Nickel: A framework for design and verification of information flow control systems

Luke Nelson
Joint work with Helgi Sigurbjarnarson, Bruno Castro-Karney, James Bornholt, Emina Torlak, Xi Wang

2018 New England Systems Verification Day

UNSAT
PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

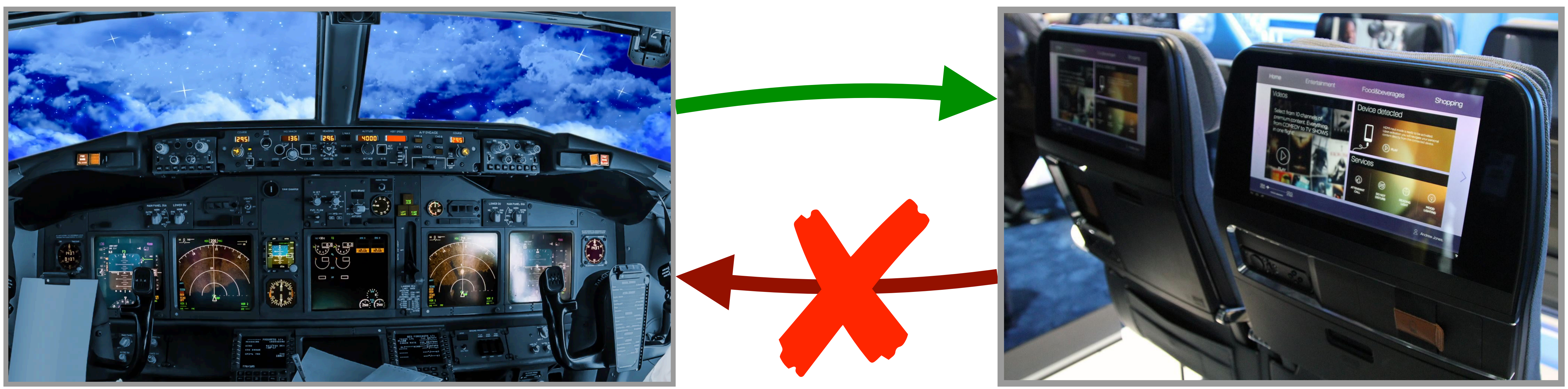
Motivation: high verification burden

- Verification is effective at eliminating bugs
- Requires expertise
- Large time investment

Approach: push-button verification

- Yggdrasil, OSDI 2016 — Crash-safe filesystems (Python)
- Hyperkernel, SOSP 2017 — Small OS kernel (C, memory isolation)
- Nickel, OSDI 2018 — Information flow control systems

Information flow control systems



CNN U.S. » Crime + Justice | Energy + Environment | Extreme Weather | Space + Science Live TV U.S. Edition +

FBI: Hacker claimed to have taken over flight's engine controls

By **Evan Perez**, CNN

Updated 9:19 PM ET, Mon May 18, 2015

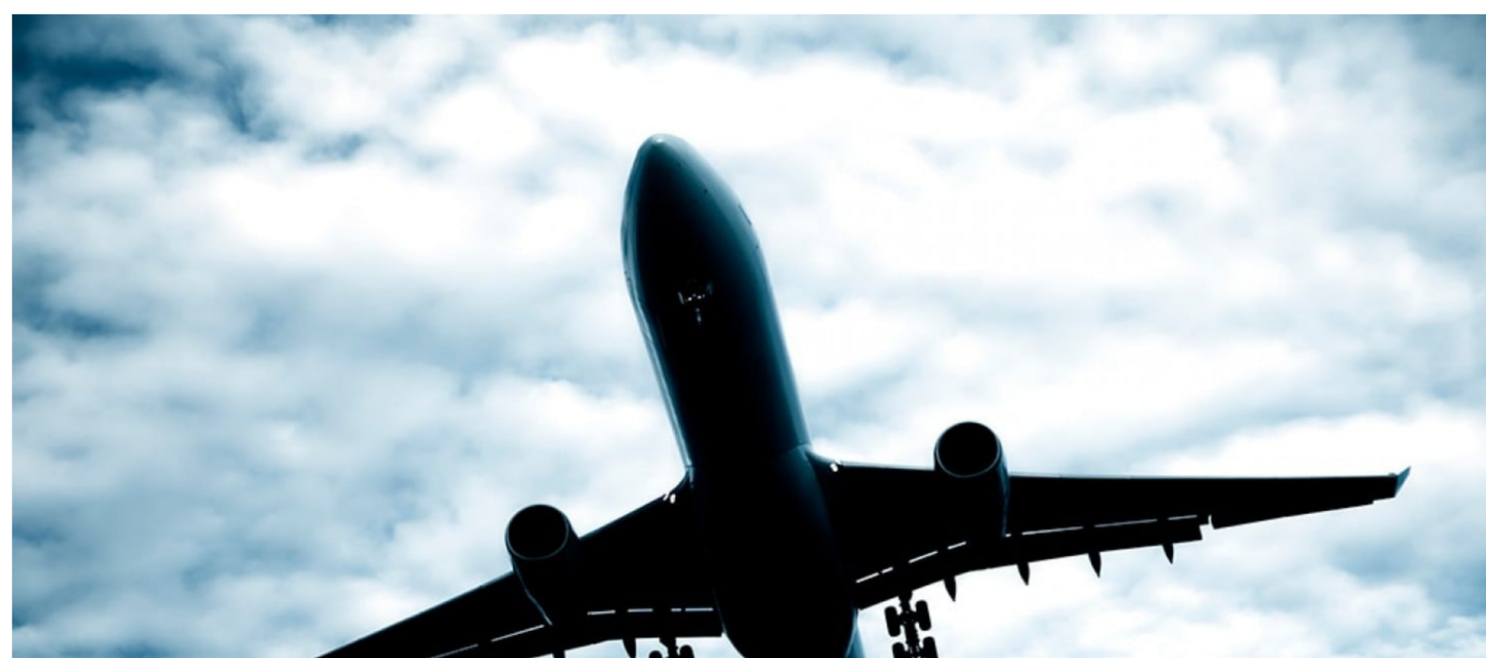
Man claims entertainment system helped him hac

Morning Mix

Hacker Chris Roberts told FBI he took control of United plane, FBI claims

By **Justin Wm. Moyer**

May 18, 2015



Eddie Kohler
@xexd

Following

I spent many years after Asbestos/HiStar down on information flow, because it makes things too hard to program for too little gain. Still think that! But this keeps happening.

noreply@hotcrp.com 2:35 AM (6 hours ago)
to me

2018/08/08 06:30:07 h.asplos19: bad doc 403 Forbidden You aren't allowed to view submission #500. [] @/asplos19-paper500.pdf xxx@stanford.edu
2018/08/08 06:30:13 h.asplos19: bad doc 403 Forbidden You aren't allowed to view submission #600. [] @/asplos19-paper600.pdf xxx@stanford.edu
2018/08/08 06:30:18 h.asplos19: bad doc 403 Forbidden You aren't allowed to view submission #1000. [] @/asplos19-paper1000.pdf xxx@stanford.edu
2018/08/08 06:30:24 h.asplos19: bad doc 403 Forbidden You aren't allowed to view submission #10000. [] @/asplos19-paper10000.pdf xxx@stanford.edu

