



Exploring the Design Space of Isolation Mechanisms

Deriving isolation mechanisms from goals and constraints with **IsoSearch**

Sidhartha Agrawal* · Shaurya Patel* · Linh Pham~ · Arya Stevinson@ · Ilias Karimalis*

Hugo Lefeuvre* · Aastha Mehta* · Reto Achermann*^ · Margo I. Seltzer*

Systopia Lab, University of British Columbia · Hammerspace~ · Oracle@ · Technical University of Munich^*

14th Workshop on Programming Languages and Operating Systems (PLOS '26)

September 29 – October 2, 2026 · Prague, Czech Republic



The problem and our solution

- Processes, containers, and VMs are just **three points** in a vast design space, each with a different trade-off of performance, isolation, and sharing.
- Building **custom** isolation is ad-hoc
 - Requirements are stated **informally**: “the key server must not be reachable from the network”
- Misconfiguration is a leading source of exposure *[Qualys 2024]*.

Our Solution

Isolation mechanisms are **points in a design space** that we can explore **algorithmically**.



The Big Picture



1. **LinTool** extracts an OSmosis graph from a live system *[OSmosis, PLOS 2025]*
2. **IsoSearch** searches the design space for a graph meeting the developer's goals & constraints ← [this paper](#)
3. **Instantiate** the discovered graph on a target OS: Linux or the capability-based CellulOS *[CellulOS, seL4 Summit 2025]*

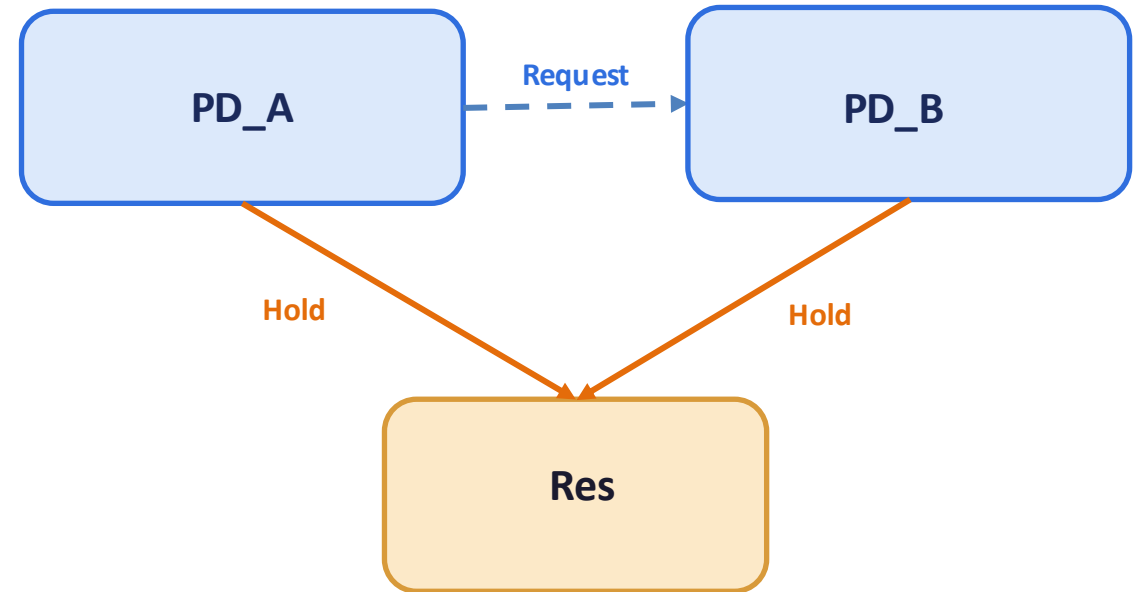


A Subset of the OSmosis model

A system is a graph.

- **Nodes:**
 - Protection Domains
 - Resources (what they use)
- **Edges:**
 - Hold (a PD owns a resource)
 - Request (a PD asks another PD for resource)
- Same resource held twice = **sharing** → Resource Similarity Index (RSI)
- Who you must Request from = **trust** → Trusted Computing Base (TCB)

Prior work [OSmosis, PLOS '25]





How the Search is setup

G₀

a starting graph

an existing deployment

C

functional constraints

what must hold

O

an optimization goal

min / max a metric

Find a graph **G'** that satisfies **C** and optimizes **O**.

Resource Similarity Index NEW

resource sharing between two PDs.
Lower $\Rightarrow$ stronger isolation.

