Analysis and Application of Depth-Bounded Systems

Damien Zufferey

IST Austria

EPFL - LARA, 12.09.2012

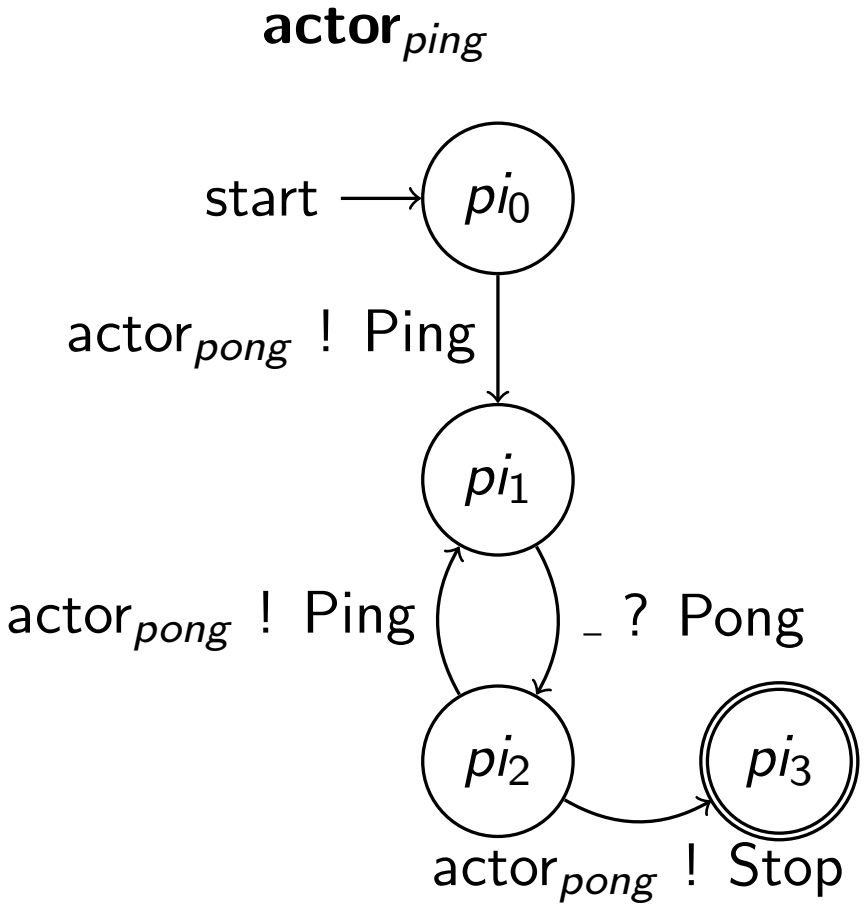
Outline

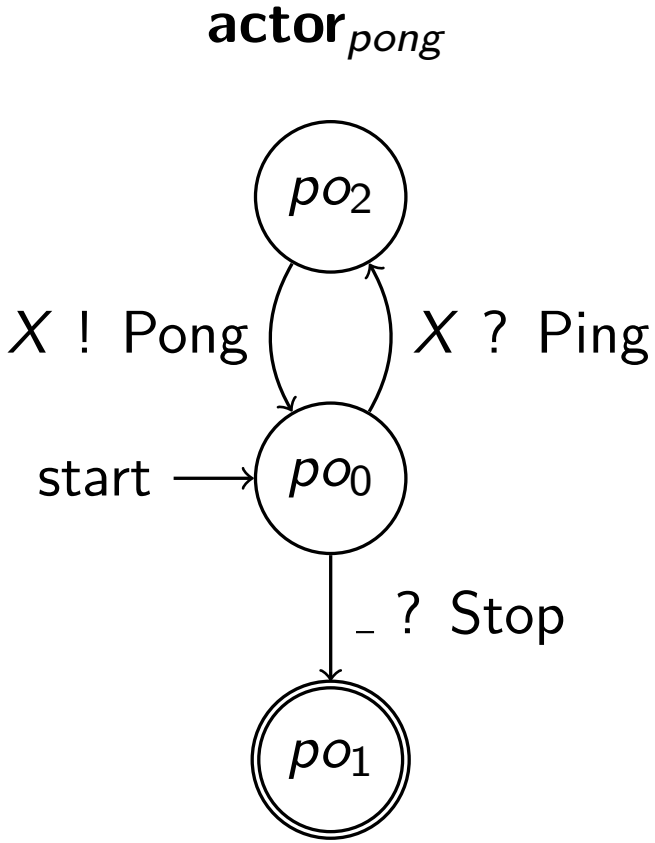
Example (1): scala/docs/examples/actors/pingpong.scala

```
class Ping(count: Int, pong: Actor) extends Actor {
  def act() {
    var pingsLeft = count - 1
    pong ! Ping
    loop {
      react {
        case Pong =>
          if (pingsLeft % 1000 == 0)
            println("Ping: pong")
          if (pingsLeft > 0) {
            pong ! Ping
            pingsLeft -= 1
          } else {
            println("Ping: stop")
            pong ! Stop
            exit()
          }
      }
    }
  }
}
```

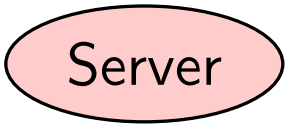
```
class Pong extends Actor {
  def act() {
    var pongCount = 0
    loop {
      react {
        case Ping =>
          if (pongCount % 1000 == 0)
            println("Pong: ping "+pongCount)
          sender ! Pong
          pongCount += 1
        case Stop =>
          println("Pong: stop")
          exit()
      }
    }
  }
}
```

Example (2): scala/docs/examples/actors/pingpong.scala

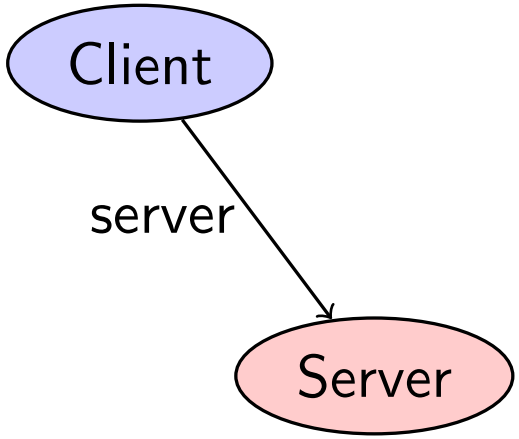




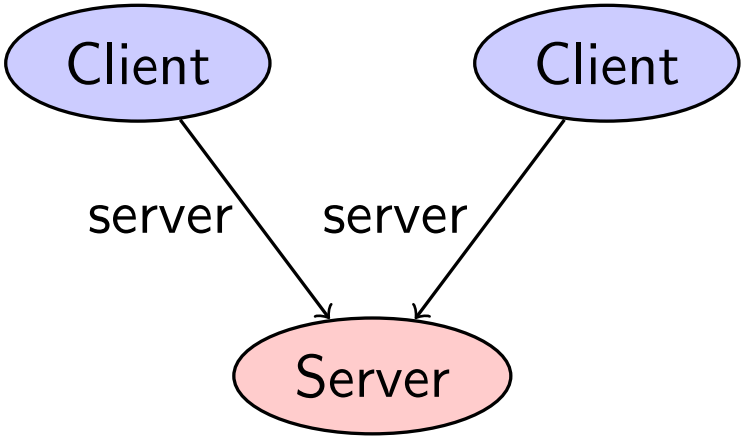
Example: client-server communication pattern



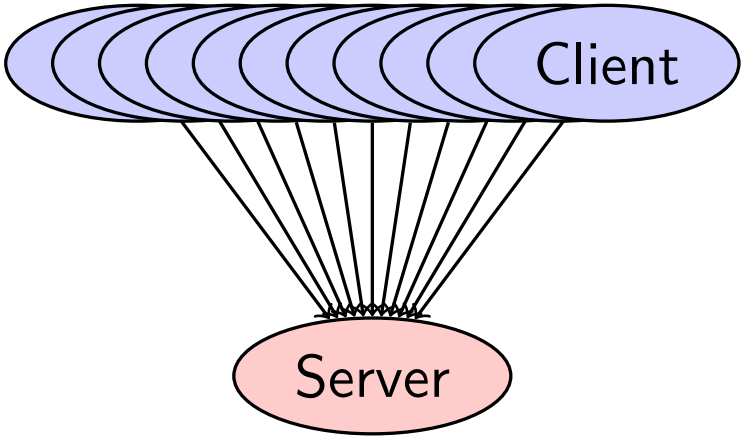
Example: client-server communication pattern



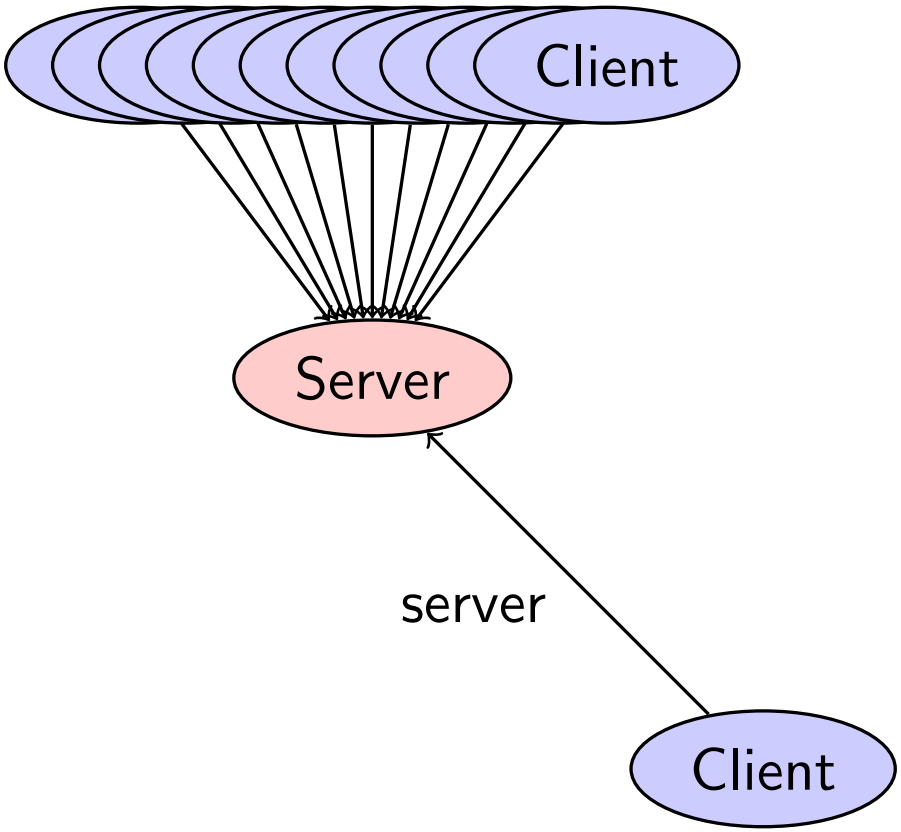
Example: client-server communication pattern



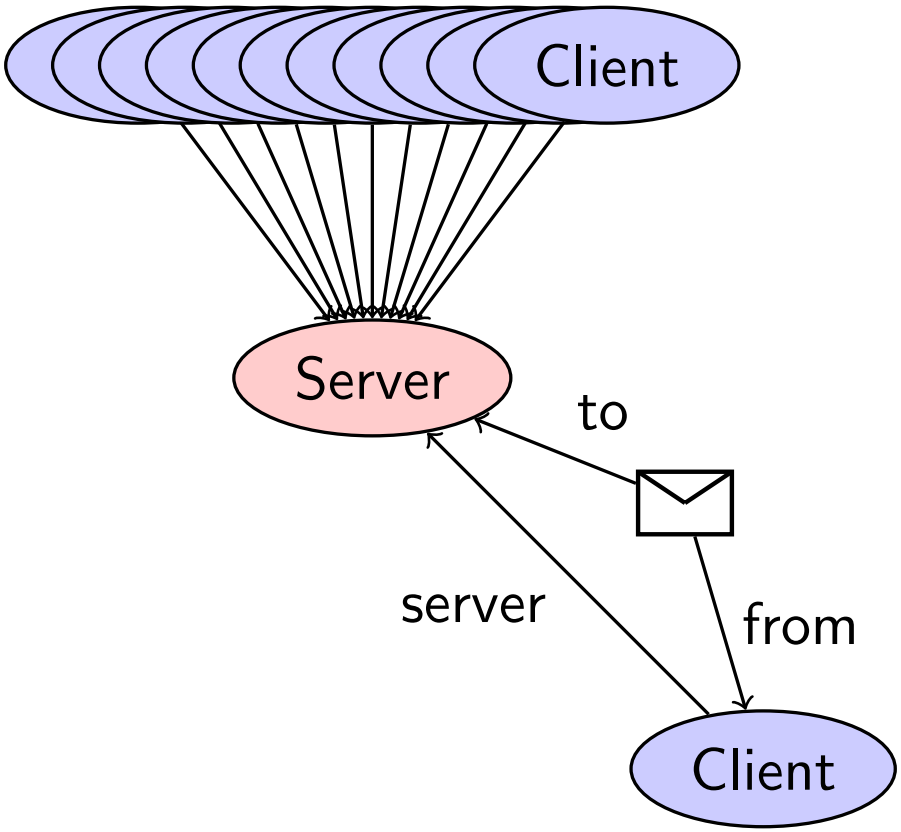
Example: client-server communication pattern



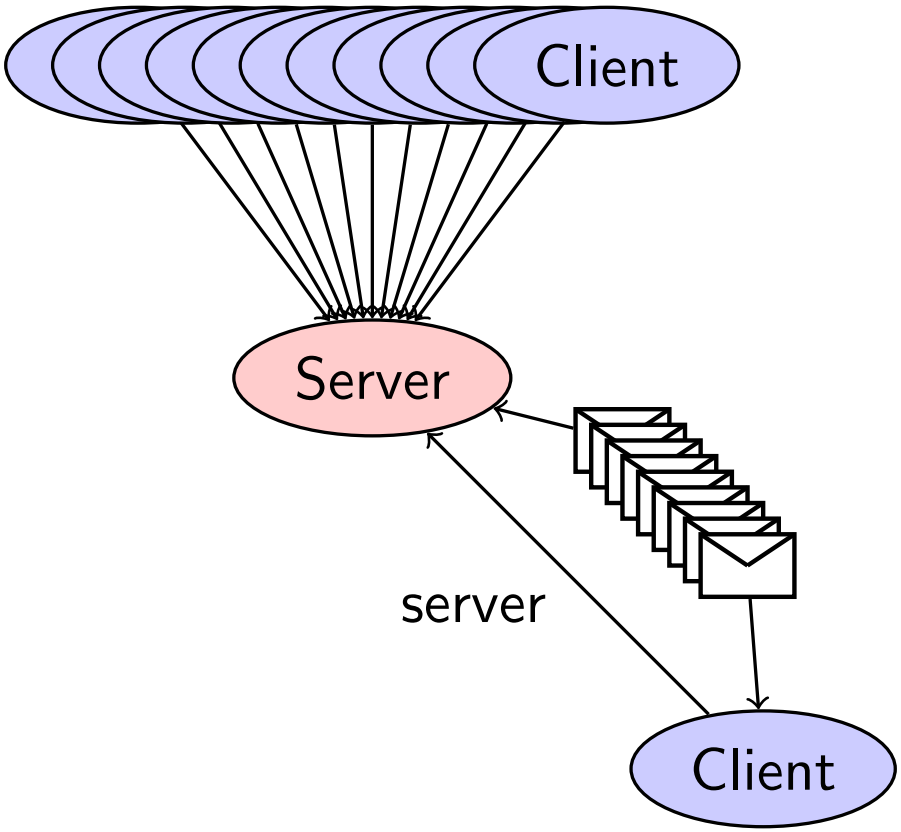
Example: client-server communication pattern