5:43 AM - 8 Aug 2018

Goal: eliminate covert channels from systems

- **Covert channel** *(Lampson '73)*: *unintended* flow between system components
- **Approach**: verification-driven development
    - Verify noninterference for interface specification
    - Verify refinement for implementation
- **Limitations**: no physical channels; no concurrency

Contributions

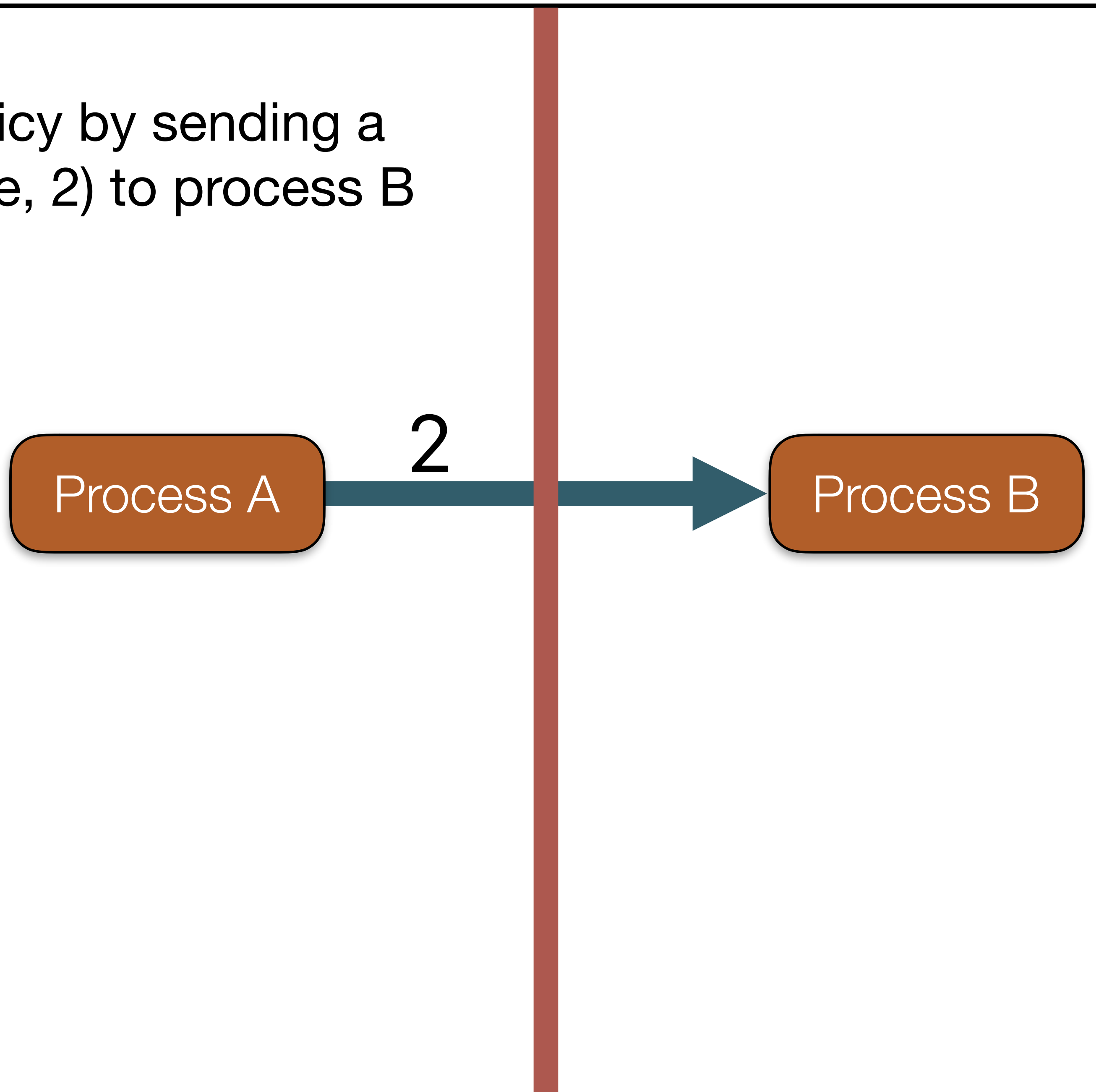
- Formulation of noninterference amenable to automated verification
- Nickel is a framework for verifying IFC systems.
- Applied Nickel to verify systems including
    - NiStar: first formally verified DIFC OS kernel
    - ARINC 653 communication interface: avionics kernel standard

Example covert channel: resource names

Policy: process A and process B should not communicate

Interface: `spawn` system call returns sequential PIDs

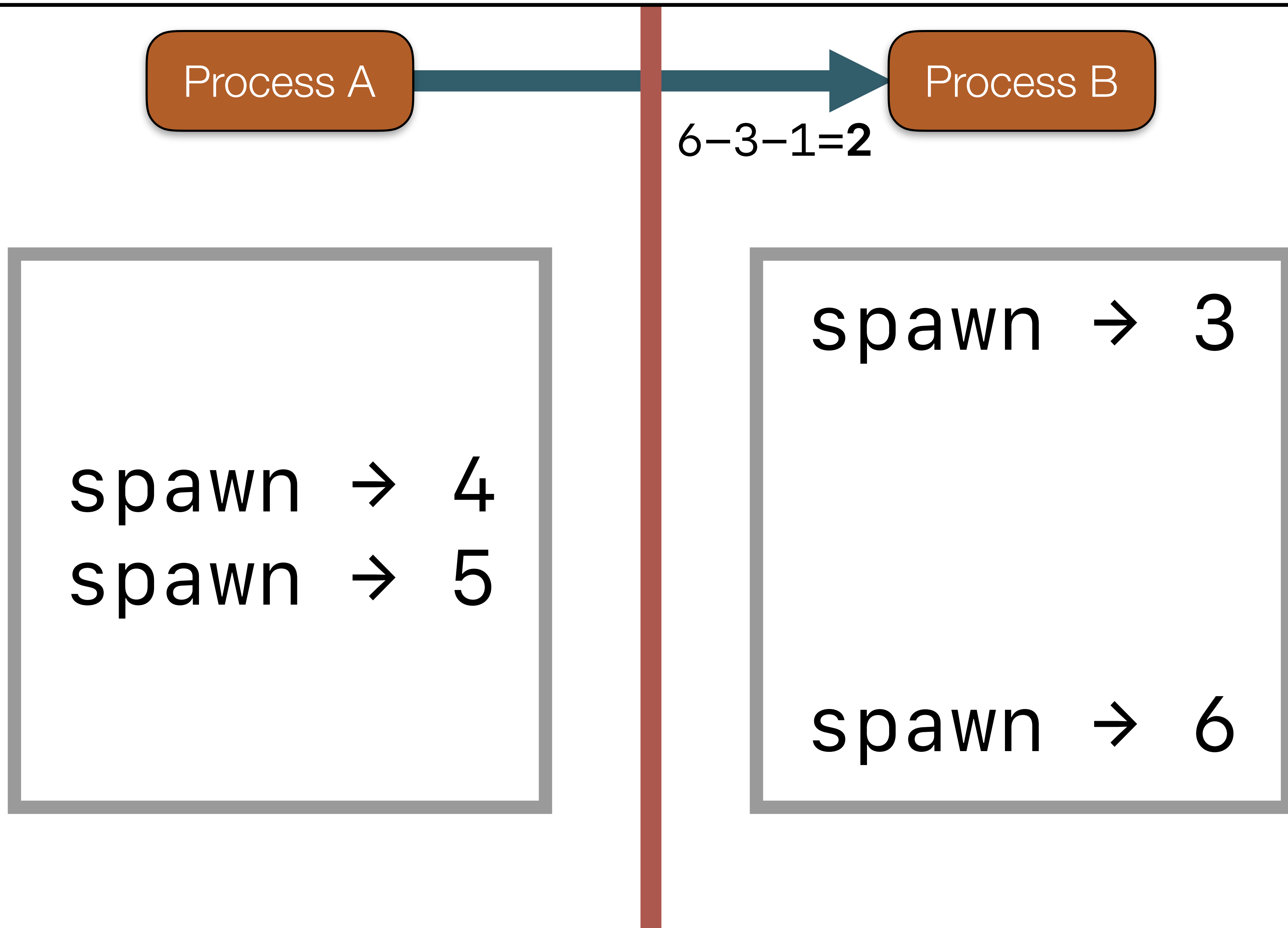
Try to violate policy by sending a secret (in this case, 2) to process B