Memory Constraint NEW

memory:
 Σ resource sizes (each once) +
per-PD overhead.

TCB REUSED

trust-set size $|\text{TCB}(\text{PD})|$, from previous
work



The moves: graph transitions

Twelve transitions (create/delete a node, add/delete an edge) each guarded by **preconditions** that preserve well-formedness.

Element	Transition	Precondition
PD	Create / Delete	delete only if no outgoing edges
Resource	Create / Delete	+ Hold & Subset edges; delete if single Hold
Resource Space	Create / Delete	+ Hold edge; delete if no Subset edges
Hold edge	Add / Delete	cannot remove the last Hold on a resource
Request edge	Add / Delete	Both PDs need to exists
Map edge	Add / Delete	between same-type nodes

Correctness

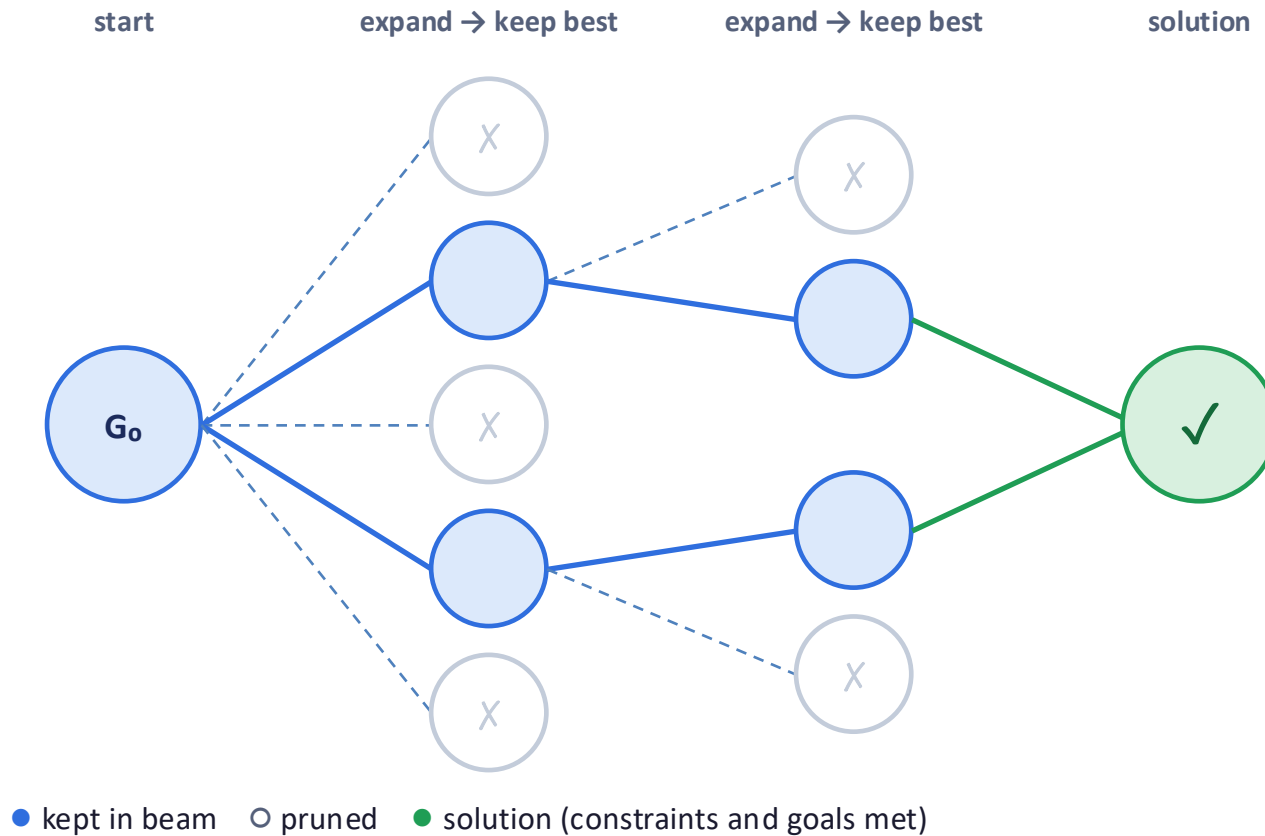
$$wf(G_i) \wedge Pre(T_i) \Rightarrow wf(G_{i+1})$$

By induction, applying n transitions to a valid G_0 yields a valid G_n .

Mechanised in Verus.



What is beam search?