Example: client-server communication pattern

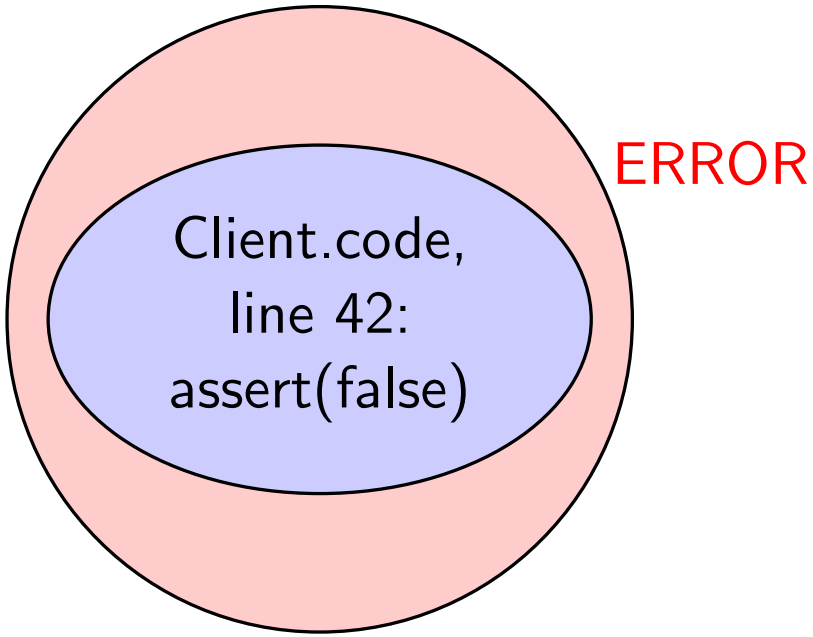


Example: client-server communication pattern



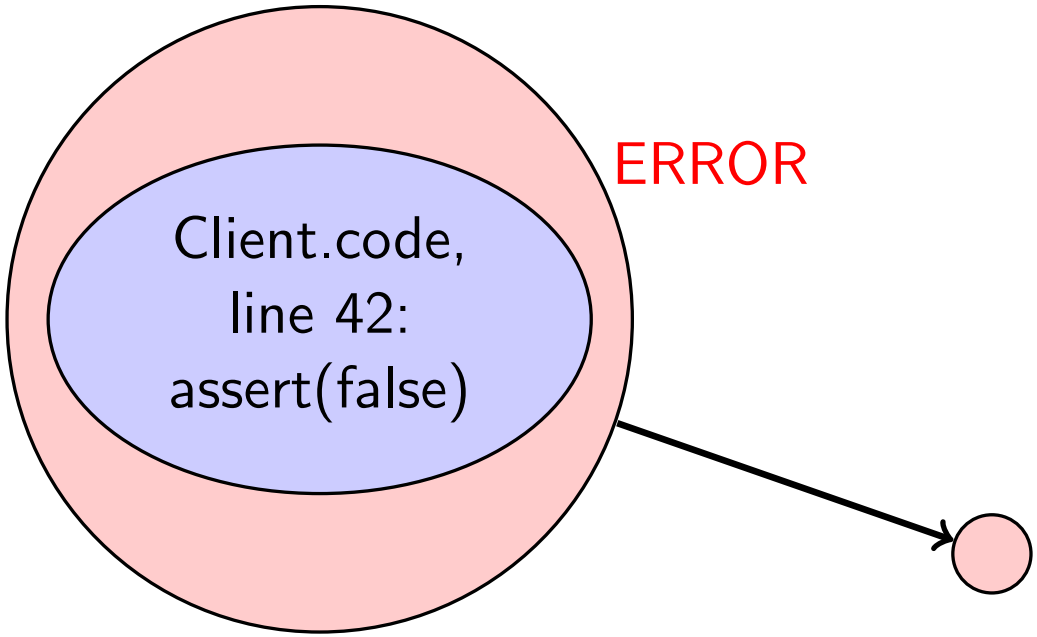
What kind of properties are we looking at ?

Safety properties, more precisely the control-state reachability problem (aka covering problem).



What kind of properties are we looking at ?

Safety properties, more precisely the control-state reachability problem (aka covering problem).



What kind of properties are we looking at ?

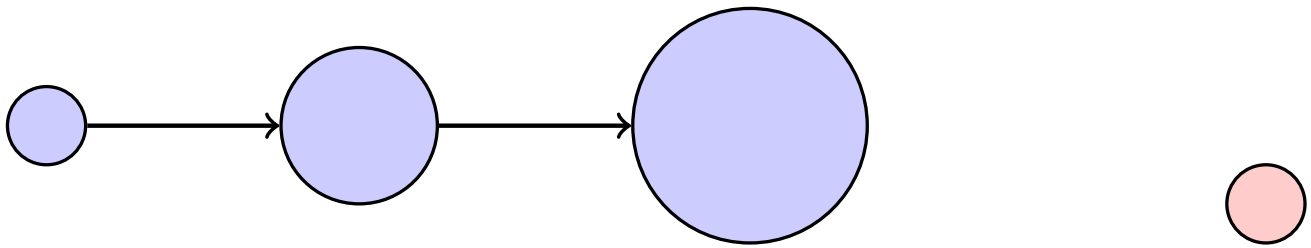
Safety properties, more precisely the control-state reachability problem (aka covering problem).

What kind of properties are we looking at ?

Safety properties, more precisely the control-state reachability problem (aka covering problem).

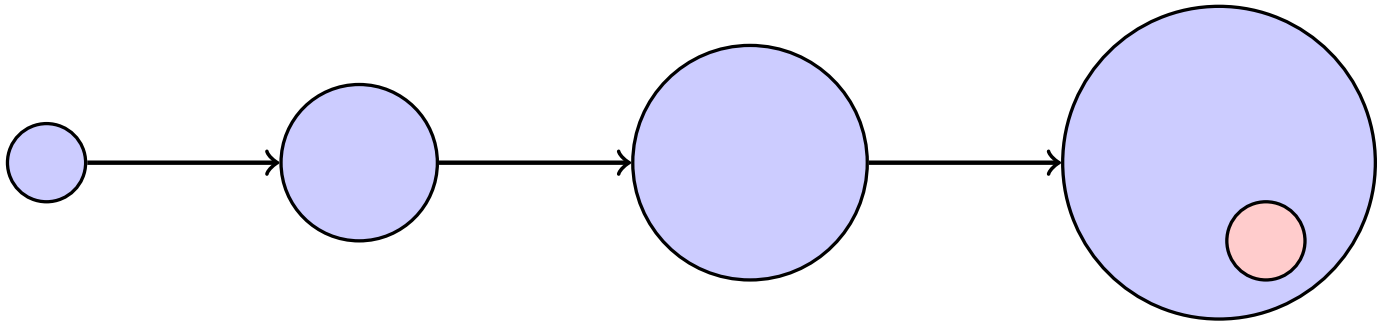
What kind of properties are we looking at ?

Safety properties, more precisely the control-state reachability problem (aka covering problem).



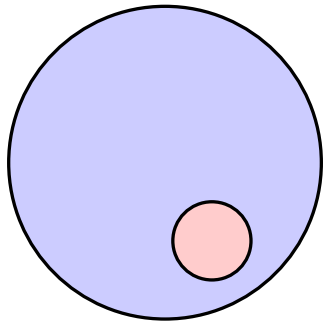
What kind of properties are we looking at ?

Safety properties, more precisely the control-state reachability problem (aka covering problem).



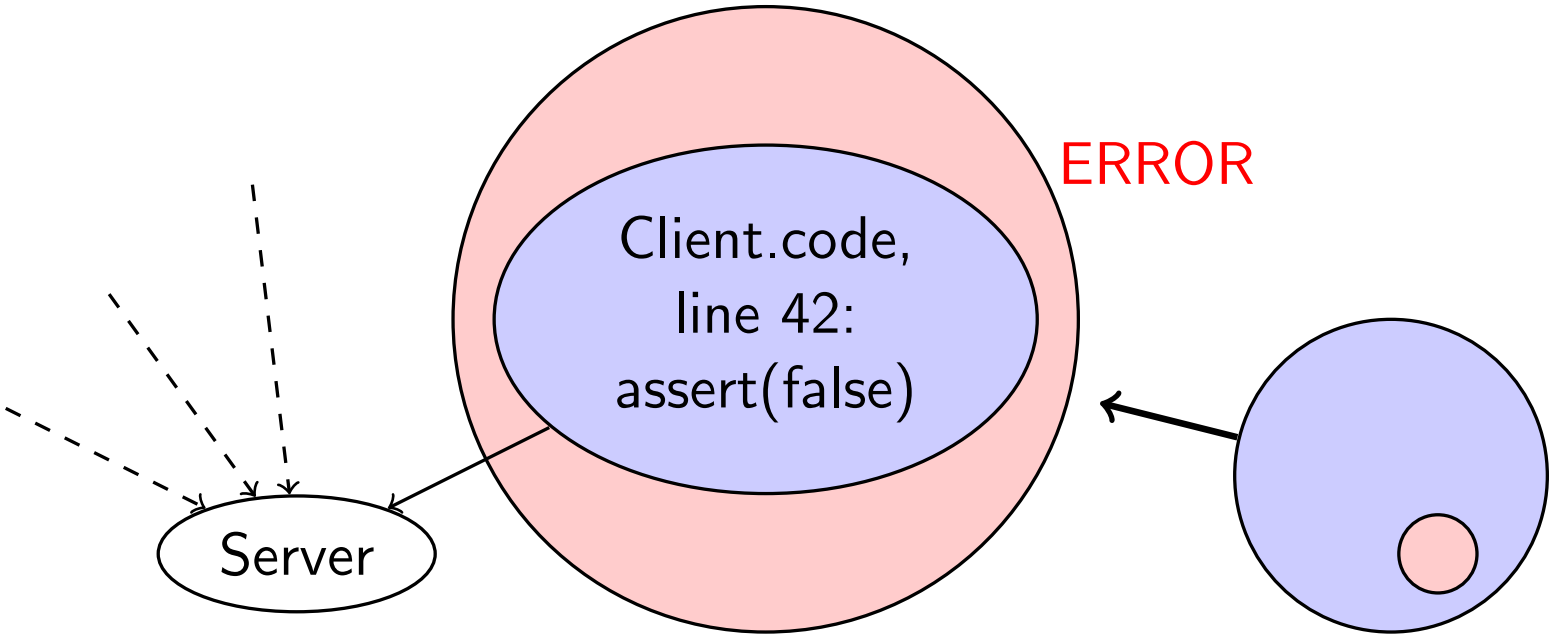
What kind of properties are we looking at ?

Safety properties, more precisely the control-state reachability problem (aka covering problem).



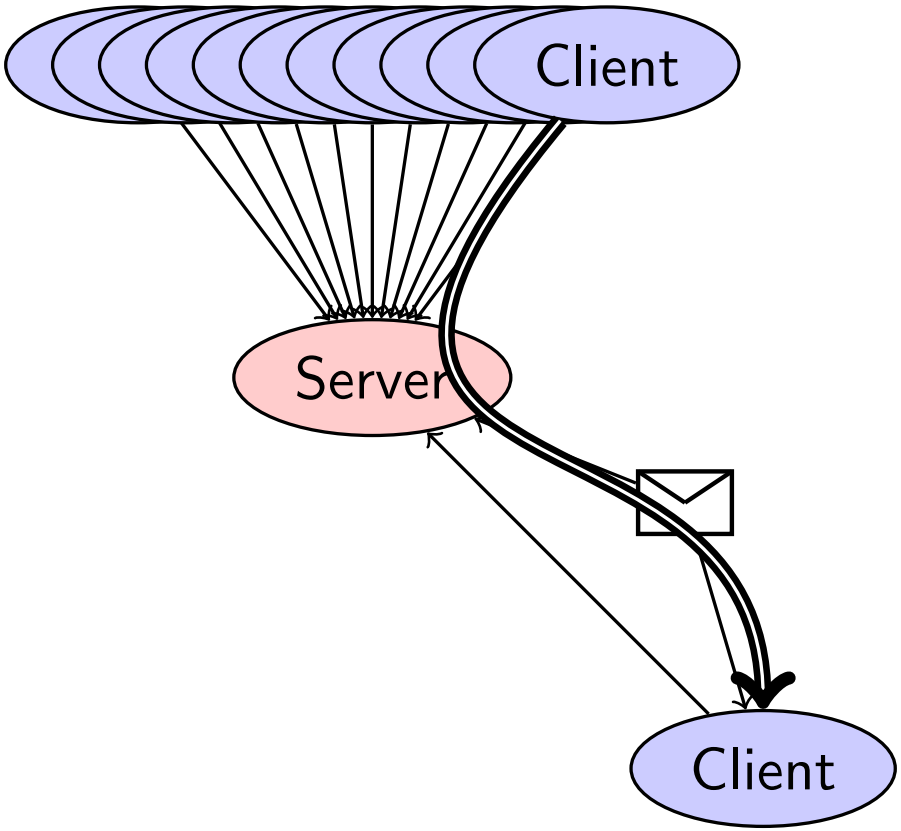
What kind of properties are we looking at ?

Safety properties, more precisely the control-state reachability problem (aka covering problem).



Depth-bounded systems: [Meyer, 2008]

System with a bound on the longest acyclic path.
(Concretely: it is not possible to encode an infinite memory.)



Formal model: WSTS

A well-structured transition system (WSTS) is a transition system $\langle S, \rightarrow, \leq \rangle$ such that:

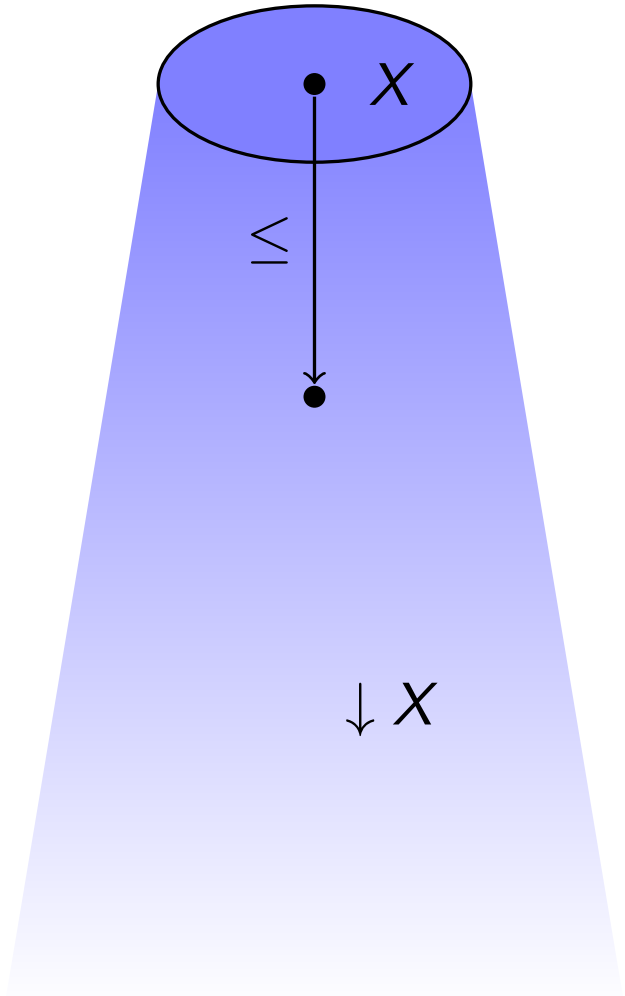
- $\leq$ is a well-quasi-ordering (wqo),
  i.e. well-founded + no infinite antichain.
- compatibility of $\leq$ w.r.t. $\rightarrow$

$$\begin{array}{ccc} & & * \\ & t & \longrightarrow & t' \\ \forall & \vee| & & \vee| & \exists \\ & s & \longrightarrow & s' \end{array}$$

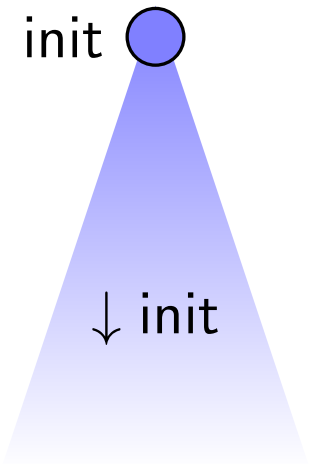
For more detail see:
[Finkel and Schnoebelen, 2001, Abdulla et al., 1996]

Downward Closure

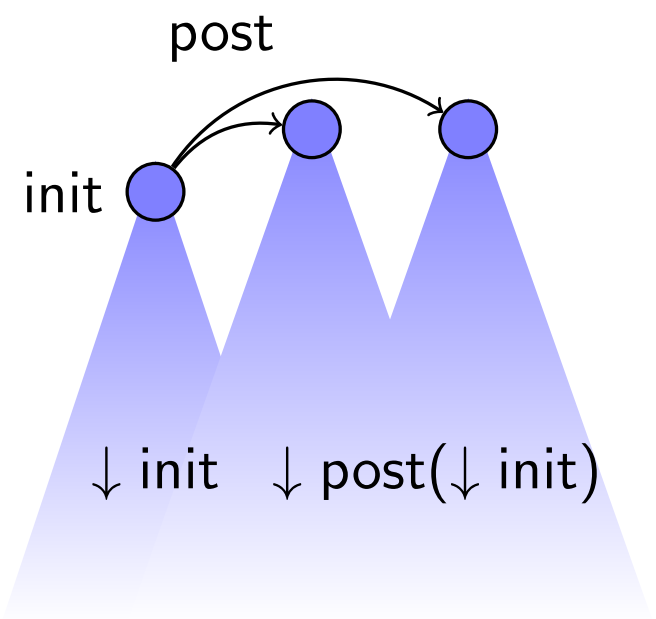
$$\downarrow X = \{\ x' \mid \exists x \in X.\ x' \leq x\ \}$$