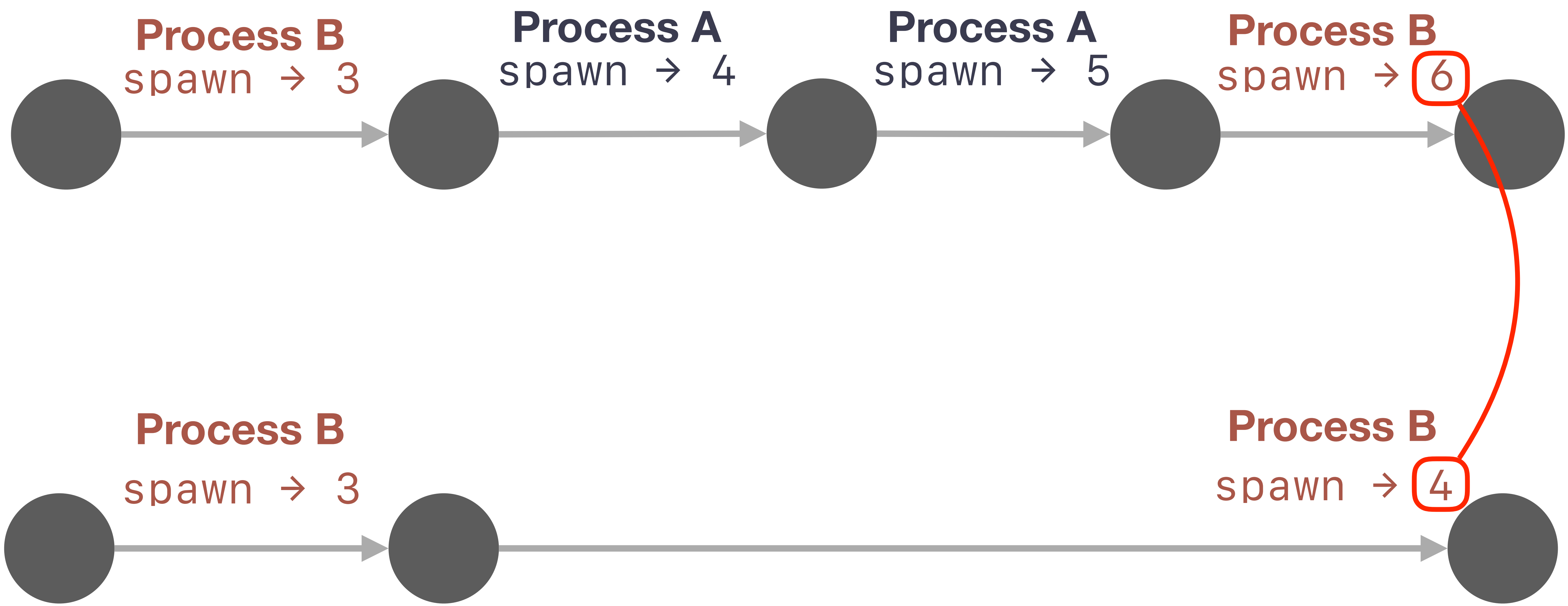
Example covert channel: resource names

Policy: process A and process B should not communicate

Interface: `spawn` system call returns sequential PIDs



Noninterference intuition



Noninterference intuition

Many kinds of covert channels

- Resource names and exhaustion
- Statistical information
- Error handling
- Scheduling
- Devices and services

Noninterference

For any trace *tr*, action *a*, removing "irrelevant" actions should not affect the output of *a*.

$$\text{output}(\text{run}(\text{init}, tr), a) =$$
$$\text{output}(\text{run}(\text{init},\ \text{purge}^*(tr, a)), a)$$