THE LOOP

Keep the best **k** candidates each round:

- expand (every legal transition)
- keep top-k by score
- prune the rest.

$$\text{Score } W = \alpha S + \beta P$$

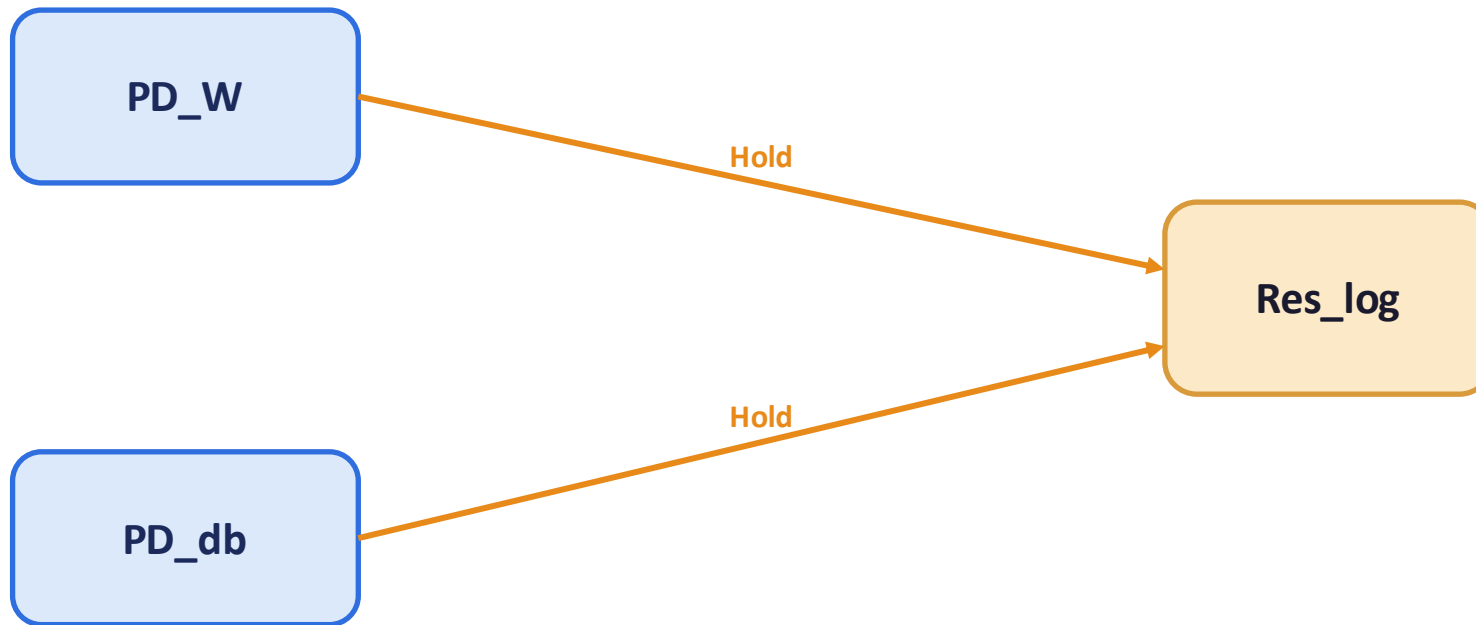
$\alpha/\beta = 50$ (constraints are prioritized over goals).

Why beam? greedy is brittle; full BFS explodes. We use $k = 12$.



Example Walkthrough: Mediation

Step 0/6 · G_0 : both PDs hold the log directly (shared)



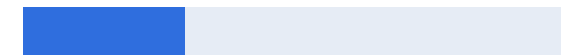
CONSTRAINTS(x2) and Goal

- ✓ Both reach Res_log
- ✗ Single holder of log

Minimize ($RSI(PD_W, PD_db)$)