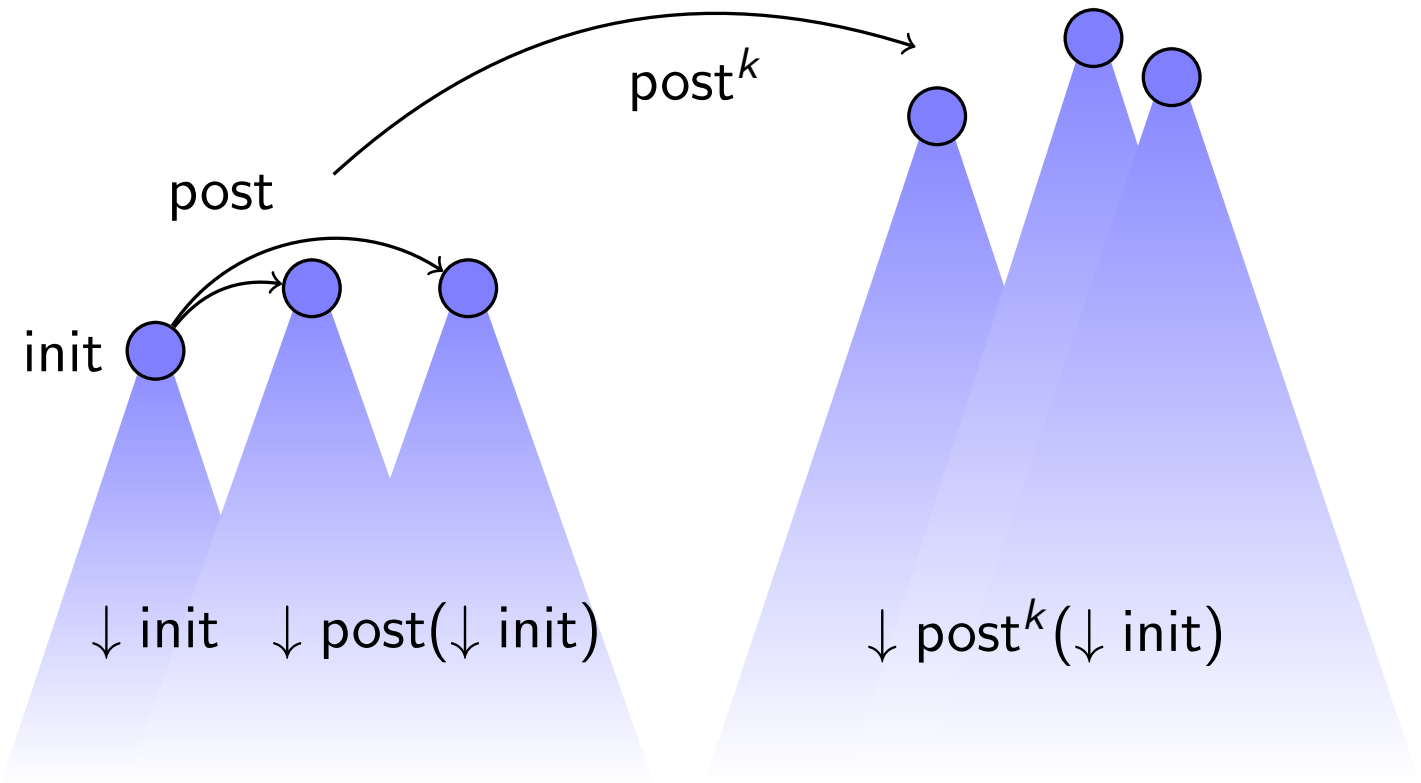
Forward algorithm for covering



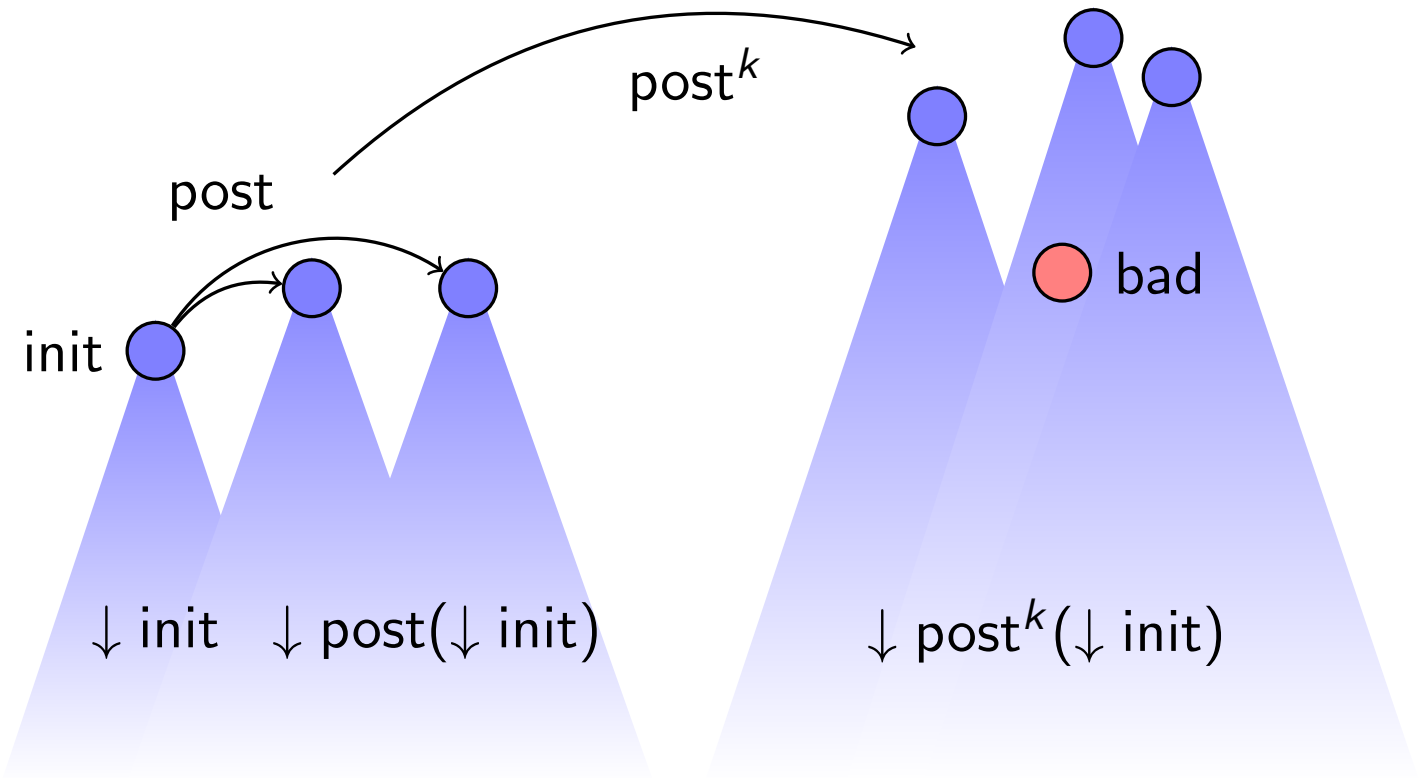
Forward algorithm for covering



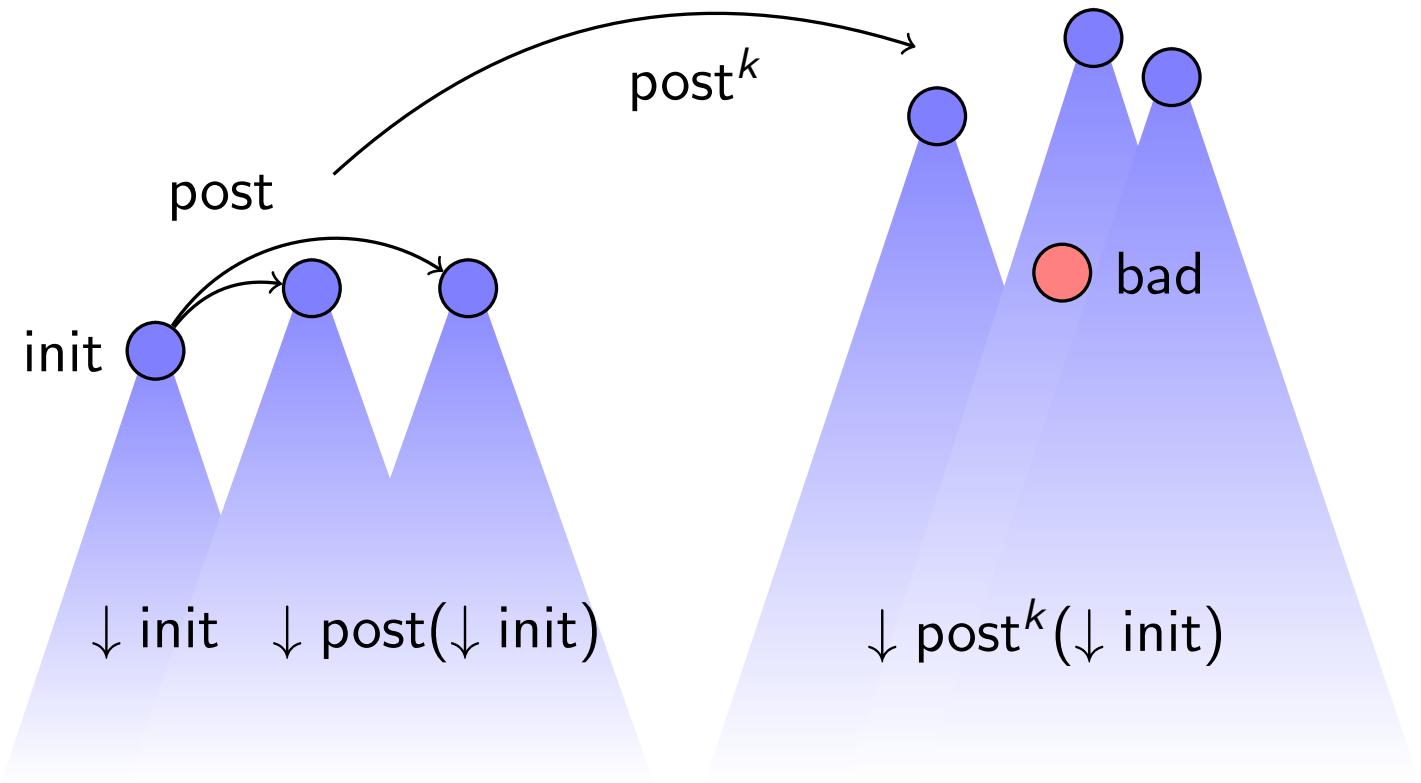
Forward algorithm for covering



Forward algorithm for covering



Forward algorithm for covering



Computes the covering set rather than answering only a single coverability query.

Representing downward-closed sets with ideals

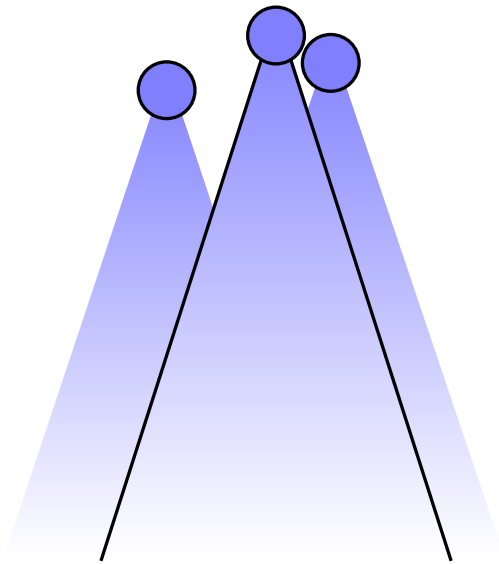
Given a wqo-set $(X, \leq)$.

A subset of X is directed if it is non-empty and closed under upper bounds.

An ideal of X is a directed downward-closed subset of X.

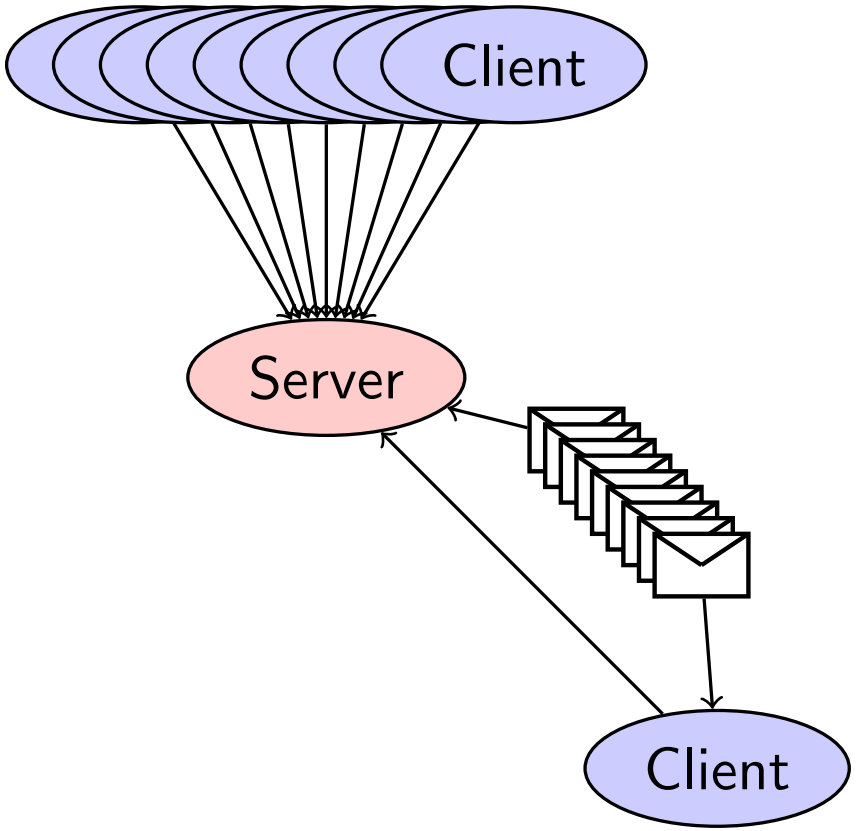
The ideal completion $Idl(X)$ of X is the set of all ideals of X.

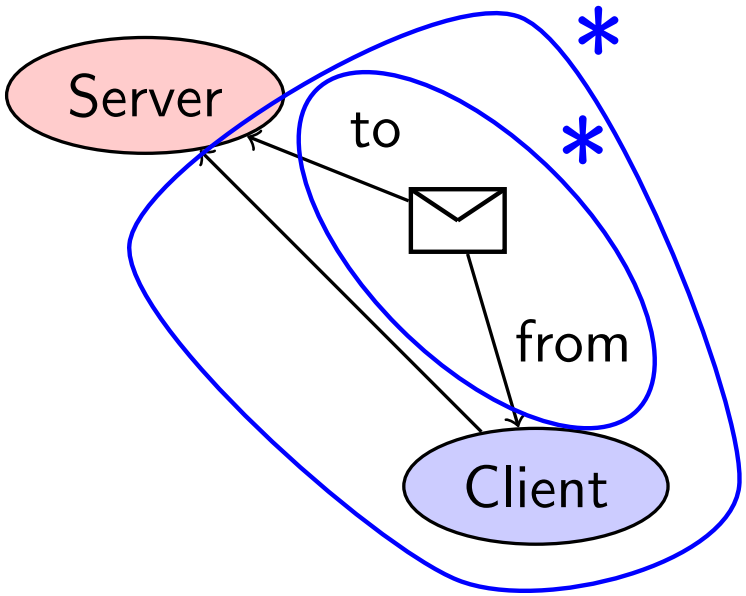
A downward-closed subset is a finite union of ideals.



Ideal for representing downward-closed sets.

ADL: [Geeraerts et al., 2006]
Further developed in [Finkel and Goubault-Larrecq, 2009]
Applied to DBP in [Wies et al., 2010]





When does acceleration work ? (flat systems)

Forward algorithms are (usually) based on acceleration.
Acceleration *executes* loops infinitely many time (saturation).

Concretely, The algorithm terminates if the covering set can be generated by executing only simple loops. This condition is known as flattability [Bardin $\mathrm{et\ al.}$, 2005]. Acceleration *executes* traces of length $< \omega^2$.