Information flow policies in Nickel

A set of domains

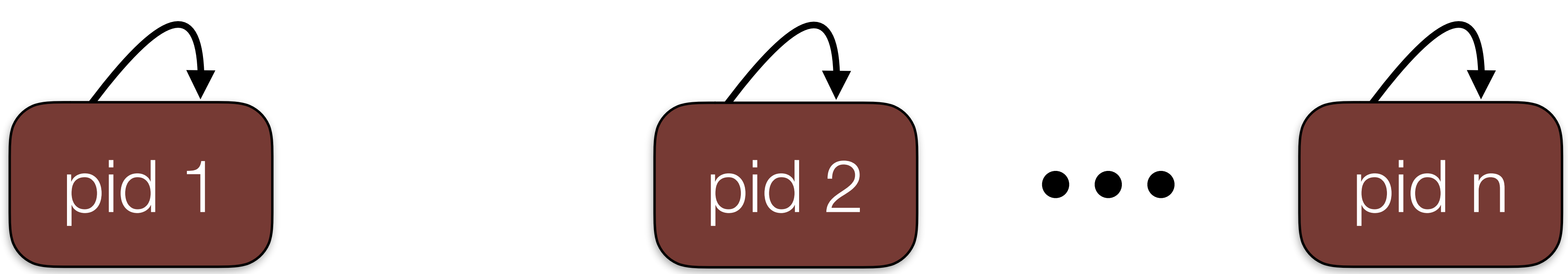
$D : \mathrm{Set}$

A can-flow-to relation specifying permitted flows among domains

$\rightsquigarrow \subseteq (D \times D)$

A function mapping an *action* in a state to a domain

$\mathrm{dom} : (A \times S) \rightarrow D$



Automated verification of noninterference

Proof strategy: unwinding conditions

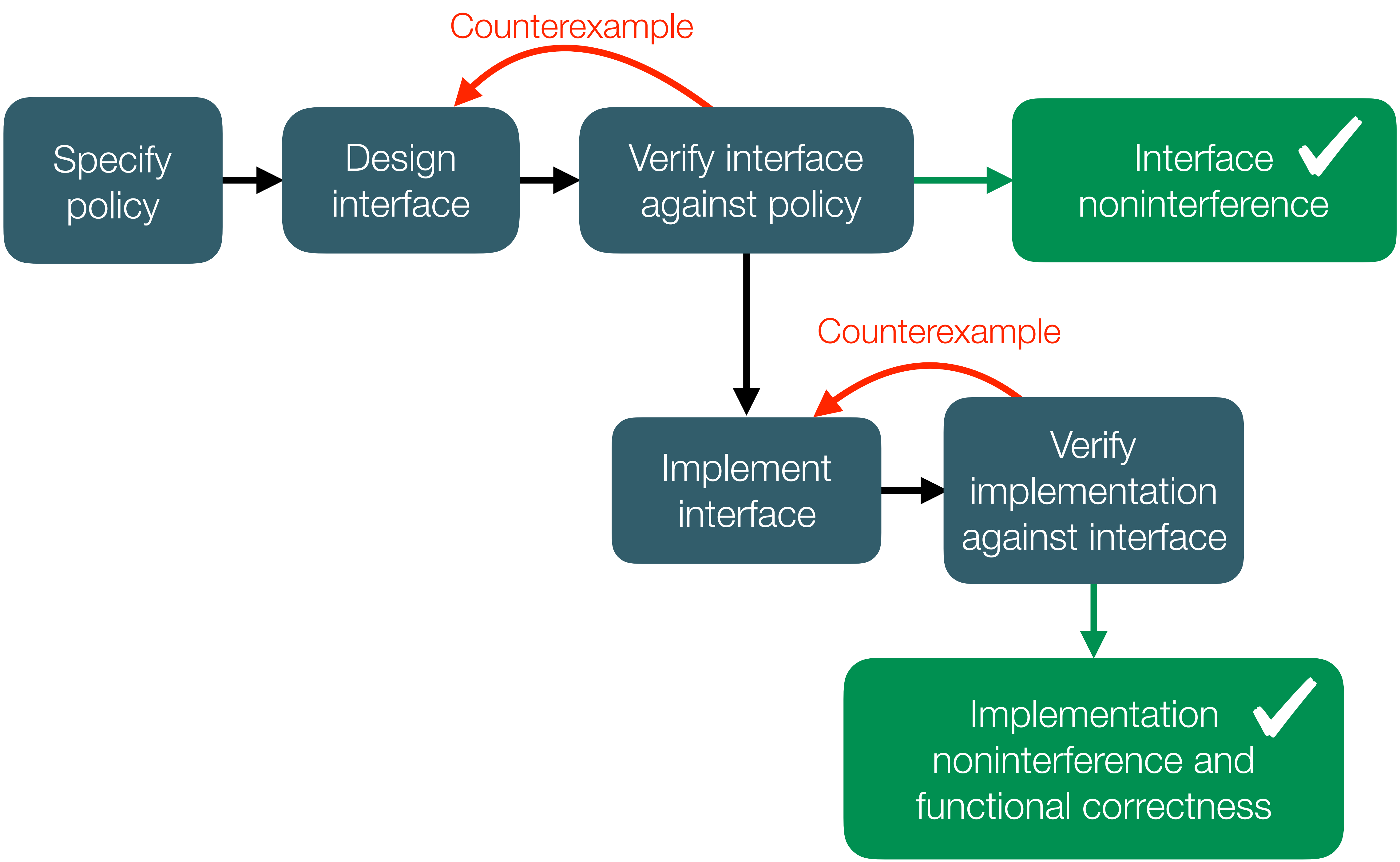
- Together imply noninterference
- Reason about one action at a time
- Amenable to SMT solving using Z3

Local respect $I(s) \land \neg(\mathrm{dom}(a, s) \rightsquigarrow v) \rightarrow s \overset{v}{\approx} \mathrm{step(s, a)}$

Output consistency $I(s) \land I(t) \land s \overset{\mathrm{dom}(a,s)}{\approx} t \rightarrow \mathrm{output}(s, a) = \mathrm{output}(t, a)$

Weak step consistency $I(s) \land I(t) \land s \overset{u}{\approx} t \ \land \ s \overset{\mathrm{dom}(a,s)}{\approx} t \rightarrow \mathrm{step(s, a)} \overset{u}{\approx} \mathrm{step}(t, a)$