1.0 *(shared)*

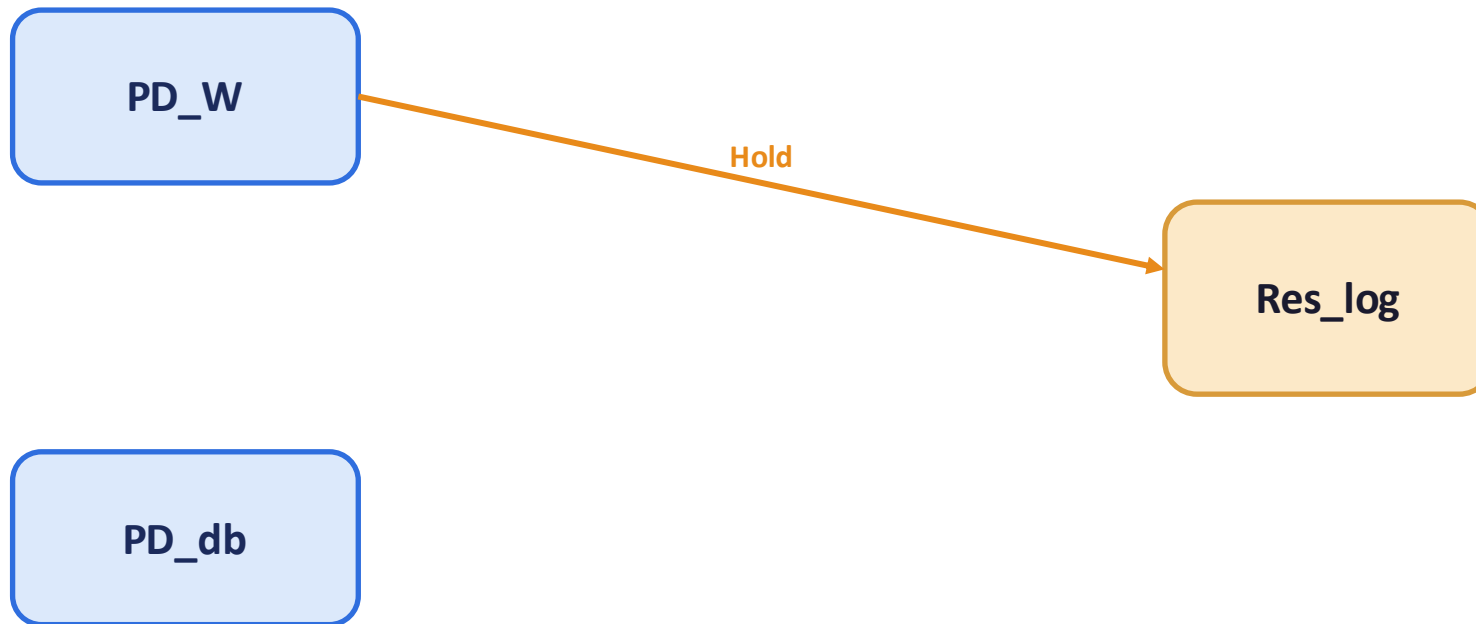
search score





Example Walkthrough: Mediation

Step 1/6 · Goal Met, but constraints violated



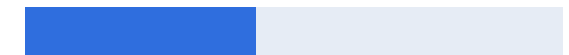
CONSTRAINTS(x2) and Goal

-  Both reach Res_log
-  Single holder of log

Minimize ($RSI(PD_W, PD_db)$)