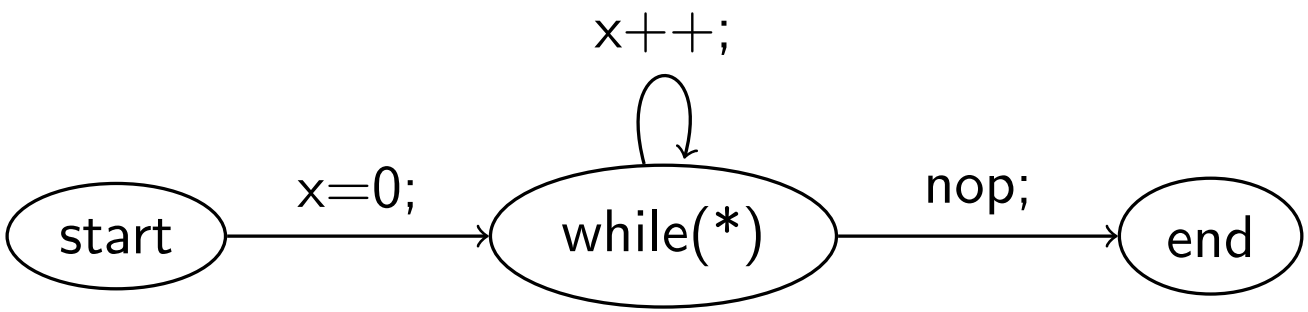
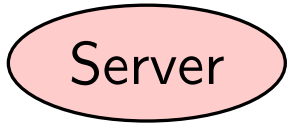


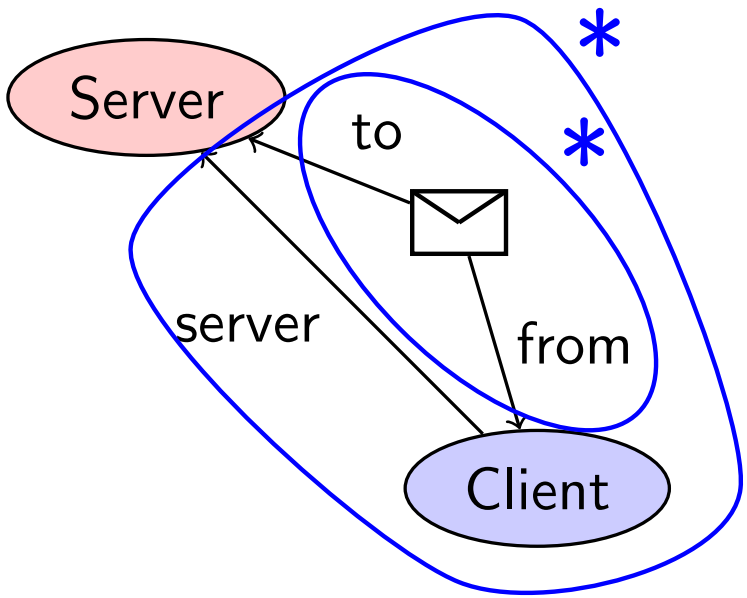
Figure: Example of a flat program

DBP are intrinsically non-flat.

initial configuration:



covering set:

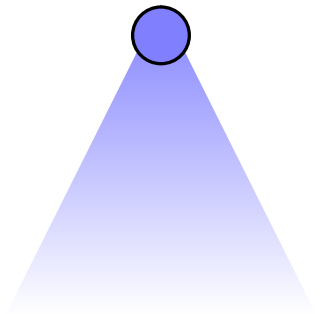


ω^2 steps from the initial state and the final state.

Nested loops are required to compute the covering set.

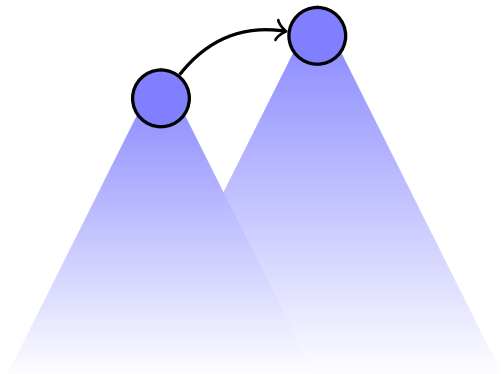
From acceleration to widening

Acceleration considers transitions - widening only states.



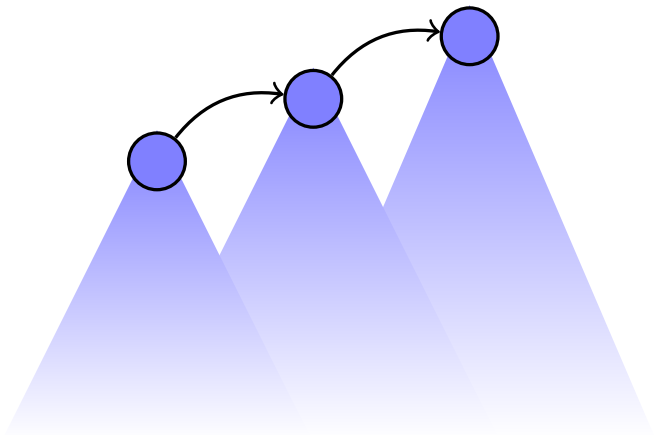
From acceleration to widening

Acceleration considers transitions - widening only states.



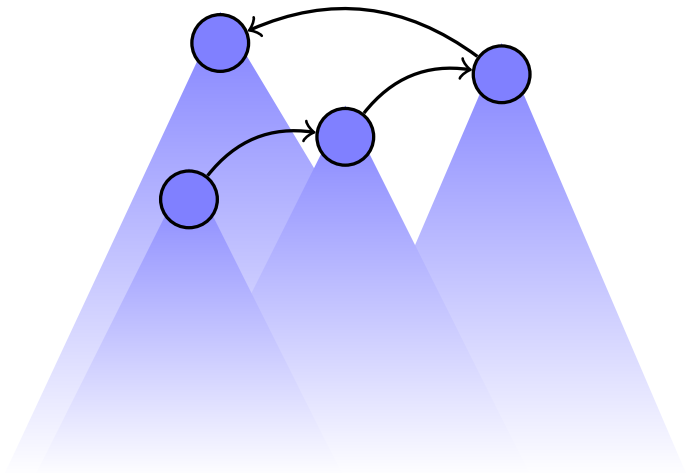
From acceleration to widening

Acceleration considers transitions - widening only states.



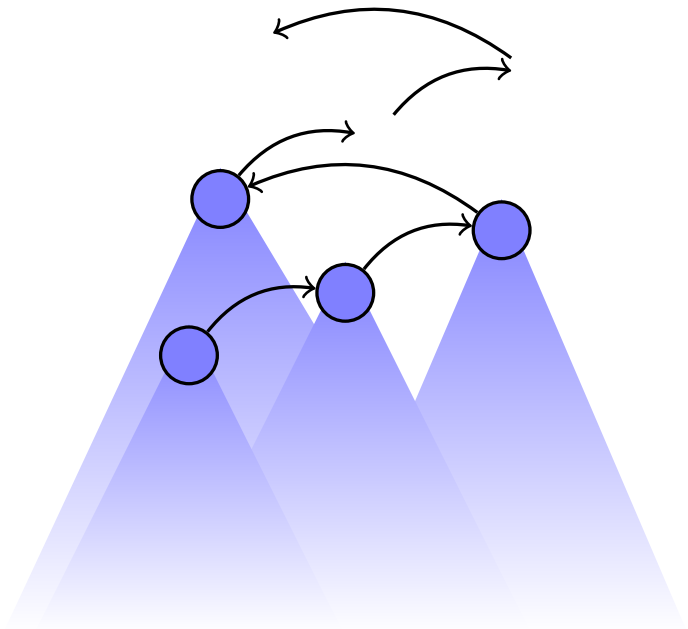
From acceleration to widening

Acceleration considers transitions - widening only states.



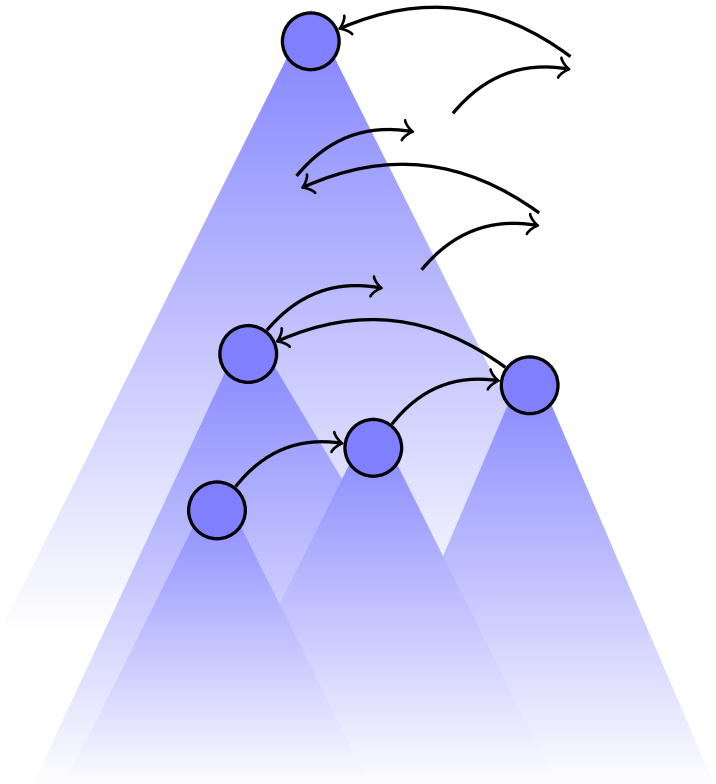
From acceleration to widening

Acceleration considers transitions - widening only states.



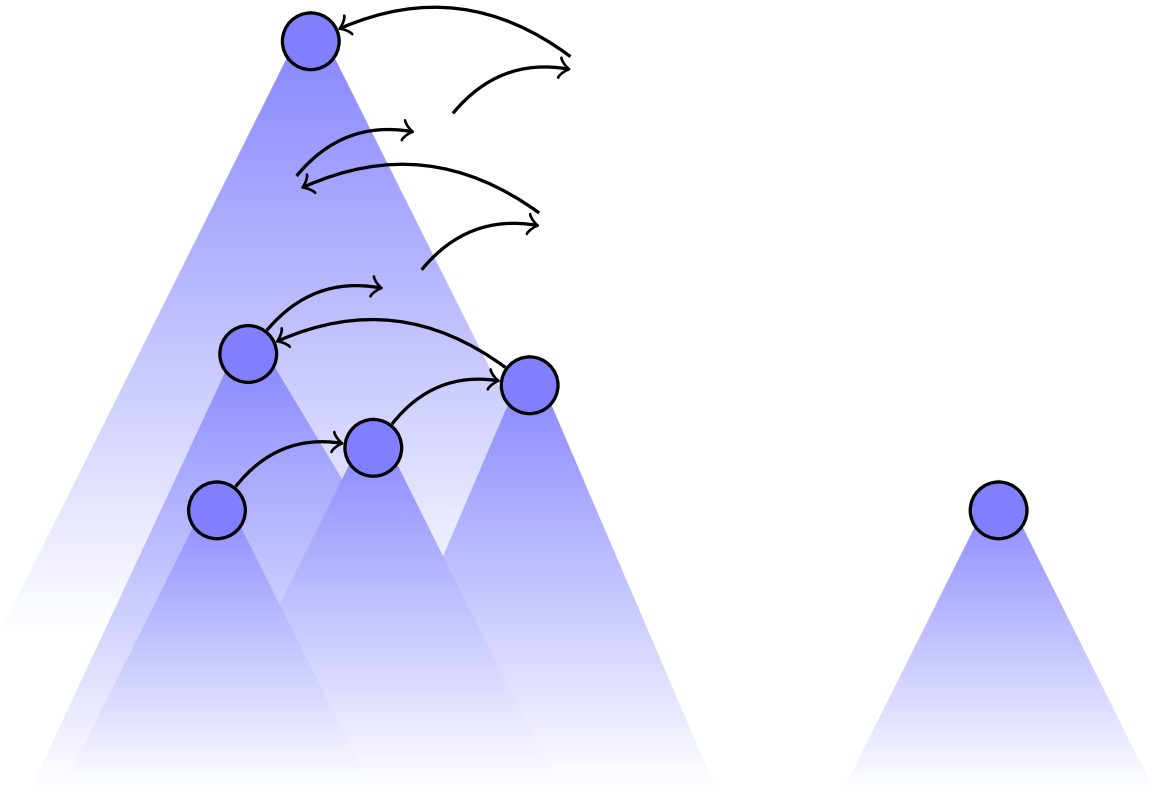
From acceleration to widening

Acceleration considers transitions - widening only states.



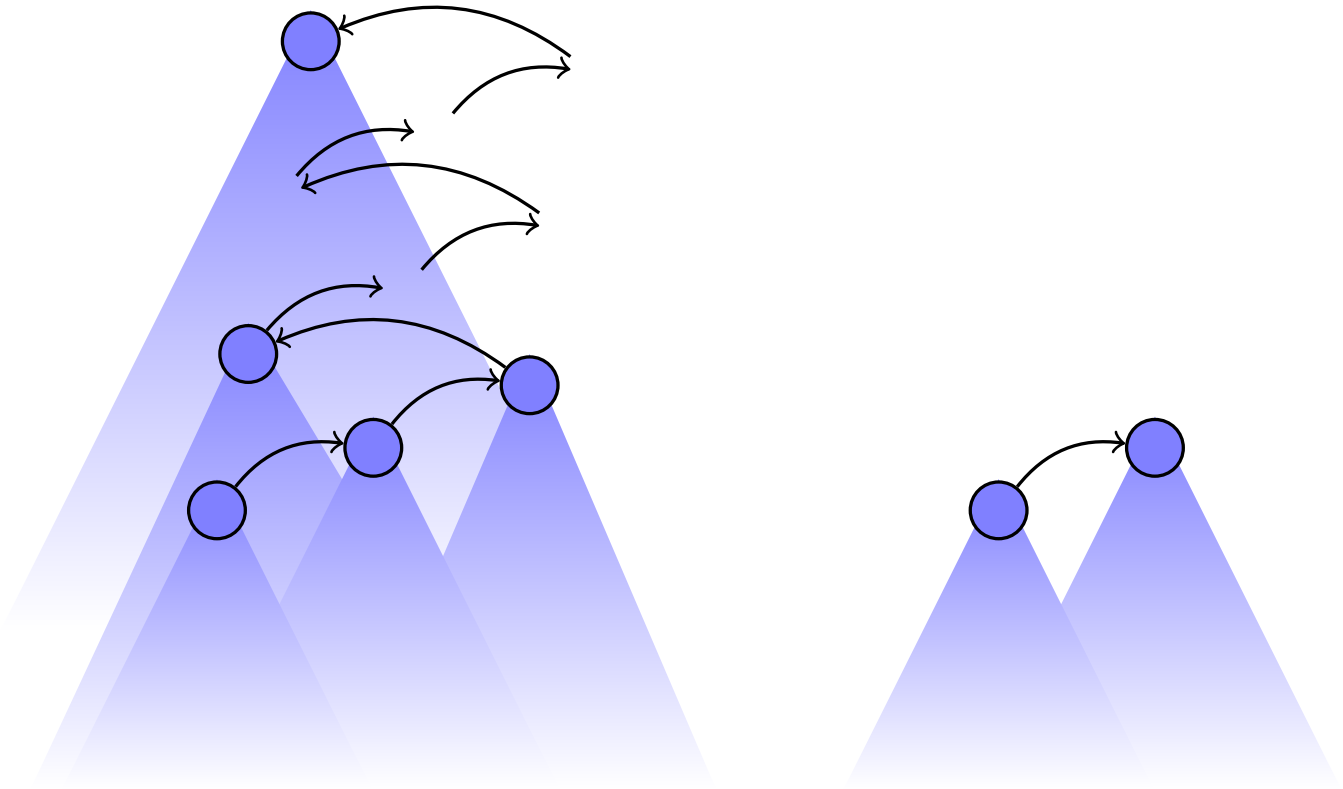
From acceleration to widening

Acceleration considers transitions - widening only states.