Nickel workflow



Programmer inputs

Information flow policy

Interface specification

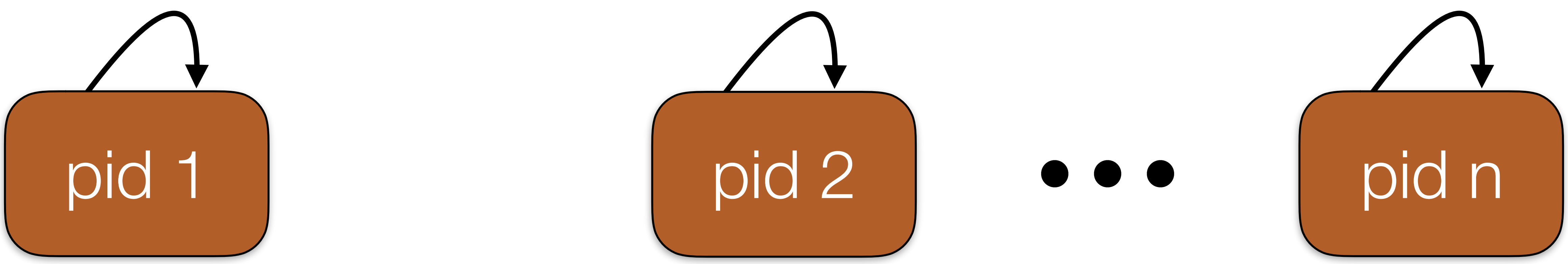
Observational equivalence

Information flow policy

Interface specification

Observational equivalence

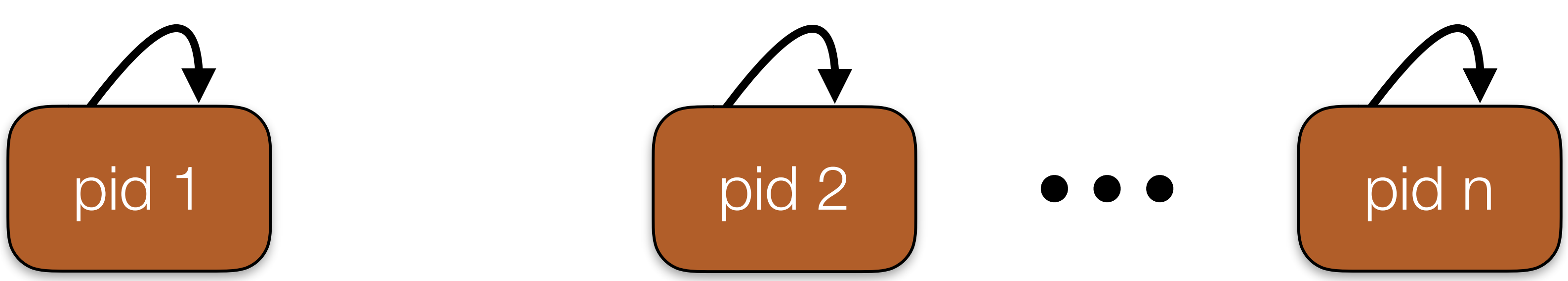
n processes that are not allowed to communicate



Information flow policy | Interface specification | Observational equivalence

n processes that are not allowed to communicate

```
class State:
   current     = PidT()
   nr_procs    = SizeT()
   proc_status = Map(PidT, StatusT)

def can_flow_to(domain1, domain2):
   # Flow only permitted if same domain
   return domain1 == domain2

def dom(action, state):
   # Domain of each action is current process
   return state.current
```



Information flow policy | **Interface specification** | Observational equivalence

Compute child pid

Precondition for system call

Update system state

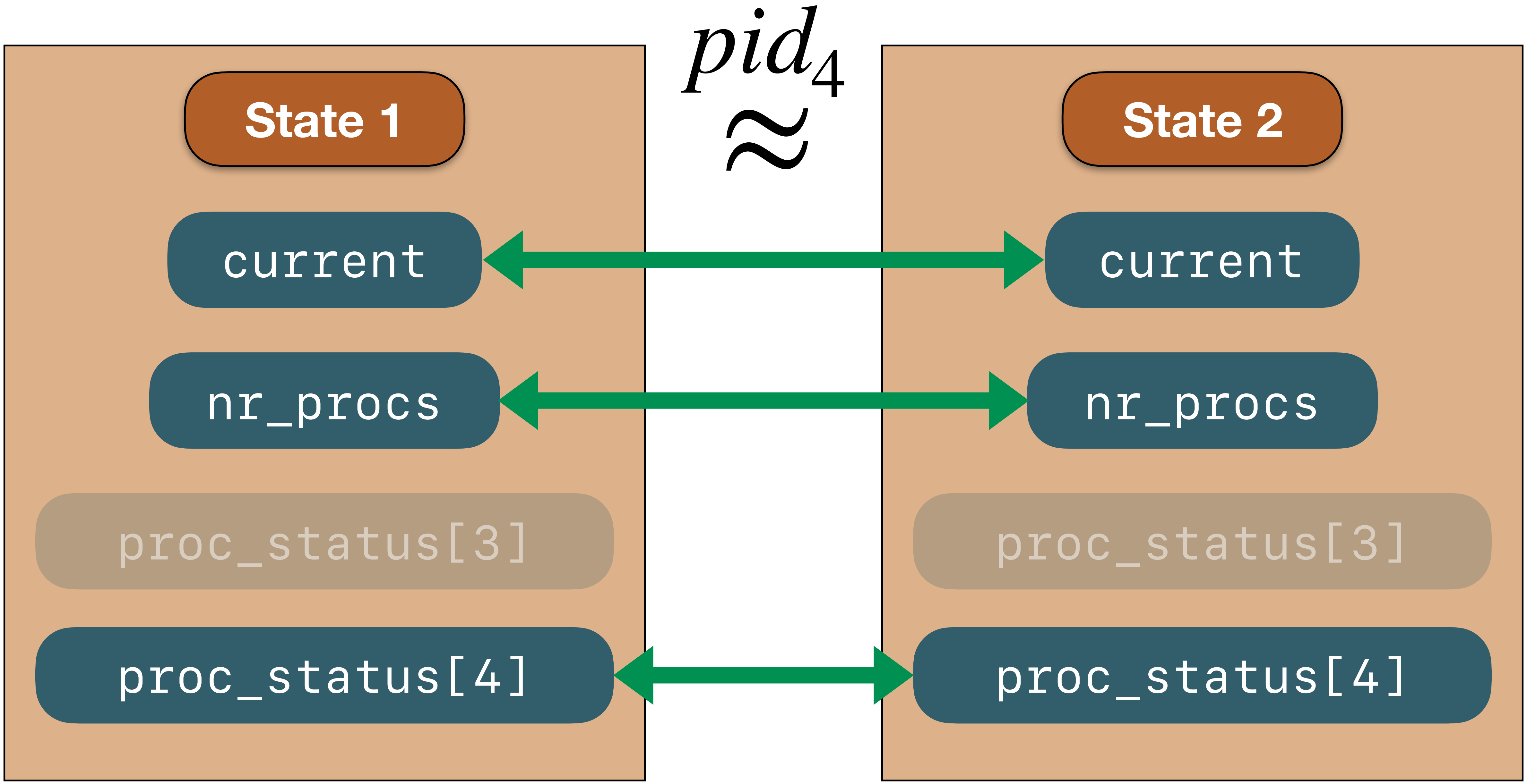
Return new state

```
def sys_spawn(old):
    child_pid = old.nr_procs + 1

    pre = child_pid <= NR_PROCS

    new = old.copy()
    new.nr_procs += 1
    new.proc_status[child_pid] = RUNNABLE

    return pre, If(pre, new, old)
```