0.0 *(isolated)*

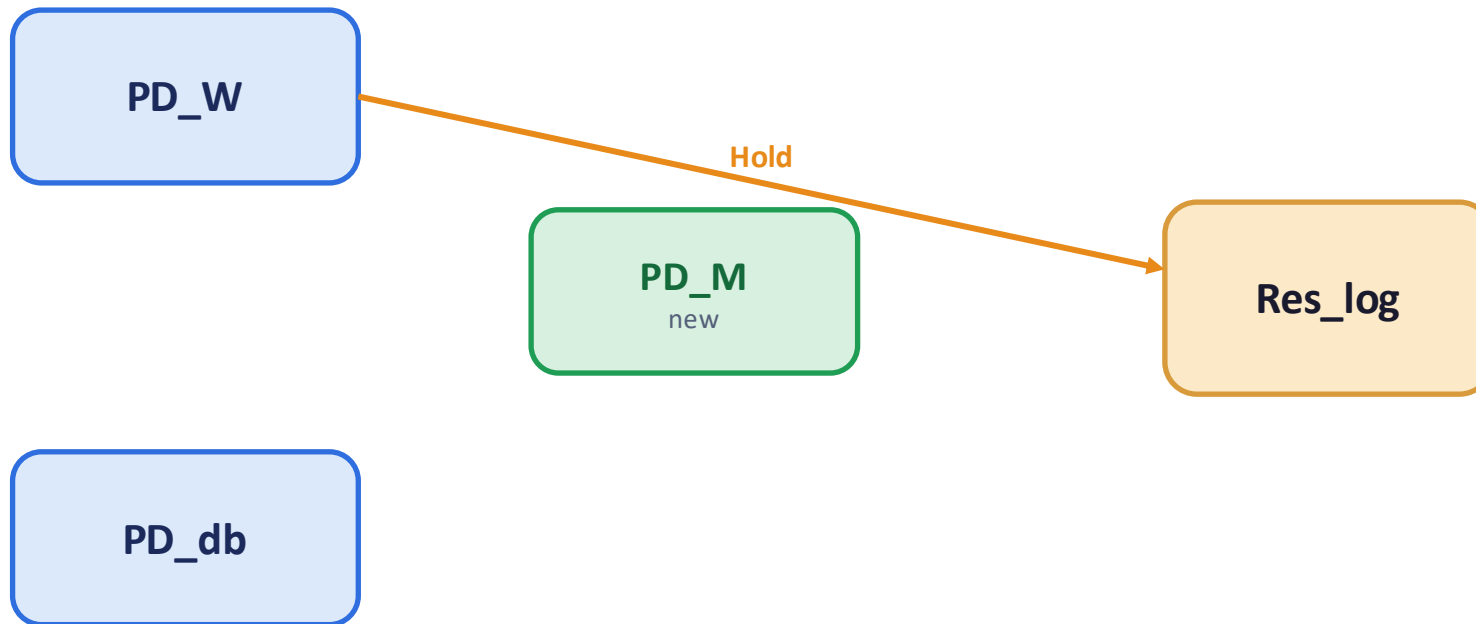
search score





Example Walkthrough: Mediation

Step 2/6 · One of the beam candidates adds a new PD



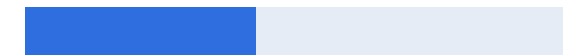
CONSTRAINTS(x2) and Goal

- ✗ Both reach Res_log
- ✓ Single holder of log

Minimize ($RSI(PD_W, PD_db)$)