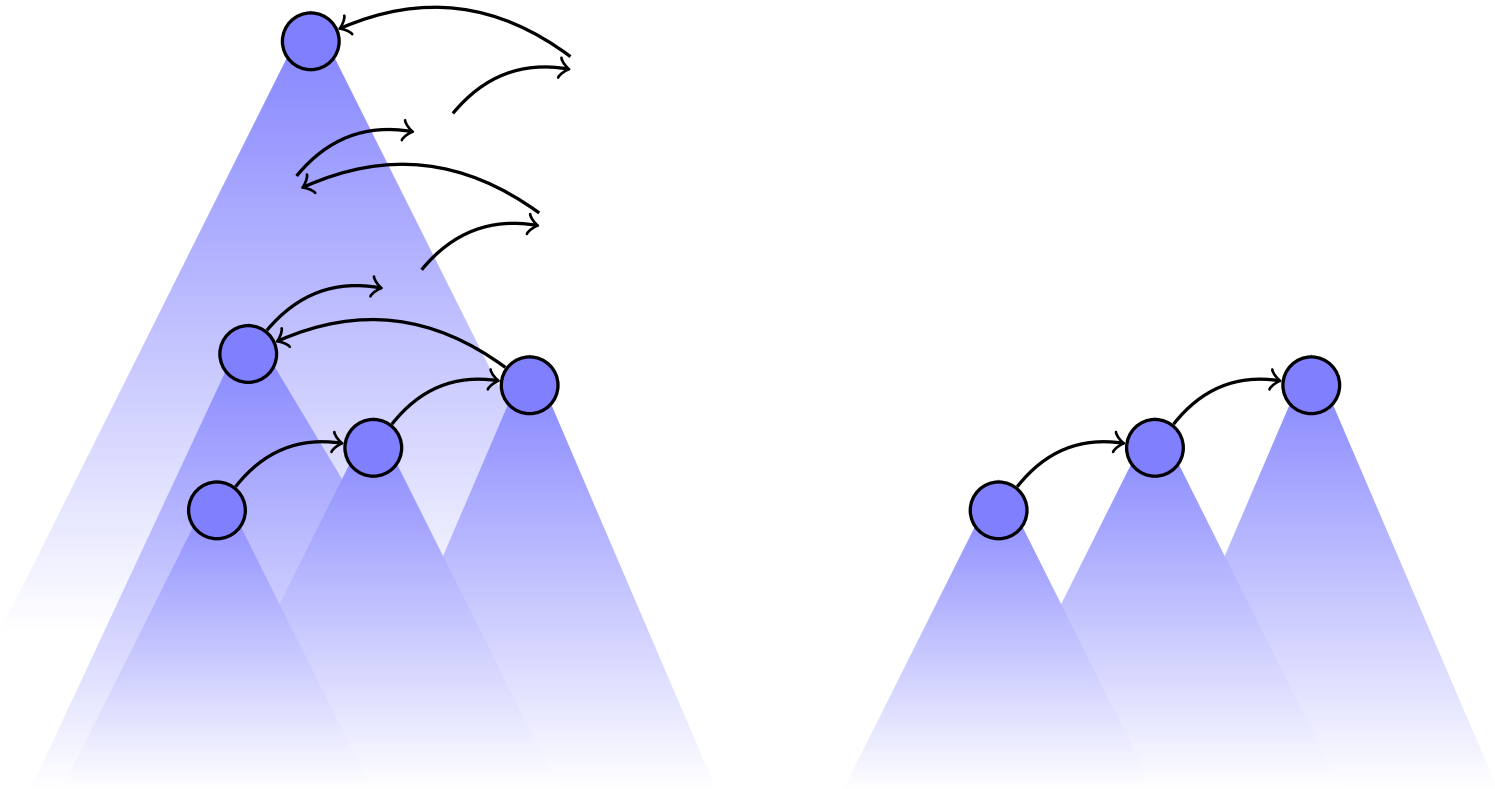
From acceleration to widening

Acceleration considers transitions - widening only states.



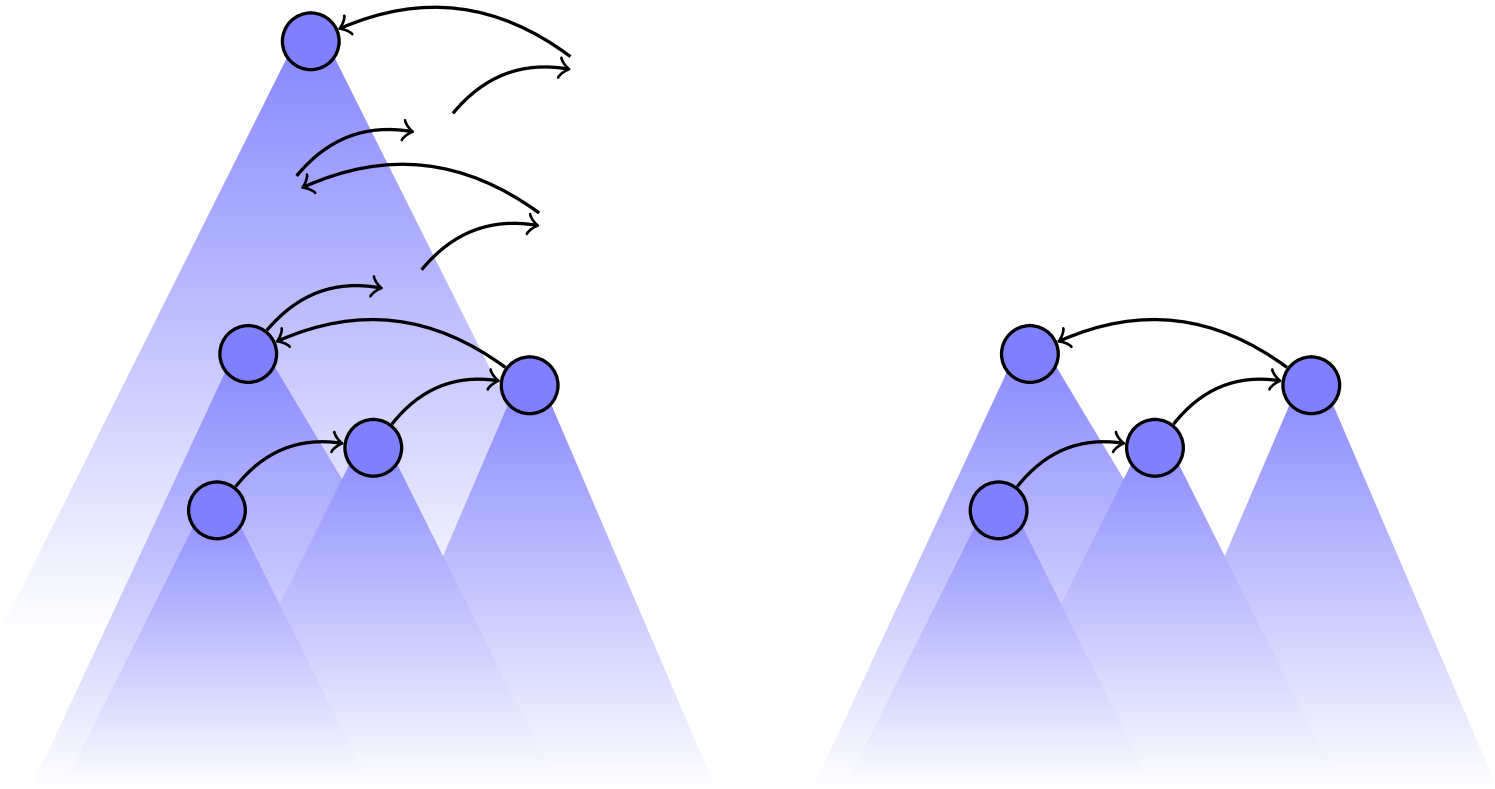
From acceleration to widening

Acceleration considers transitions - widening only states.



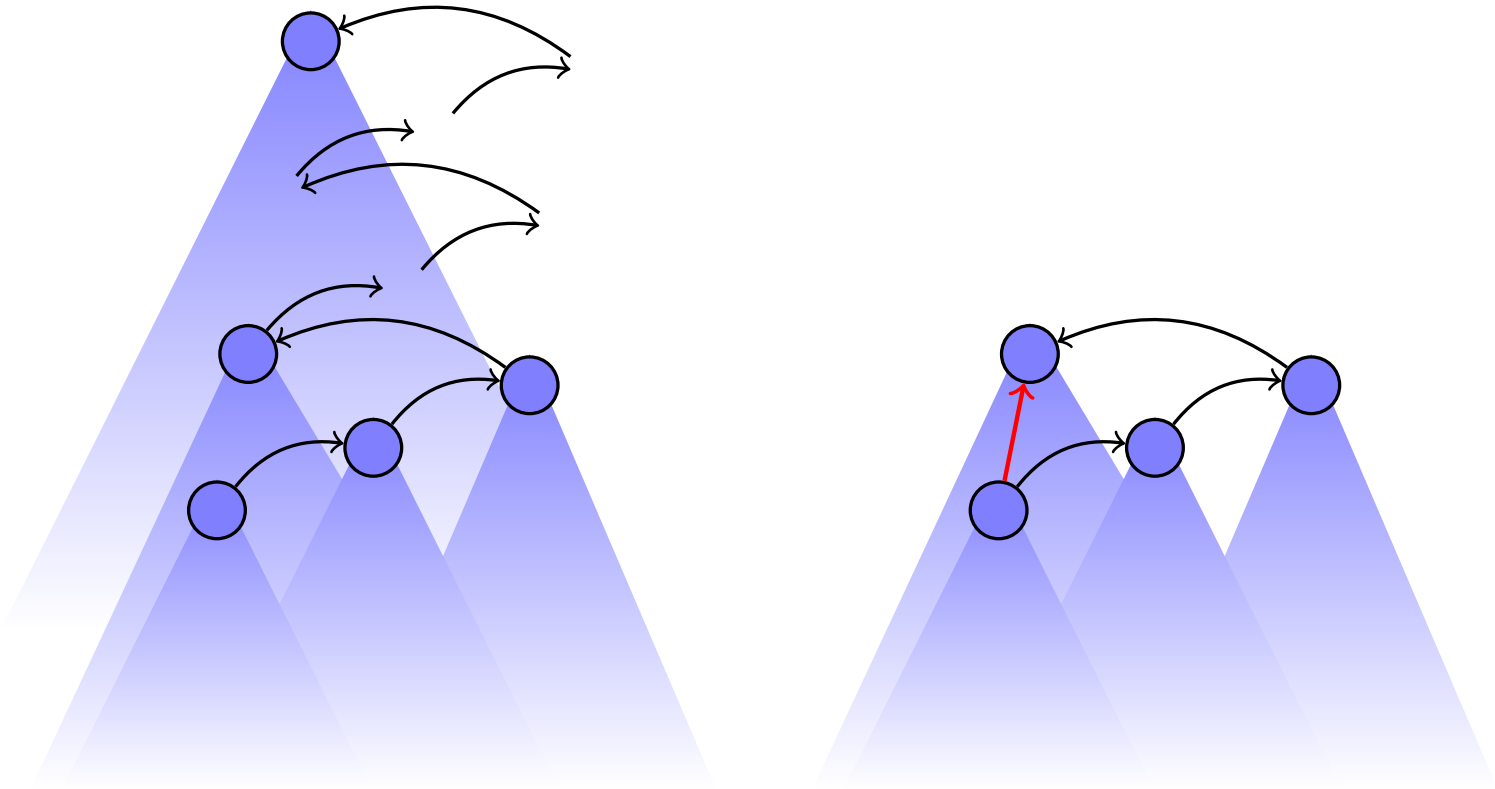
From acceleration to widening

Acceleration considers transitions - widening only states.



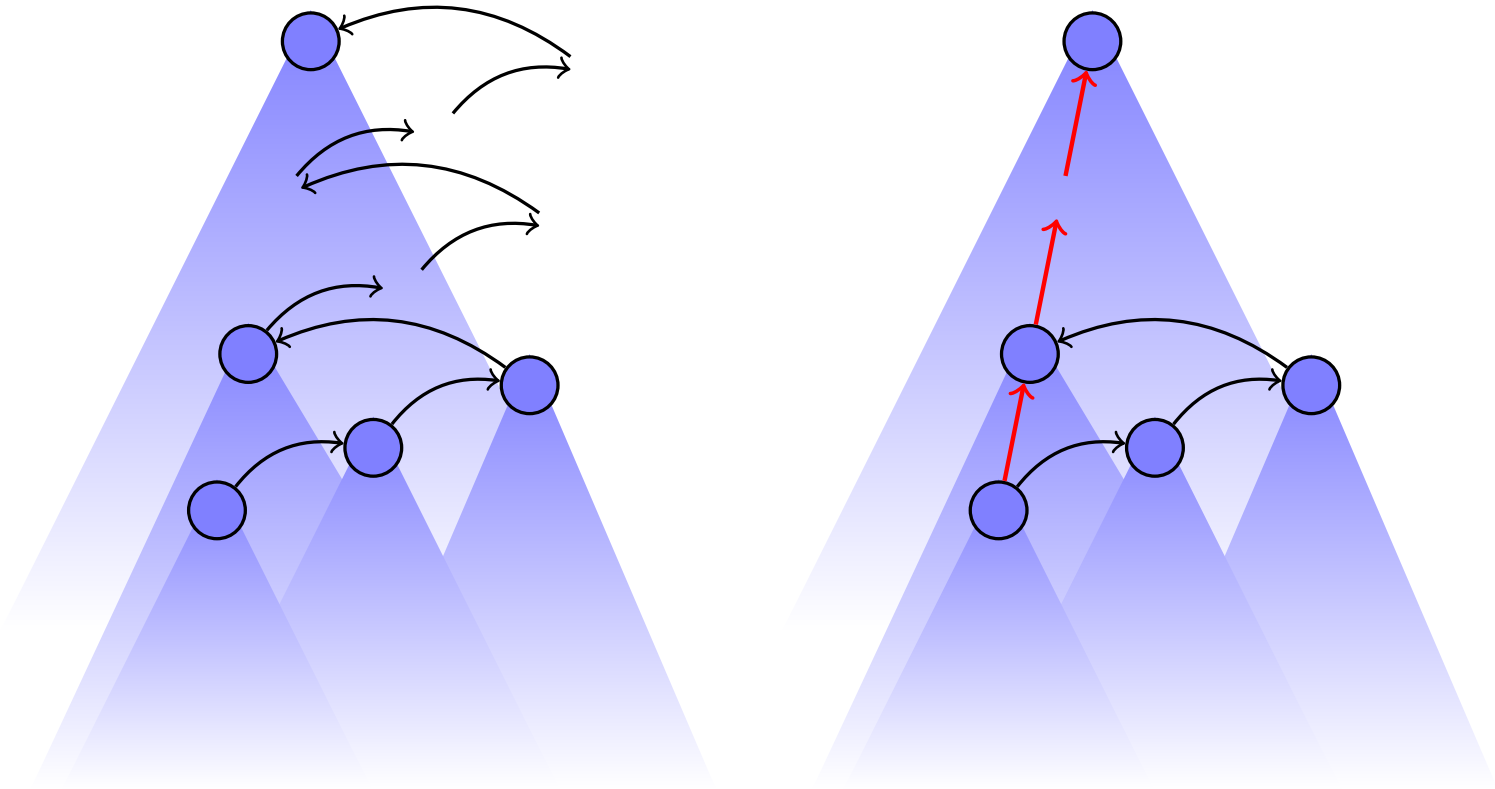
From acceleration to widening

Acceleration considers transitions - widening only states.



From acceleration to widening

Acceleration considers transitions - widening only states.



Ideal abstraction

Rephrase analysis in terms of abstract interpretation [Cousot and Cousot, 1977].

- Concrete domain: sets of configurations ($\mathcal{P}(S)$)
- Abstract domain: ideal completion of $(S, \leq)$ ($\mathcal{P}_{finite}(Idl(S))$)

- Concretization function γ: identity
- Abstraction function α: downward-closure

(α, γ) is a Galois connection.

Covering set is abstract fixed point: $\mu X.\alpha(init) \cup \alpha \cdot post \cdot \gamma(X)$

To guarantee termination we need a widening operator.

Widening (1)

Goal: try to mimic acceleration (when possible), and force termination

A set-widening operator (∇) [Cousot and Cousot, 1992] for a poset S is partial function ($\mathcal{P}(S) \to S$) that satisfies:

Covering : for all $X \subseteq S$, $x \in X \Rightarrow x \leq \nabla(X)$;

Termination : widening of any ascending chain stabilizes.

Why set-widening rather than the usual pair-widening.
Reason of using a set-widening operator: we need the history.

Widening (2)

Keeping it simple: need only set-widening operator on $Idl(S)$.
It can be lifted to $\mathcal{P}_{finite}(Idl(S))$, using a general construction (details in the paper).