Information flow policy

Interface specification

Observational equivalence

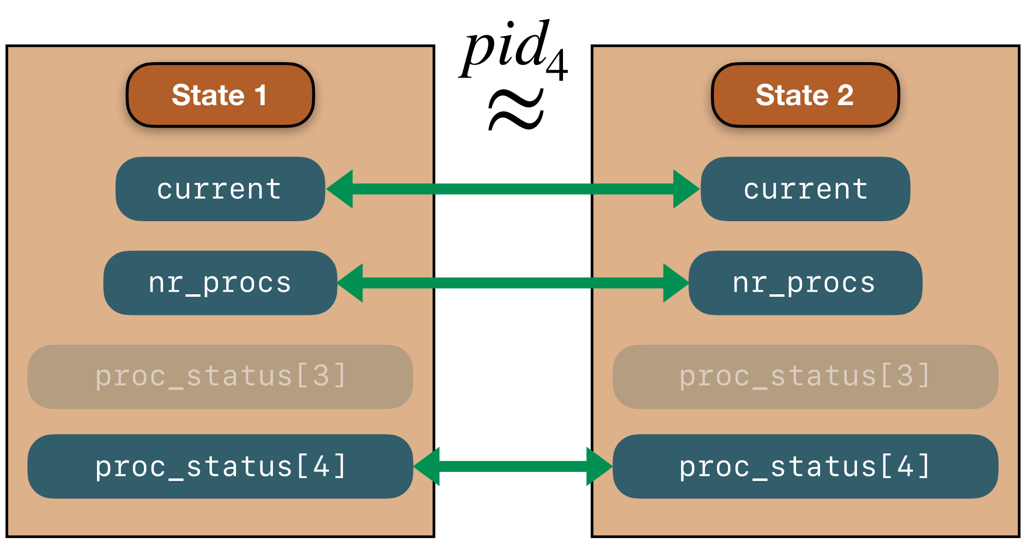


Information flow policy

Interface specification

Observational equivalence

```
class State:
  current     = PidT()
  nr_procs    = SizeT()
  proc_status = Map(PidT, StatusT)

def obs_eqv(domain, state1, state2):
  return And(
    state1.current == state2.current,
    state1.nr_procs == state2.nr_procs,
    state1.proc_status[domain.pid] ==
        state2.proc_status[domain.pid]
    )
```

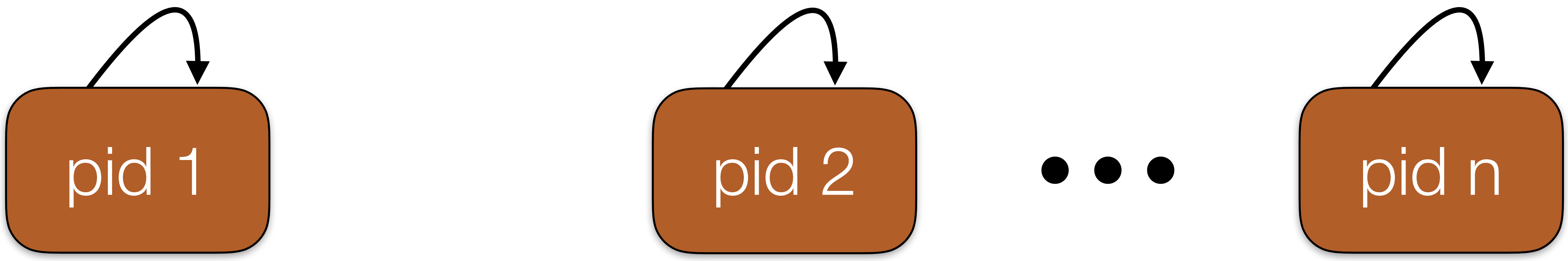


Systems verified using Nickel

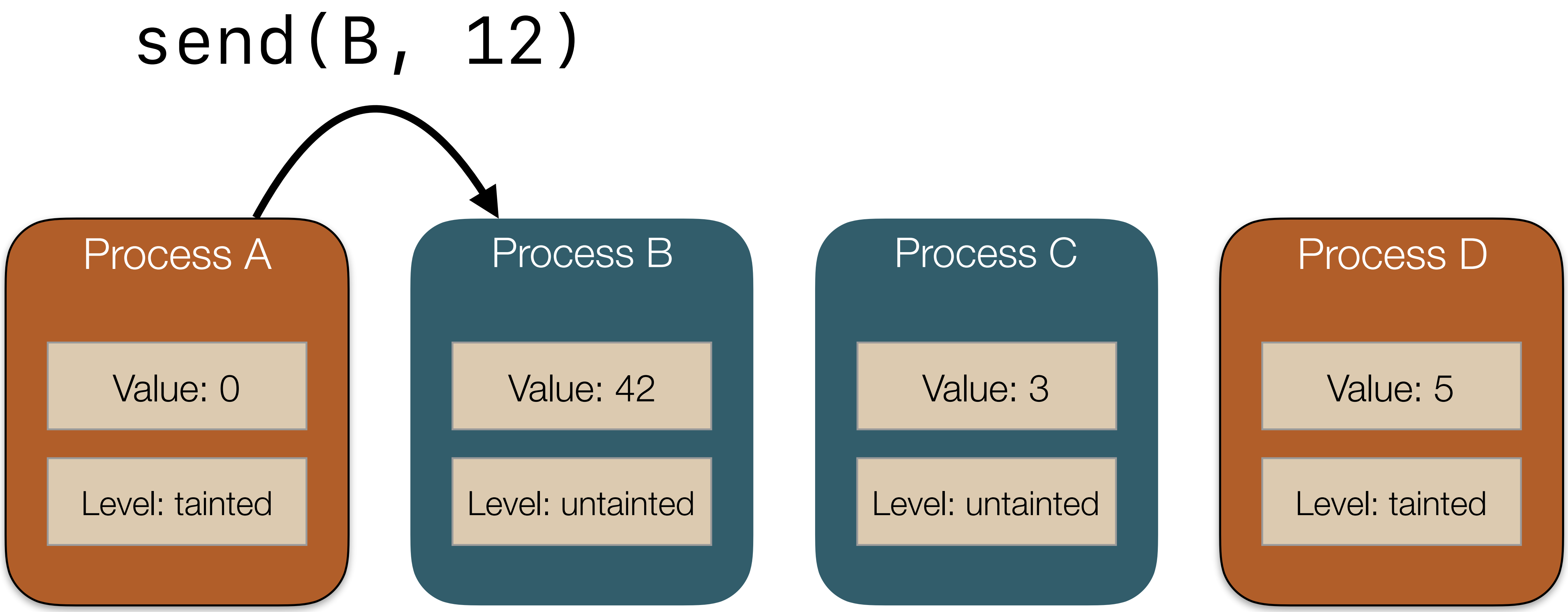
Component	NiStar	NiKOS	ARINC 653
Information flow policy	26	14	33
Interface specification	714	82	240
Observational equivalence	127	56	80
Implementation	3,155	343	—
User-space implementation	9,348	389	—
Common kernel infrastructure	4,829 (shared by NiStar/NiKOS)		—

