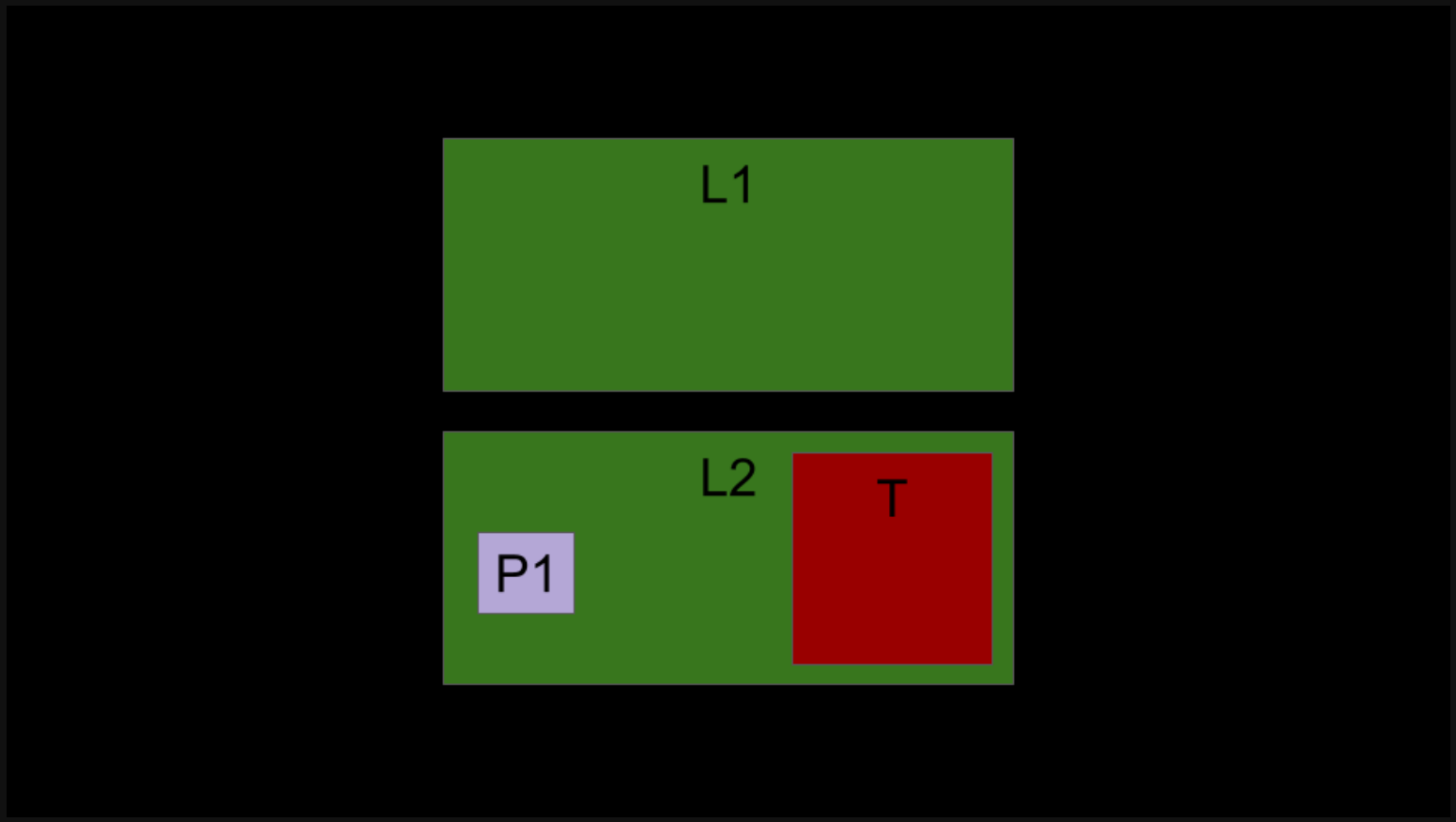
PD-PDR - Example



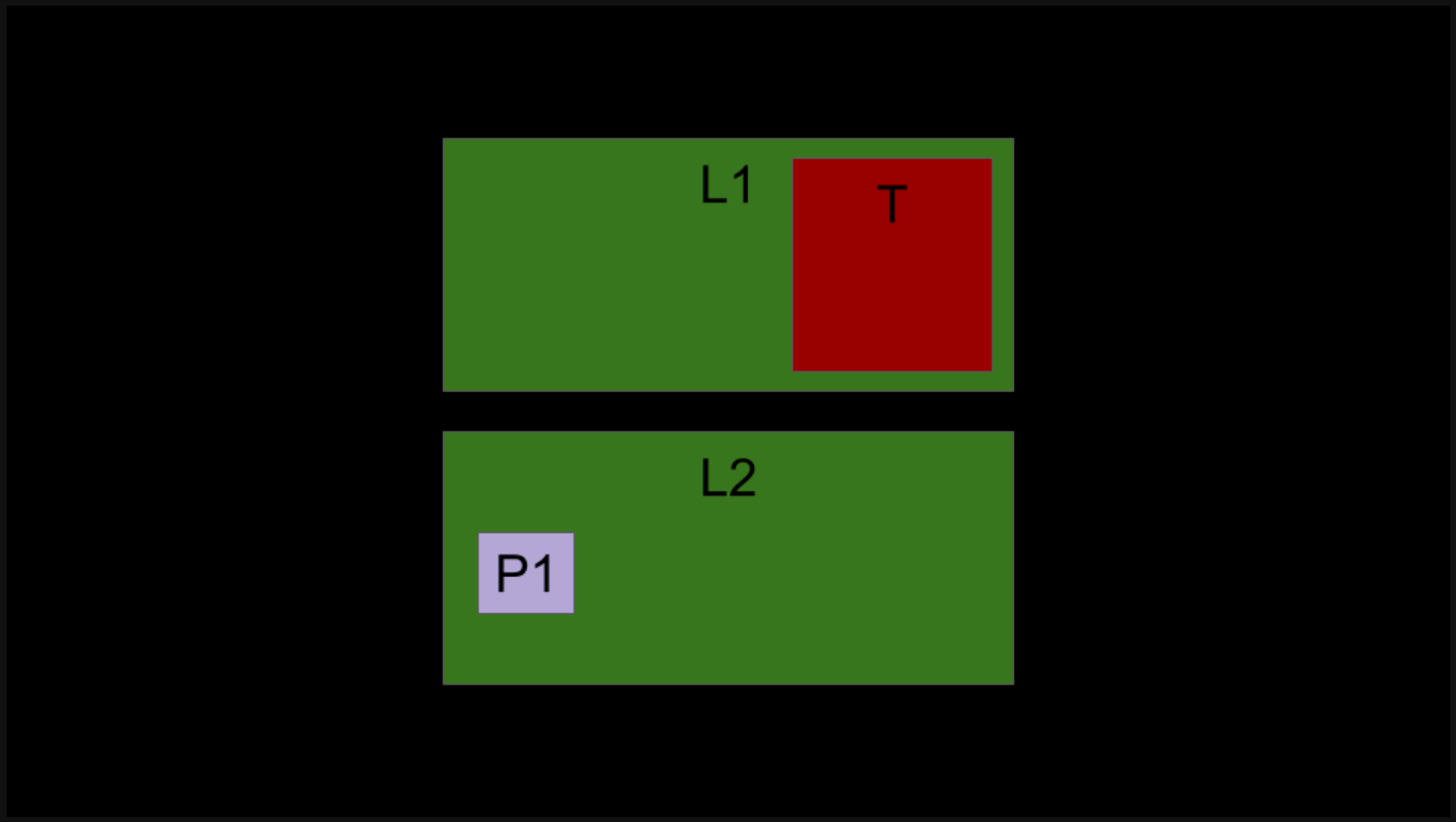
PD-PDR - Example



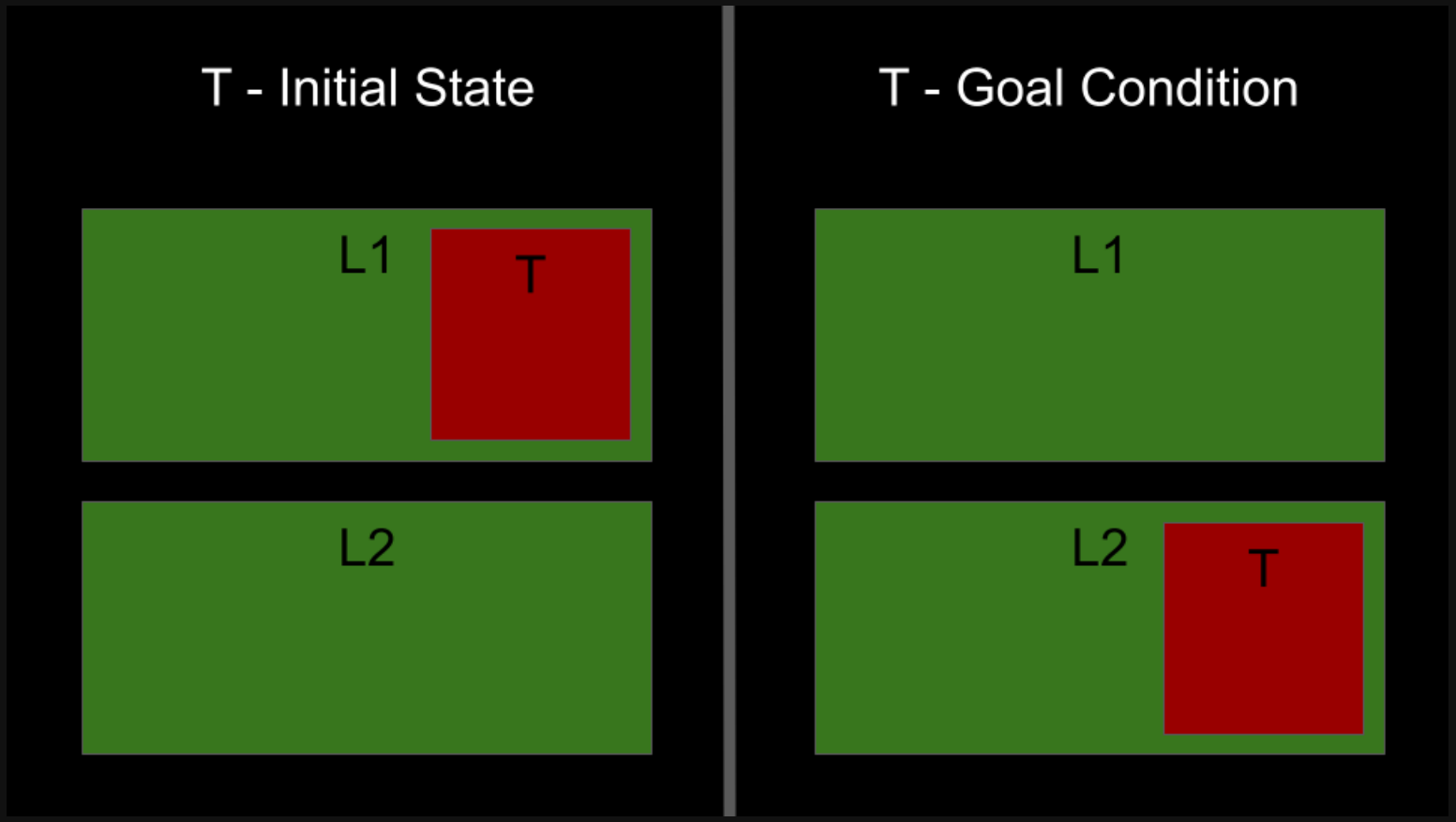
PD-PDR - Example