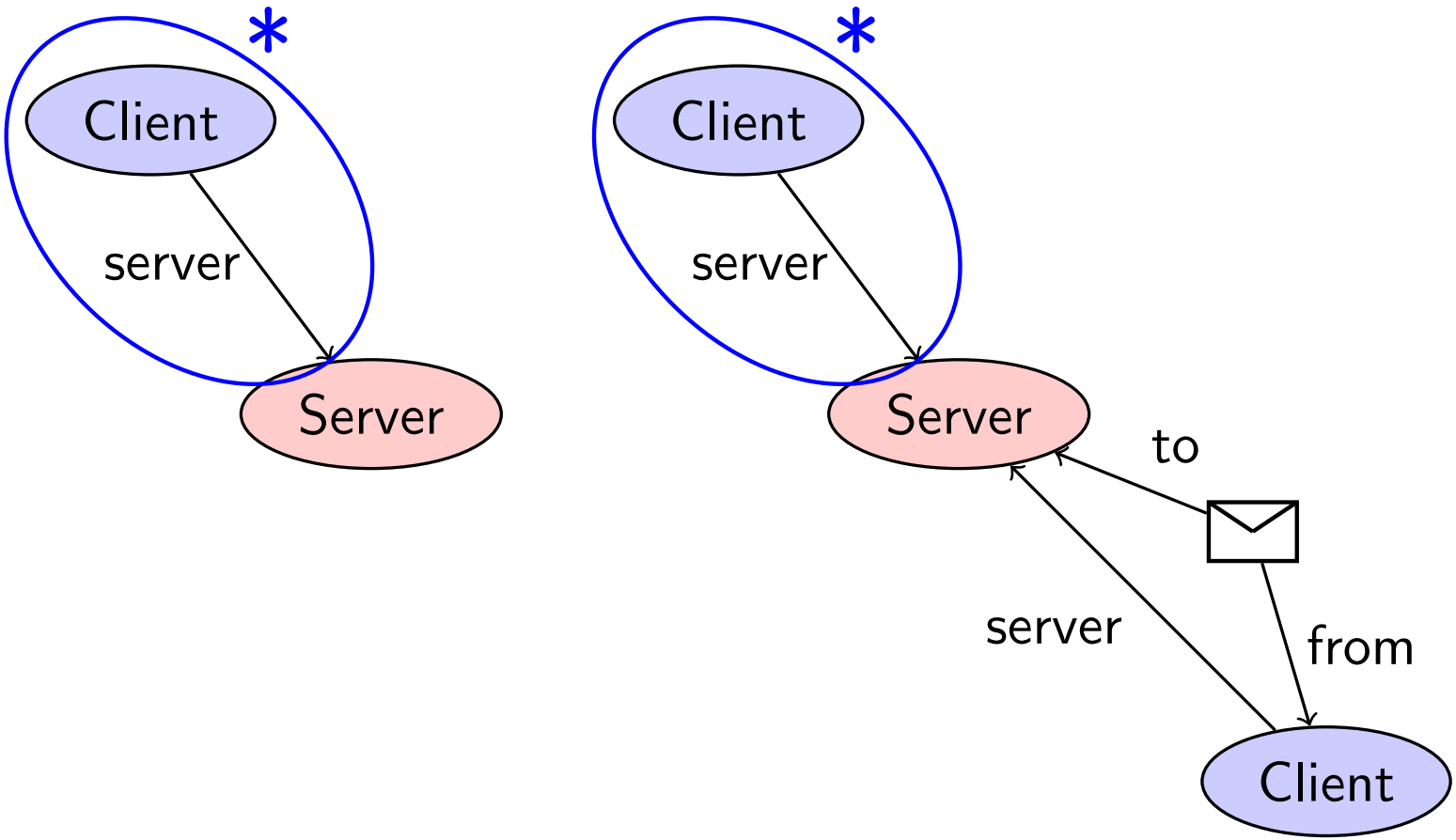
We assume that the ordering S is a better-quasi-ordering (bqo).
$\Rightarrow$ Thus, $Idl(S)$ is also a bqo.
$\Rightarrow$ No infinite antichain in $Idl(S)$.

We still need to define the widening on ideals.

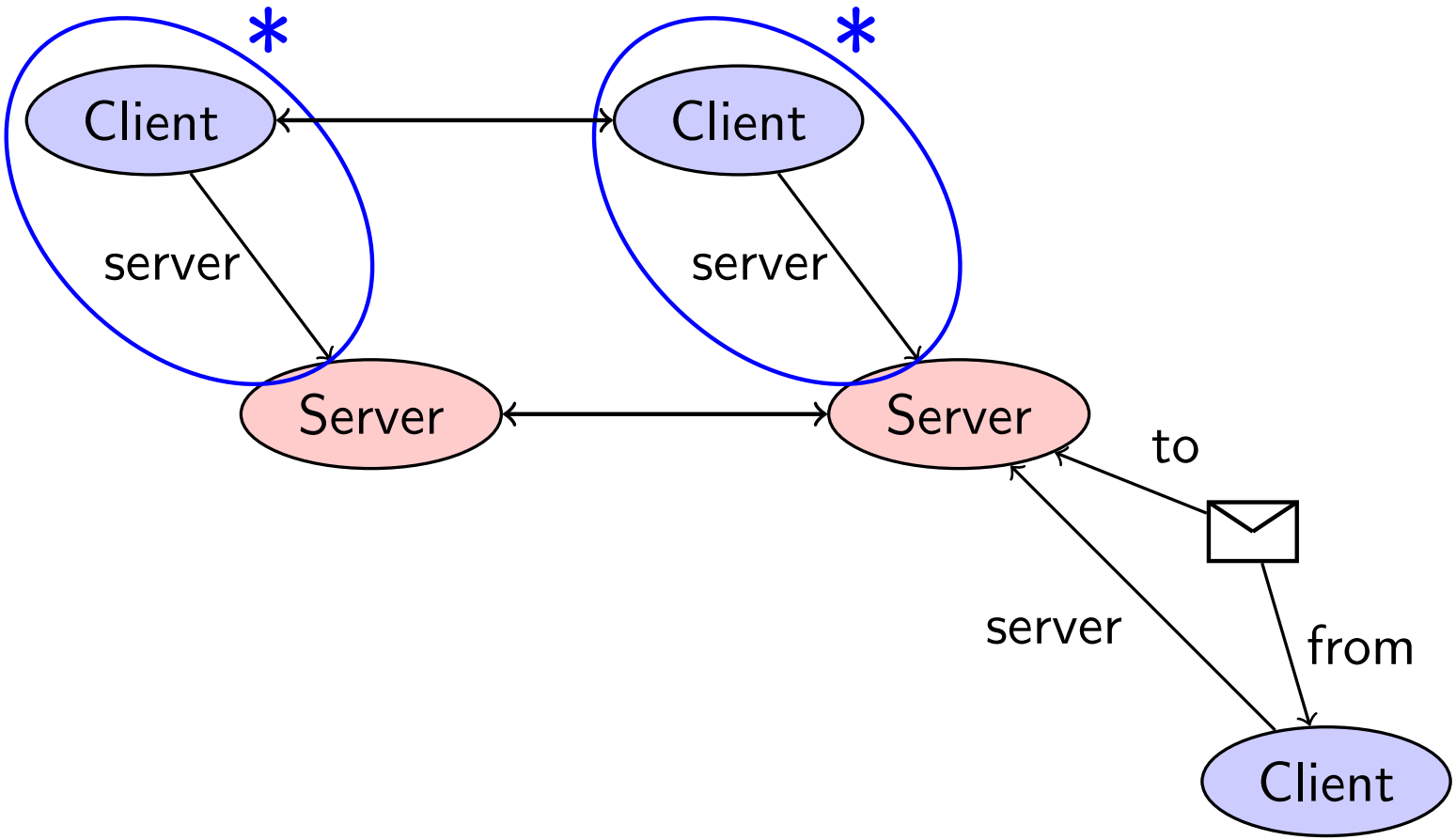
We provide concrete operators for

- Petri nets (and monotonic extensions),
- Lossy channel systems [Abdulla and Jonsson, 1993],
- Depth-bounded processes (DBP).

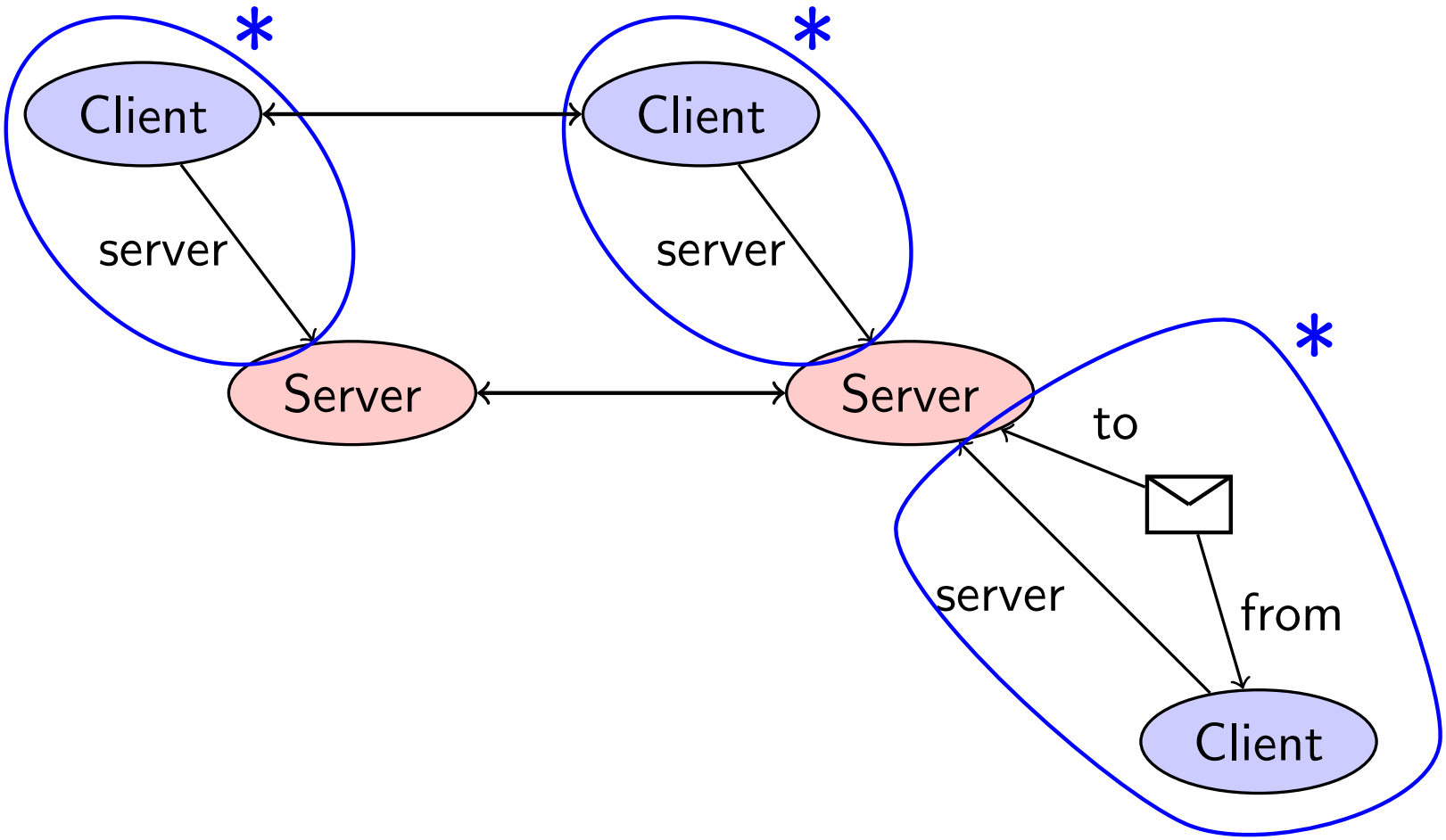
Set-widening for DBP (1)



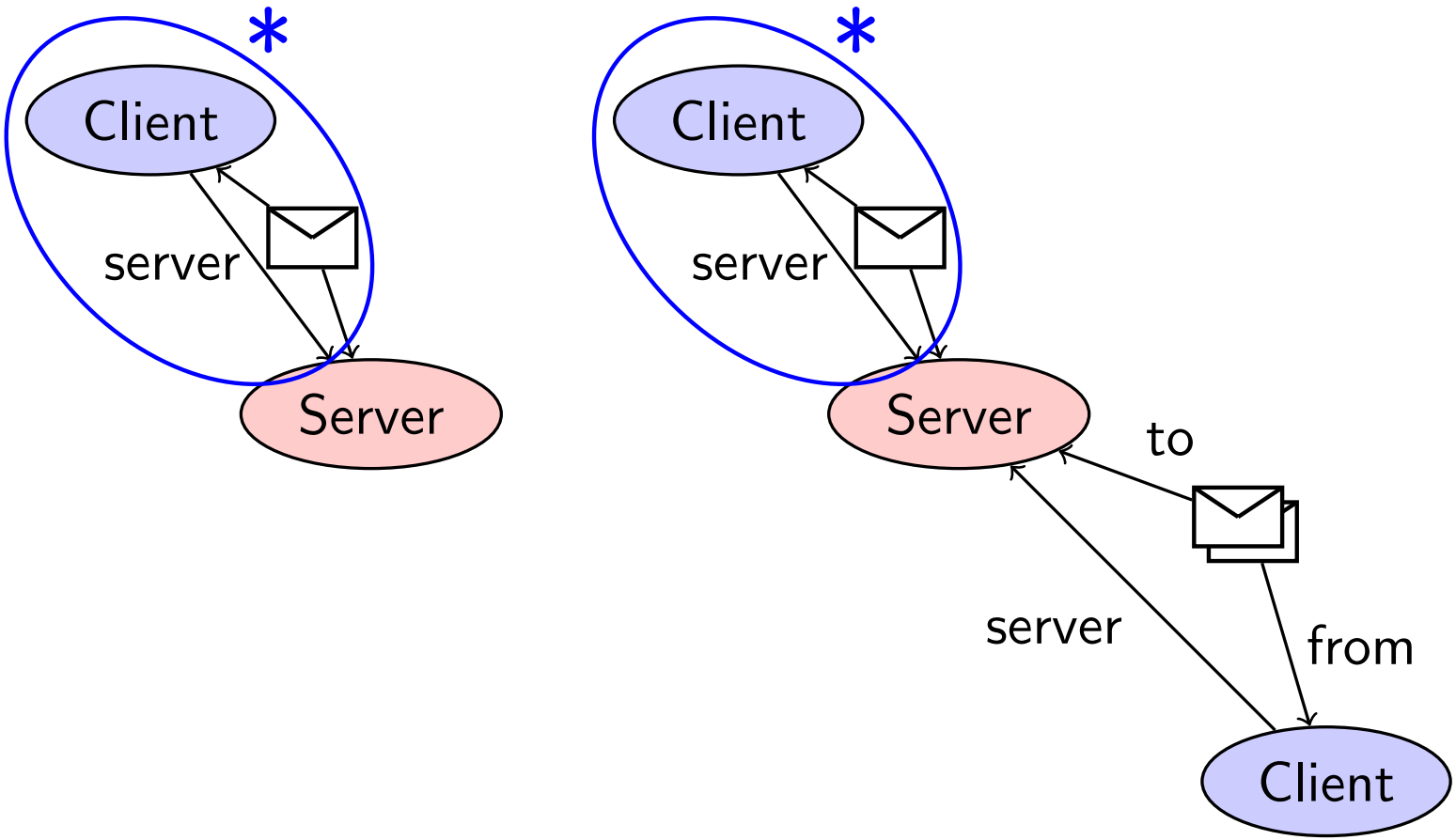
Set-widening for DBP (1)