Demo

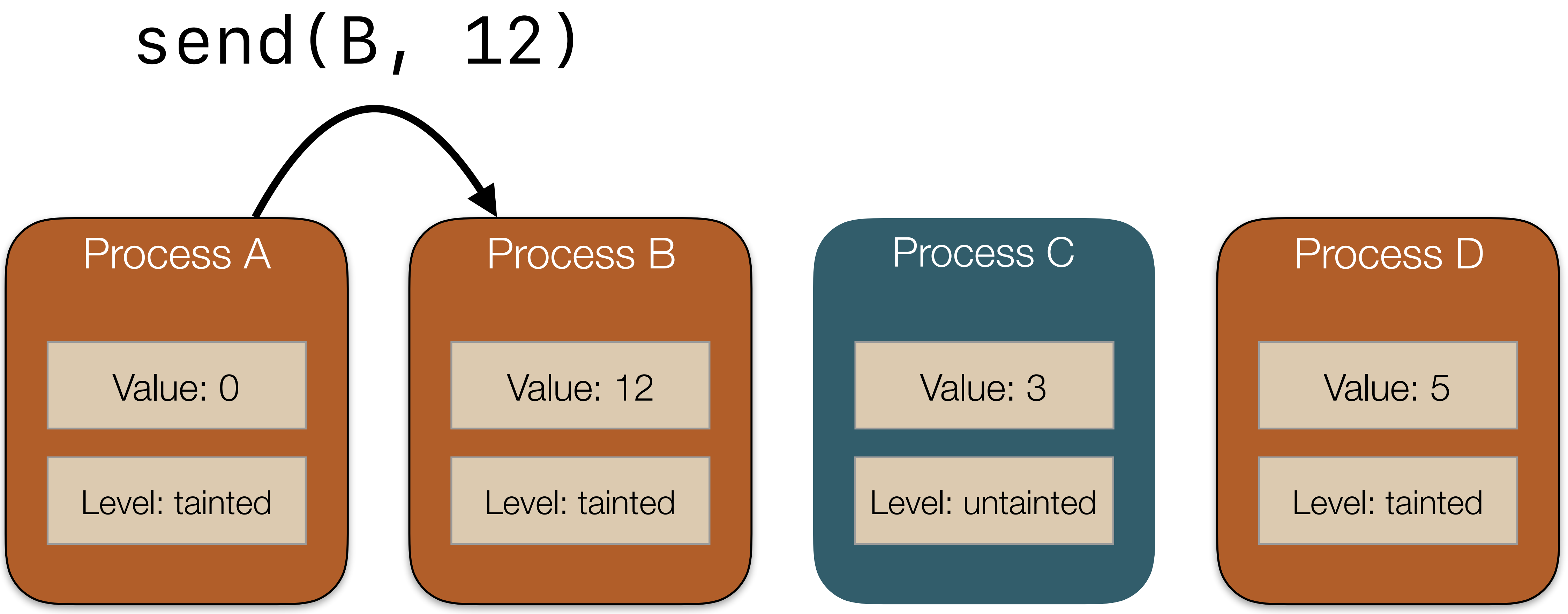
`spawn` example



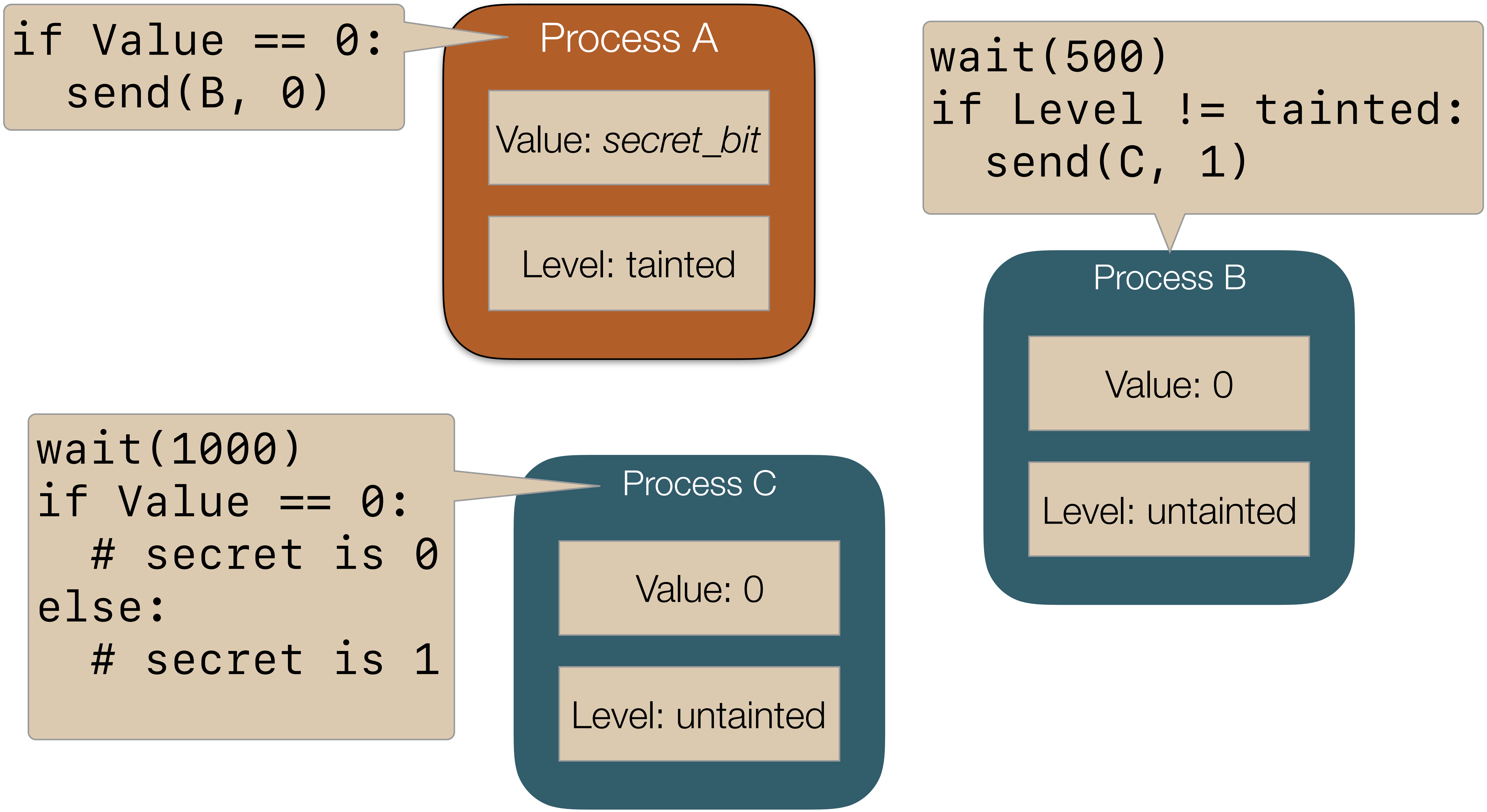
`tainting` example



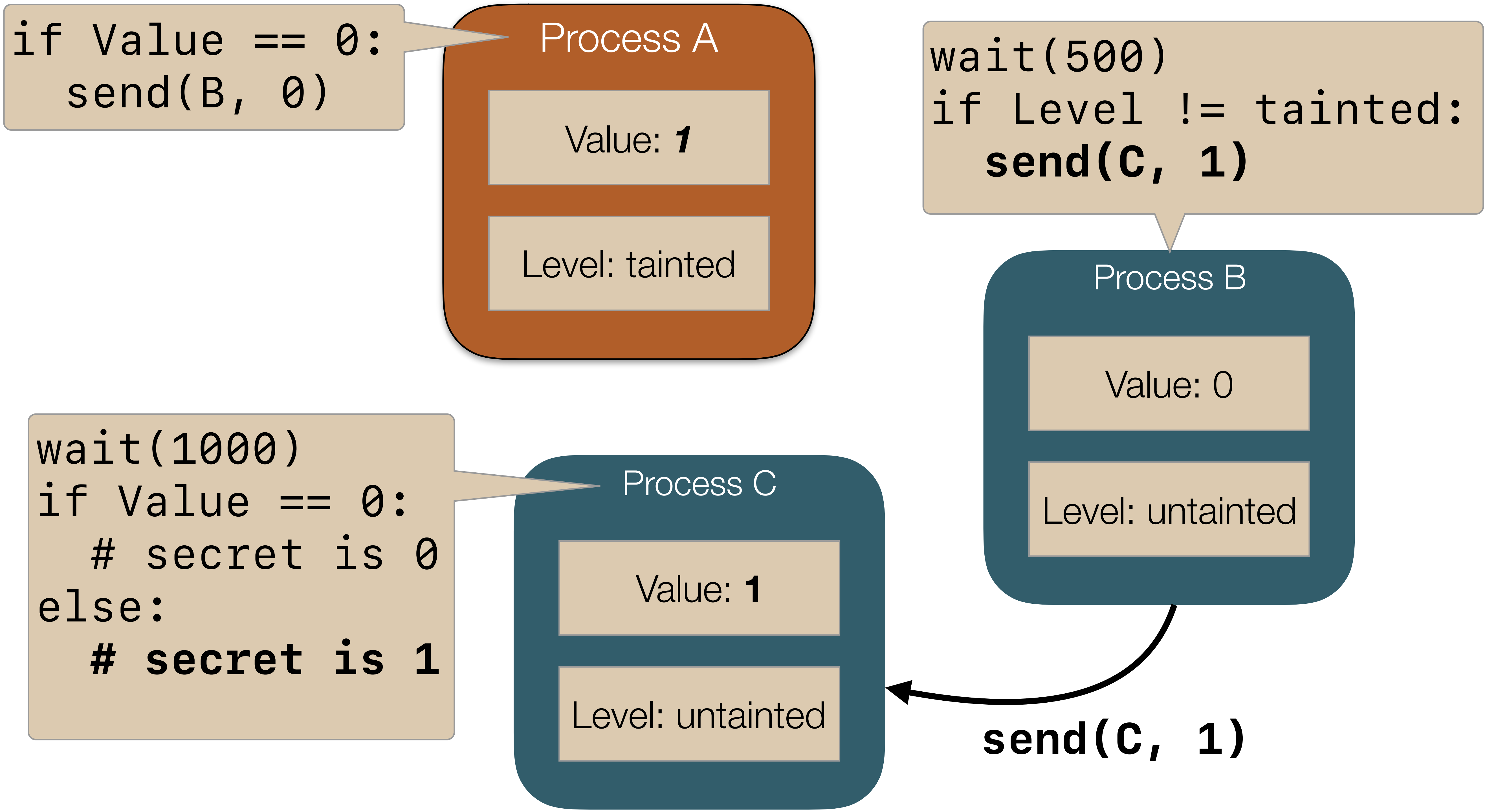