0.0 *(isolated)*

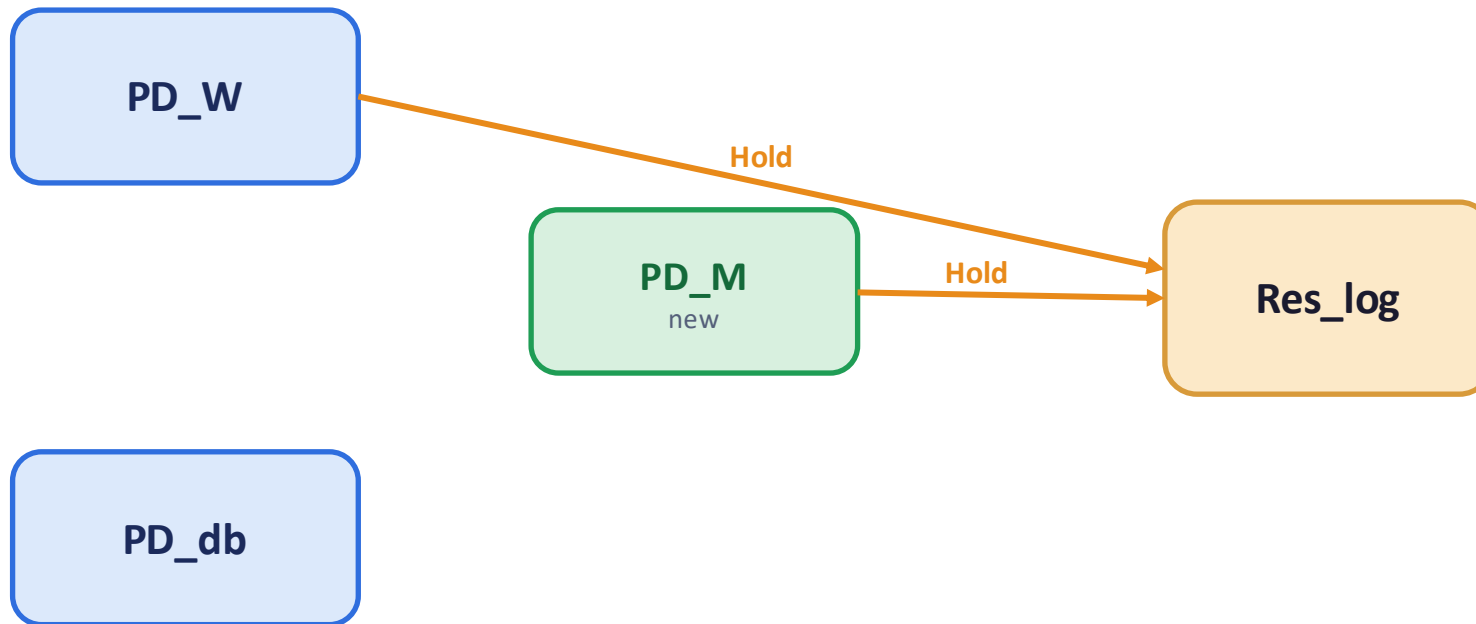
search score





Example Walkthrough: Mediation

Step 3/6 · Attach new PD to existing resources



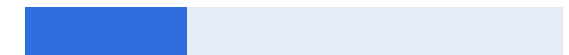
CONSTRAINTS(x2) and Goal

- ✗ Both reach Res_log
- ✗ Single holder of log

Minimize ($RSI(PD_W, PD_db)$)

0.0 *(isolated)*

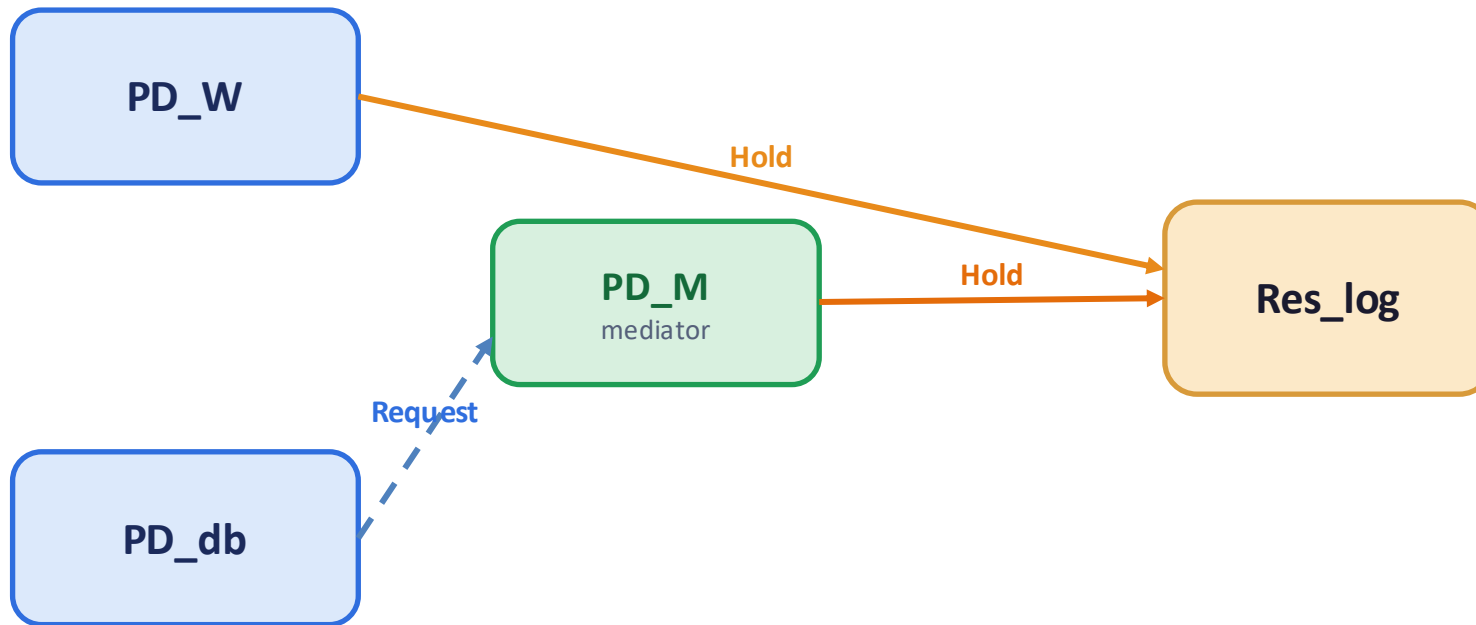
search score





Example Walkthrough: Mediation

Step 4/6 · Step 5/6 · Rewire through the mediator, but constraints don't hold



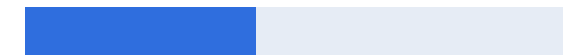
CONSTRAINTS(x2) and Goal

- ✓ Both reach Res_log
- ✗ Single holder of log

Minimize ($RSI(PD_W, PD_db)$)