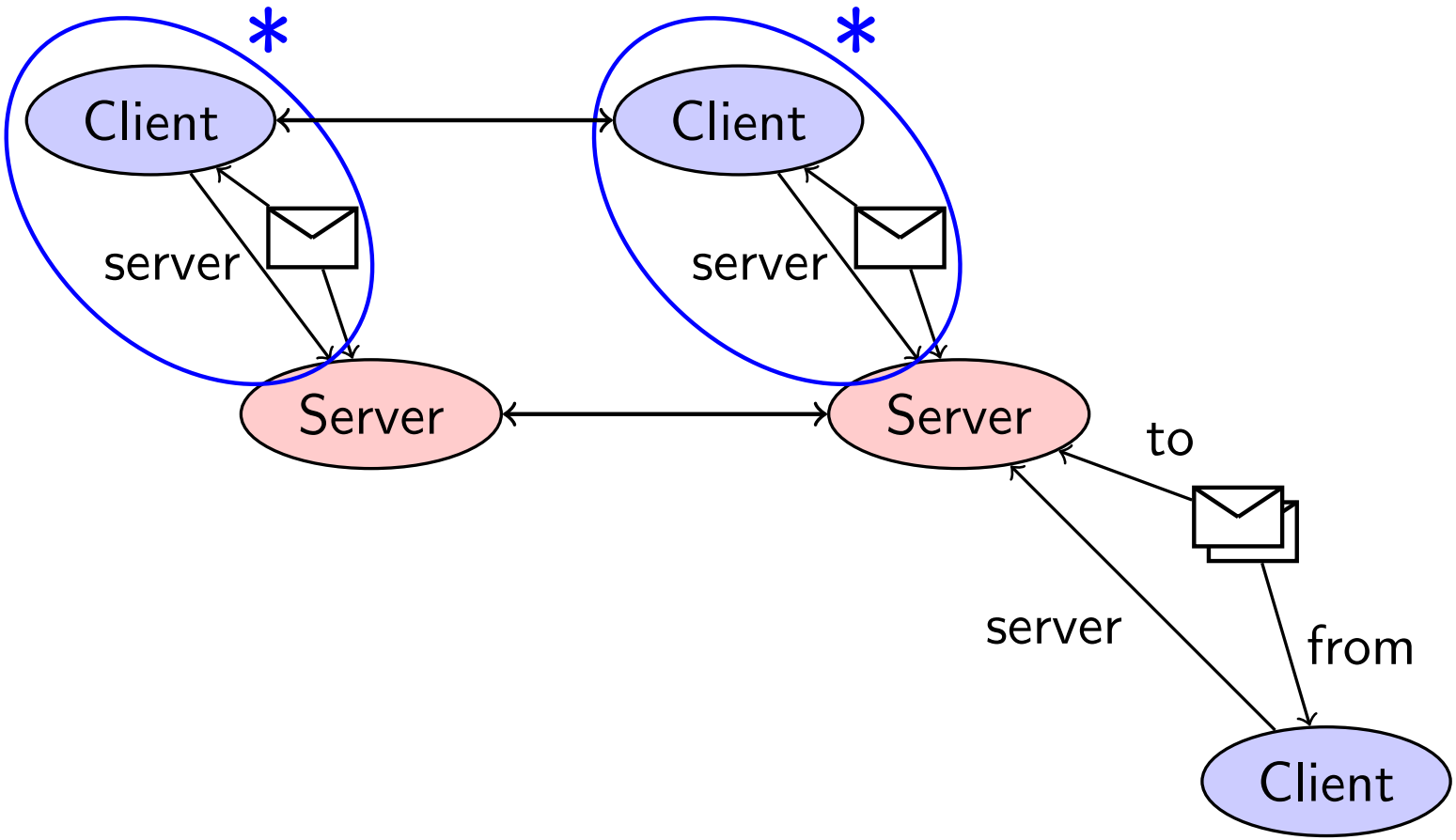
Set-widening for DBP (1)



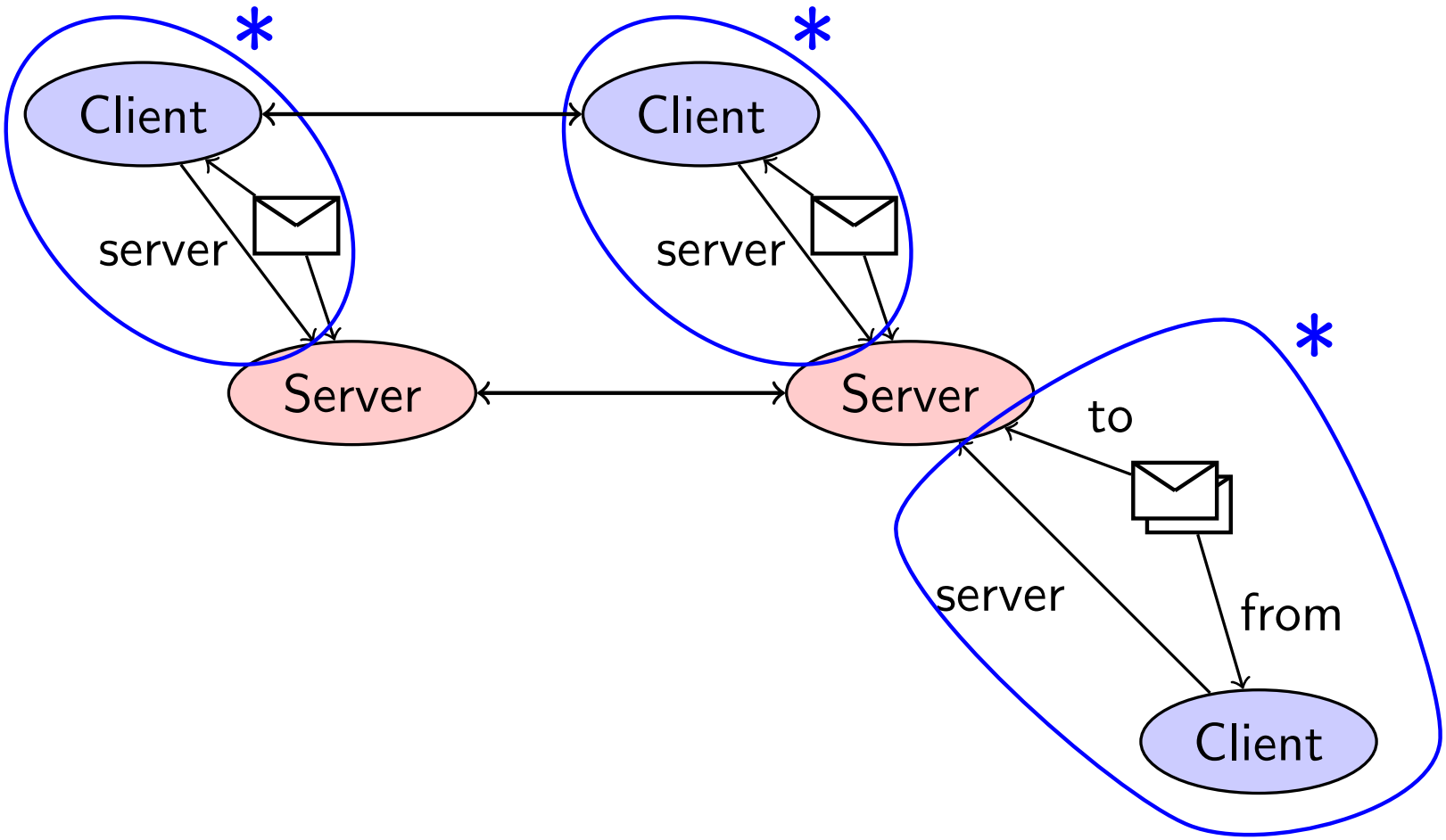
Set-widening for DBP (2)



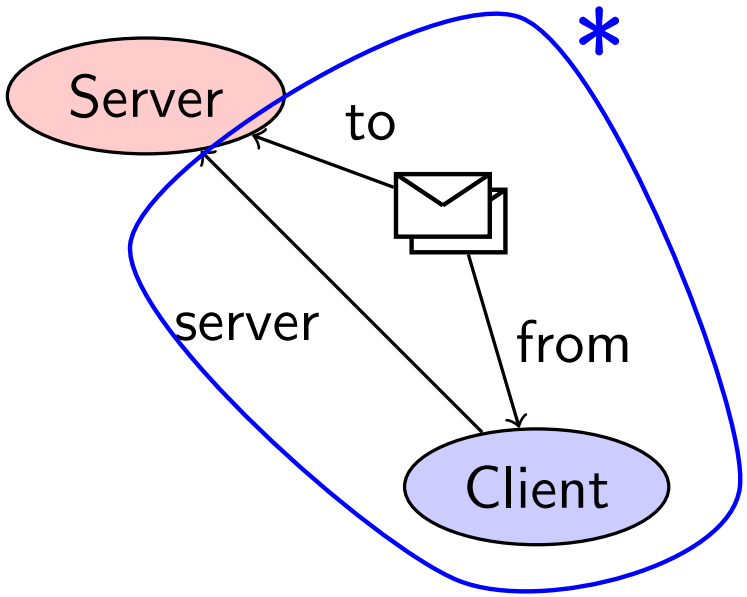
Set-widening for DBP (2)



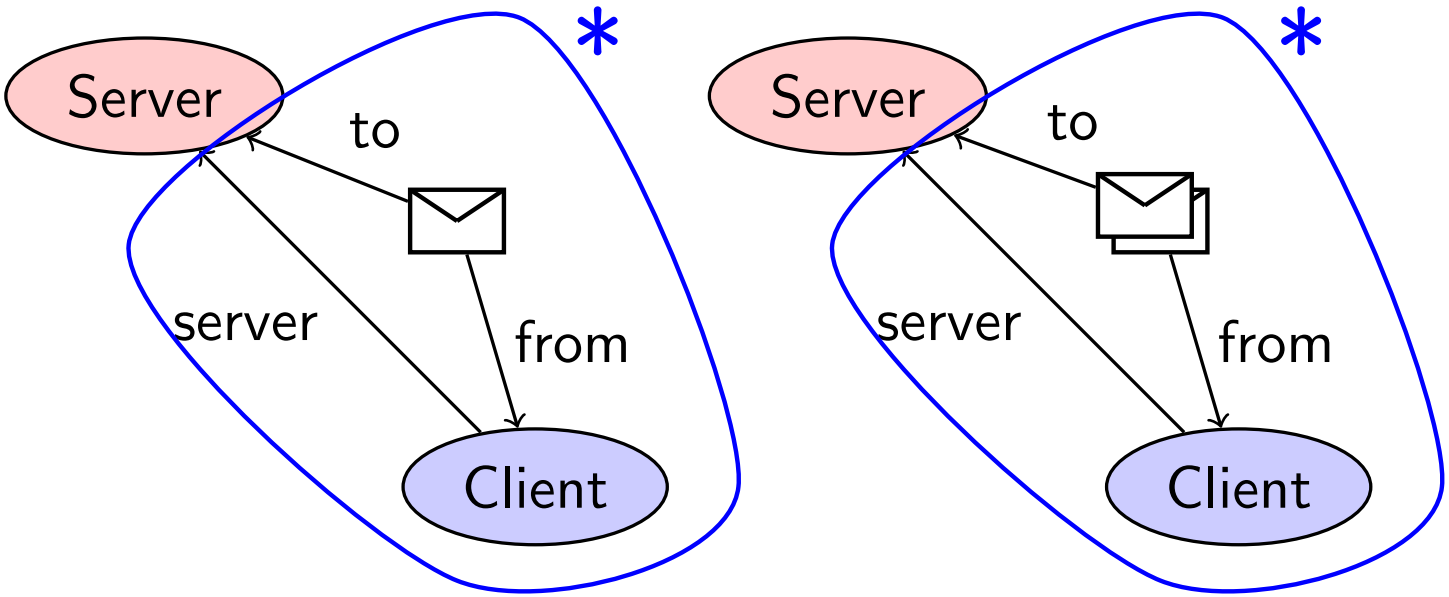
Set-widening for DBP (2)



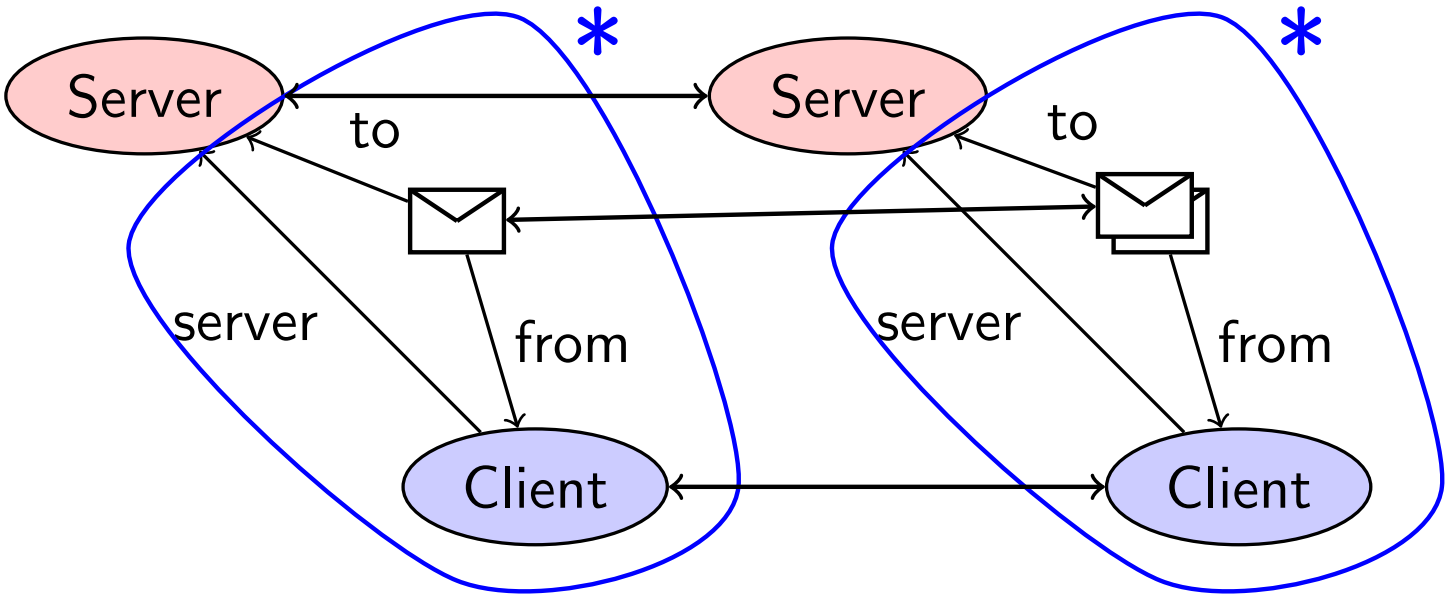
Set-widening for DBP (3)



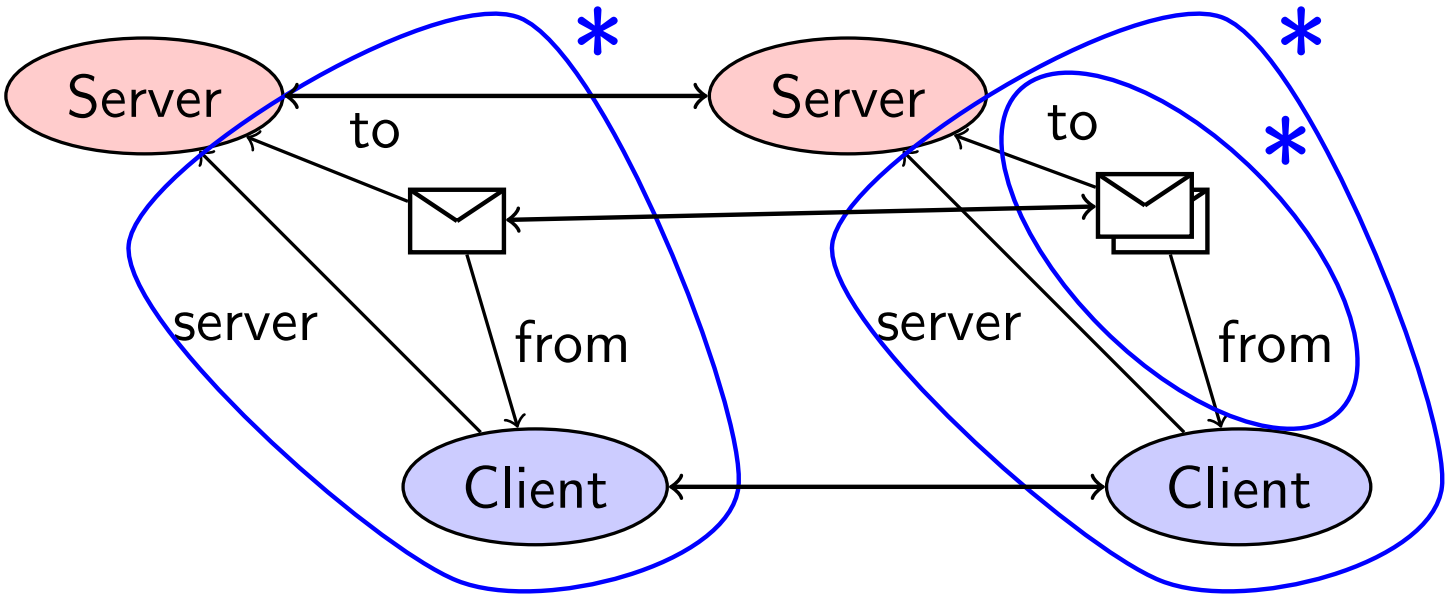
Set-widening for DBP (3)



Set-widening for DBP (3)



Set-widening for DBP (3)



How the Covering Set Grows ?