`tainting` example



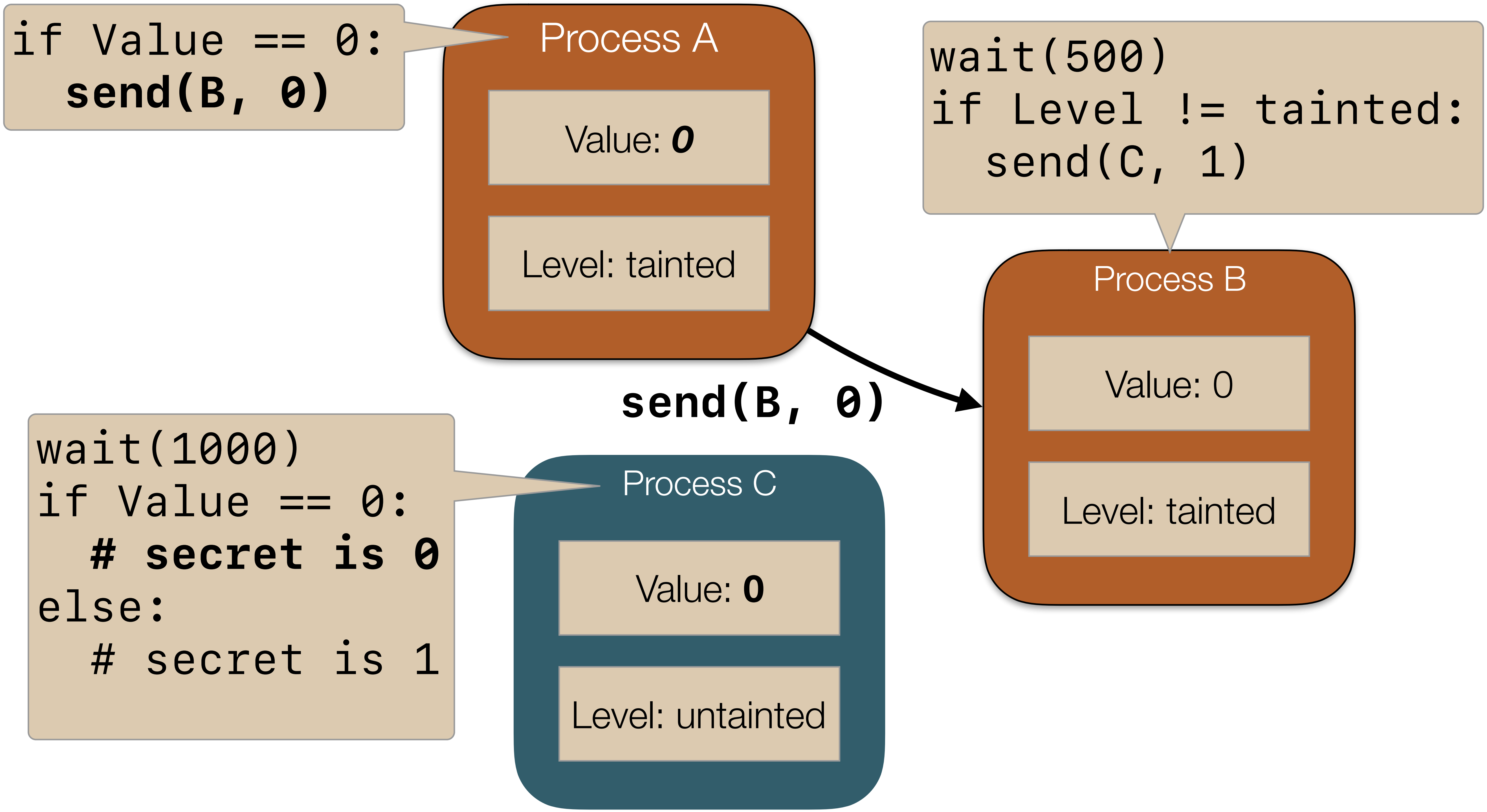
`tainting` example



tainting example



tainting example



Thanks!

UNSAT

https://nickel.unsat.systems