0.0 *(isolated)*

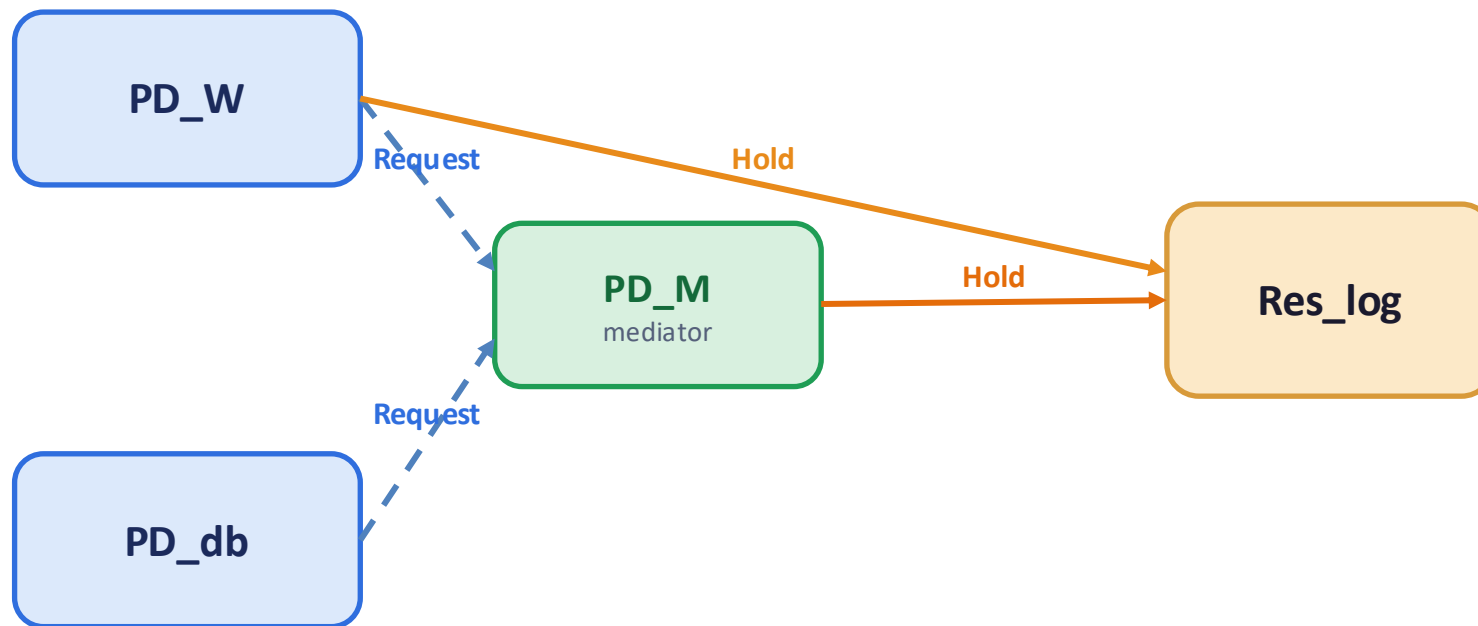
search score





Example Walkthrough: Mediation

Step 5/6 · Rewire through the mediator, but constraints don't hold



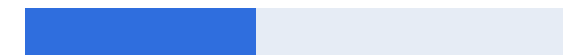
CONSTRAINTS(x2) and Goal

- ✓ Both reach Res_log
- ✗ Single holder of log

Minimize ($RSI(PD_W, PD_db)$)