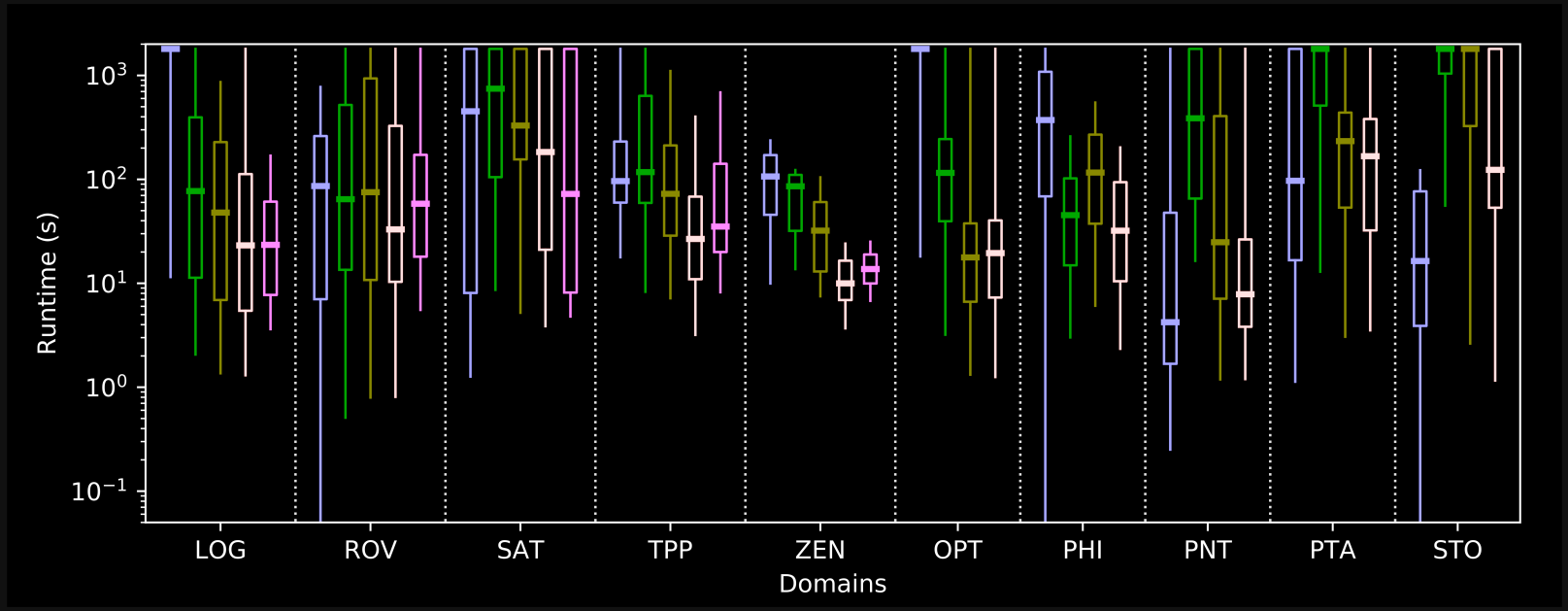
PD-PDR - Example



PD-PDR - Example



Results



PDRPLAN SERIAL PDR PORTFOLIO PDR
PS-PDR PD-PDR

Thank You

Interested in a related honours project?
Contact me (Marshall Clifton), Charles Gretton or Mark Burgess (first.last@anu.edu.au)

Scholarships may be available for Australian Citizens.

Slides: https://clifton-m.github.io/ai_guest_lecture/slides/index.html

Clifton, M., Gretton, C. 2021. Computing Multiple PDR Steps in a Single SAT Call and a PDR Comparison to Madagascar with Completeness Thresholds.
Clifton, M., Gretton, C. 2022. Fast Parallel PDR Algorithms for Planning.