0.0 *(isolated)*

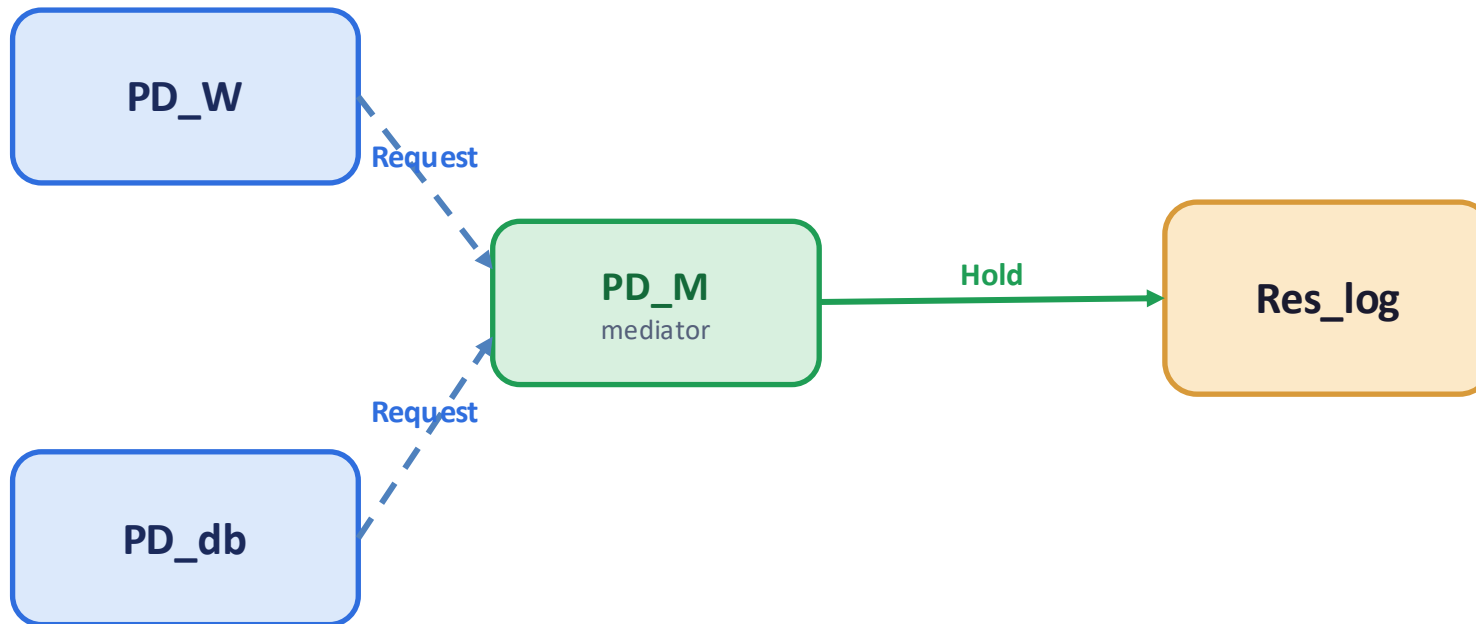
search score





Example Walkthrough: Mediation

Step 6/6 · Remove hold edge to satisfy constraints



CONSTRAINTS(x2) and Goal

- ✓ Both reach Res_log
- ✓ Single holder of log

Minimize (RSI(PD_W, PD_db))

0.0 *(isolated)*

search score



6 steps · 2,439 candidates: nobody specified a mediator



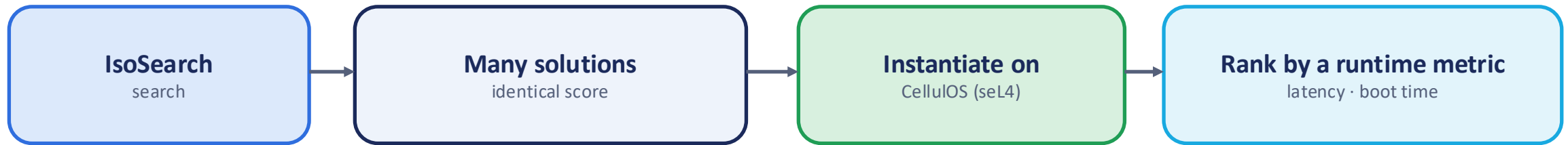
Five case studies

Case study	Goal(s)	Steps	# Eval	# Sol	Time
Mediation	argmin RSI	6	2439	8	2.2 s
Privatization	argmin RSI	2	1246	22	1.1 s
Sharing	argmax RSI	2	4096	37	6.0 s
Tenant isolation	argmin $\overline{RSI}$; Mem $\leq$ 320 MB	8	2798	12	3.1 s
Privilege separation	argmin TCB of 2 PDs, $\overline{RSI}$; Mem $\leq$ 530 MB	6	10916	109	20.5 s

IsoSearch produces many solutions, so, we go to hardware.



Beyond the model: real-hardware evaluation



- The model scores designs, but **ties are common**
- The model can't rank ties as it **can't see boot time, latency**.
- Instantiating on **CellulOS** adds those real dimensions and validates the graph (its extracted graph matches IsoSearch's output).

HARDWARE BREAKS THE TIE

Two designs the model rates **equal** (both $RSI = 0$), shared mediator vs. two private PDs, differ by **+9.7% boot time** on CellulOS.

An additional cost of the extra protection domain in boot time, invisible at the model level.



Takeaways

- IsoSearch derives isolation mechanisms from goals and constraints
- It finds mechanisms developers never specified
- Real hardware+OS narrows the derivation further

github.com/ubc-systopia/plos2026-isosearch

This work was supported by NSERC. Thanks to the anonymous PLOS reviewers.