Name	tree size	cov. set size	time
`ping-pong`	17	14	0.6 s
`client-server`	25	2	1.9 s
`client-server-with-TO`	184	5	12.8 s
`genericComputeServer`	57	4	4.6 s
`genericComputeServer-fctAsActor`	98	8	14.8 s
`liftChatLike`	1846	21	1830.9 s
`round_robin_2`	830	63	48.8 s
`round_robin_3`	3775	259	737.8 s

Table: Experimental results: the columns indicate the number of nodes in the Karp-Miller tree, the number of ideals in the covering set, and the running time.

Interesting Properties of the Cover

The covering set:

- can answer an covering question (control-state reachability),
- has a compact representation,
- is an inductive invariant.

The first point was the original motivation for this work.
The last two points allows us to do much more.

Show PICASSO output for the client-server example.
PICASSO is available at
`http://pub.ist.ac.at/~zufferey/picasso/`.

Table of Contents

Termination Proofs and Counter Abstractions

Termination proof:
finite union of ranking function [Podelski and Rybalchenko, 2004]

Counter abstraction:
counts the number of occurrences of some elements in a system.
Typically one counts the number of processes in a state/control-location.

In this work we automatically infer complex termination proof using a flexible counter abstraction. The number of counters is not fixed a priori, but depends on the reachable states in the system. This is joint work with Kshitij Bansal, Eric Koskinen, and Thomas Wies (just rejected from POPL).

Treiber's stack: code

```
struct node {
    struct node *next;
    value data;
};
struct stack { struct node *Top; };
struct stack *S;

value pop() {
  struct node *t, *x;
  do {
    t = S->Top;
    if (t == NULL) return EMPTY;
    x = t->next;
  } while (!CAS(&S->Top,t,x));
  return t->data;
}
```

```
void init() {
    S = alloc();
    S->Top = NULL;
}


void push(value v) {
  struct node *t, *x;
  x = alloc();
  x->data = v;
  do {
      t = S->Top;
      x->next = t;
  } while (!CAS(&S->Top,t,x));
}
```

Treiber's stack: as DBS and Covering Set

Encoding:

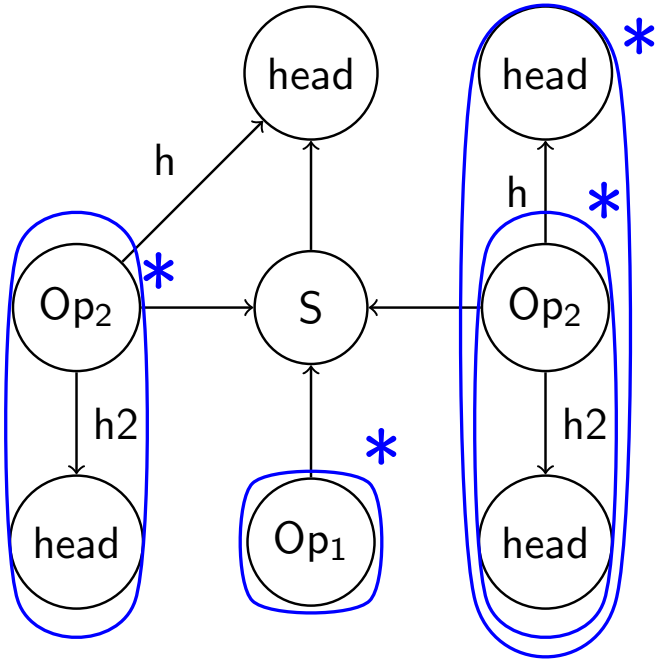
- `push` and `pop` are a single operation.
- Operation: first read the top of stack, then CAS.

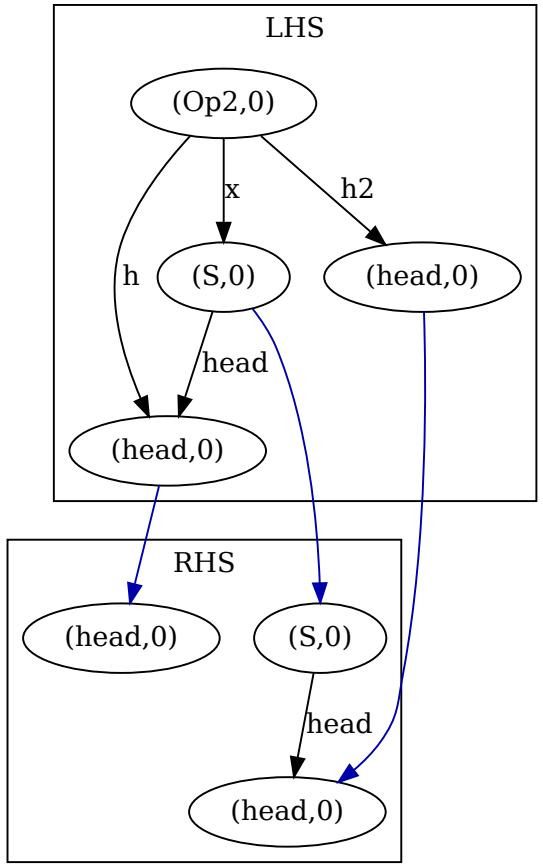
Invariant for termination:
A process cannot fail twice unless someone else did succeed.

Show PICASSO output: rewrite rules, cover, and cover + post.

Numerical Abstraction from the Covering Set

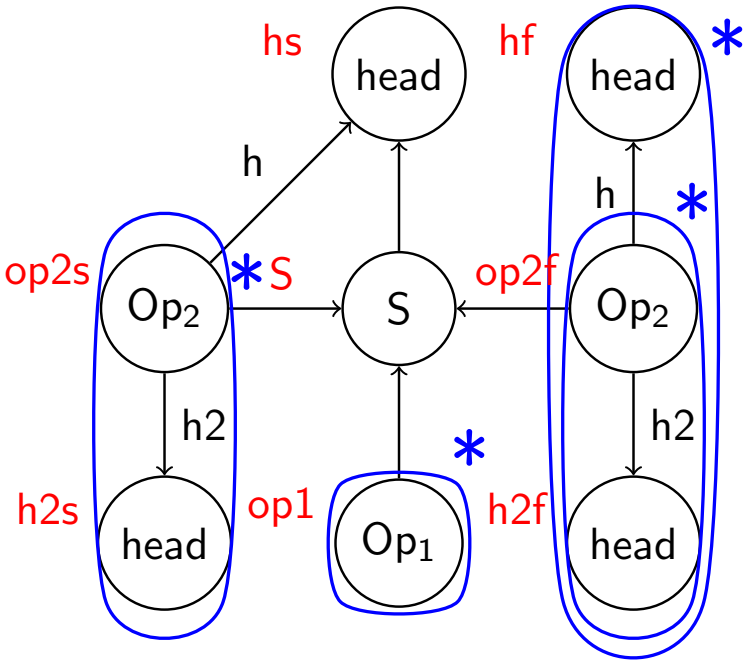
Op2: CAS succeed

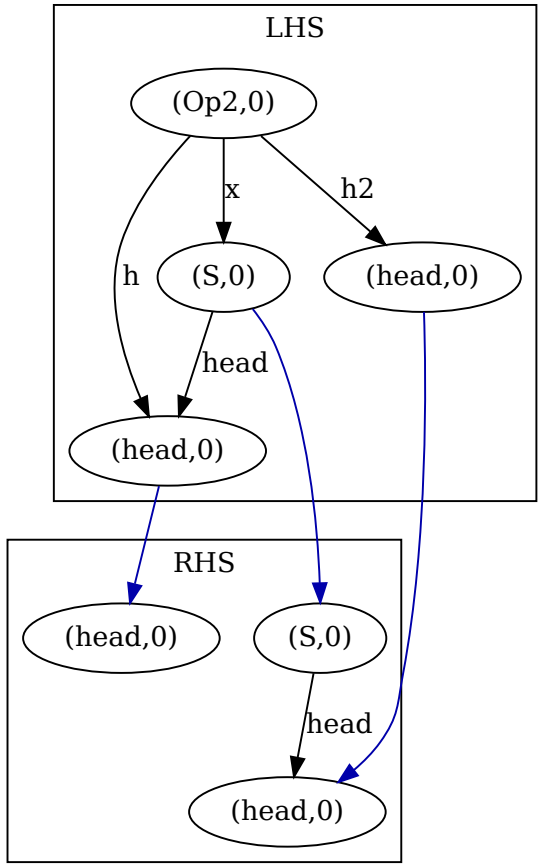




Numerical Abstraction from the Covering Set

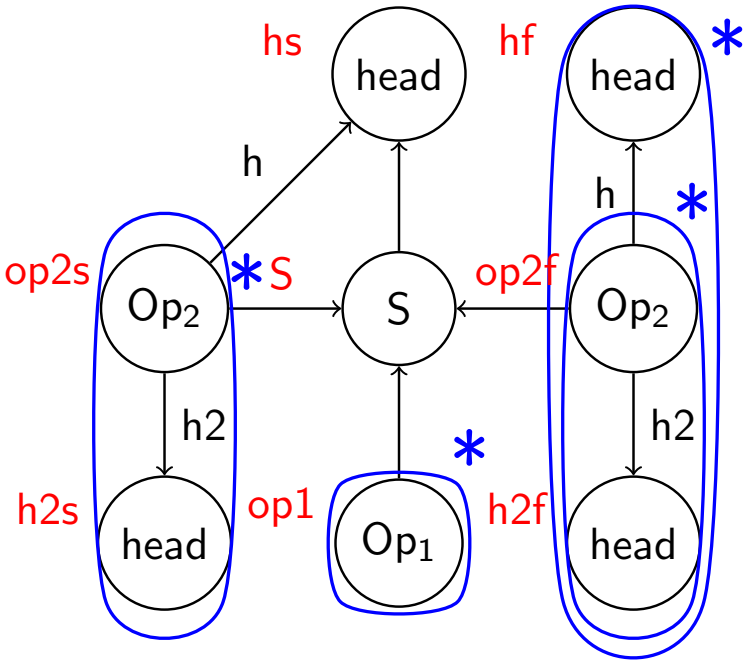
Op2: CAS succeed

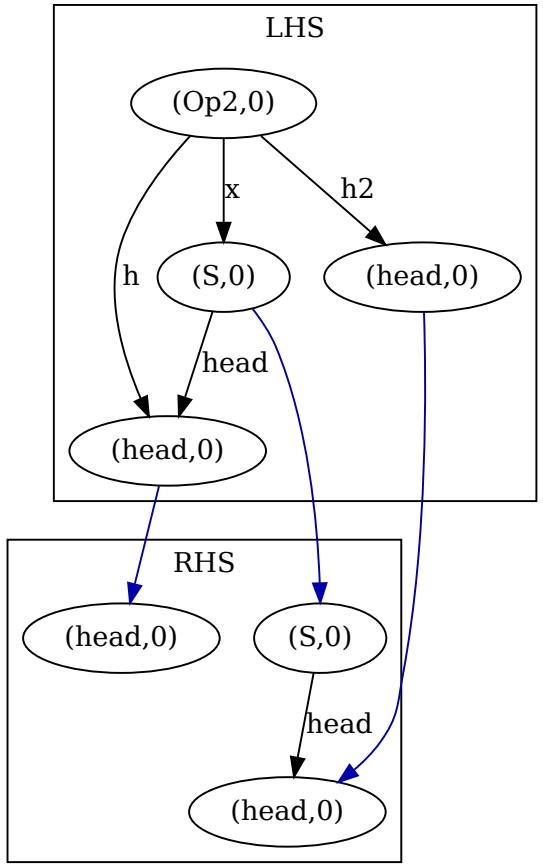




Numerical Abstraction from the Covering Set

Op2: CAS succeed





S' = S
op1' = op1
op2s' = 0

op2f' = op2f + op2s -1
h2f' = h2f + h2s - 1
hf' = hf + 1

hs' = 1
h2s' = 0

Theoretical results

Soundness:

The numerical abstraction preserves the weak-fair termination.

Preserving Monotonicity:

The numerical abstraction preserves the monotonicity of the DBS. The numerical abstraction is actually a Petri Net.

Implementation

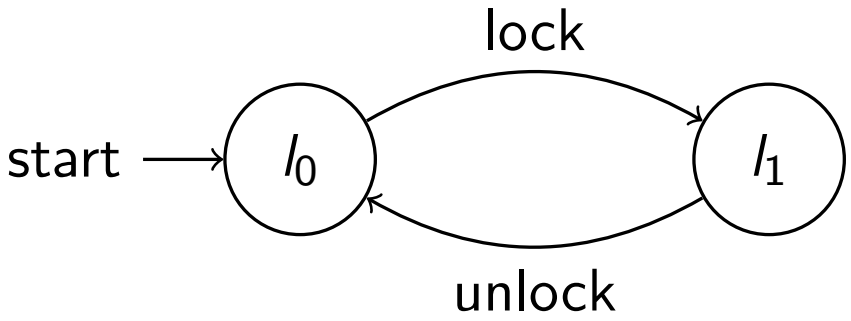
Example	**Cov size**	**Cov+$\mathcal{N}$** (sec)	**ARMC** (sec)	**Total** (sec)
`split/merge`	3	3.5	4.5	8.0
Work stealing, 3 processors	3	5.4	1.0	6.4
Parameterized work stealing	1	3.7	0.9	4.6
Compute server job queue	1	3.9	1.2	5.1
Map reduce	4	5.6	0.9	6.5
Map reduce with failure	5	7.9	6.0	13.9
Treiber's stack	1	3.8	0.5	4.3
Treiber's stack (fine-grained)	3	10.3	4.7	15.0
Herlihy Wing queue	2	20.4	85.9	106.3

Table: Experimental results. Columns indicate the size of the cover, the running time for its construction and the construction of the numerical abstraction, the running time of the final termination, and the total time of our method.

Table of Contents

Object Interfaces

Example: one should invoke the lock and unlock methods of a lock object in alternation, starting by lock.



In this work we use the covering set to extract interface for *groups of objects* in a non-concurrent setting. This is joint work with Shahram Esmaeilsabzali, Rupak Majumdar, and Thomas Wies (submitted to ICSE).

Example: Sets and Iterators

```
class Set {
  protected int sver, size;
  public Set() {
    sver := size := 0;
  }
  public void add(int elem) {
    if (!duplicate) {
      sver++; size++;
    }
  }
  protected void delete(int pos) {
    sver++; size--;
  }
  public Iterator iterator() {
    return (new Iterator(this));
  }
}
```

```
class Iterator {
  int iver, pos;
  Set it_of;
  protected Iterator(Set s) {
    it_of := s; pos := 0;
    iver := s.sver;
  }
  public int next() {
    if (iver == S.sver) then pos++;
    else throw new Exception();
  }
  public void remove() {
    if (iver == S.sver) then {
      it_of.delete(pos);
      iver := S.sver;
    } else {
      throw new Exception();
} } }
```

Predicate Abstraction and DBS Encoding

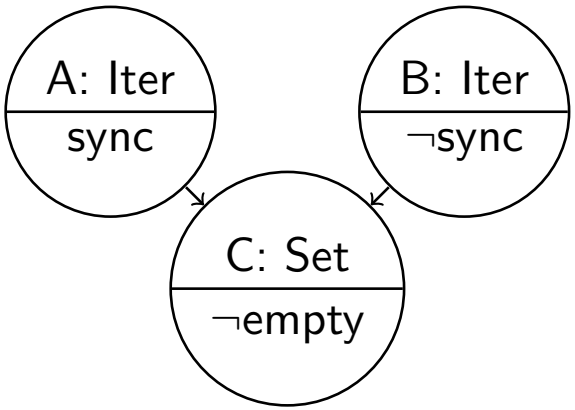
How do we get from the code to the DBS ?
We use predicates abstraction and symbolic execution.

- unary predicate for `Set`:
  empty (`size = 0`)
- binary predicates for `Set` and `Iterator`:
  sync (`iver = it_of.sver`)

Rewriting rule for `Iterator.remove`

We have to consider a few cases:

- An iterator can be either synchronized or not.
- An iterator can be either the caller or part of the frame.



$$
\begin{aligned}
\varphi(A, A) &\Rightarrow wp(A.\text{remove}, \text{sync}_{A'}) \\
\varphi(A, A) &\Rightarrow wp(A.\text{remove}, \neg\text{sync}_{A'}) \\
\varphi(B, A) &\Rightarrow wp(A.\text{remove}, \text{sync}_{B'}) \\
\varphi(B, A) &\Rightarrow wp(A.\text{remove}, \neg\text{sync}_{B'}) \\
\varphi(C, A) &\Rightarrow wp(A.\text{remove}, \text{empty}_{C'}) \\
\varphi(C, A) &\Rightarrow wp(A.\text{remove}, \neg\text{empty}_{C'})
\end{aligned}
$$

To know what cases we must consider, we go back and forth between the cover computation and the symbolic execution.

Extracting the DPI

- compute the covering set and the DBS
- apply the post operator once more
- track each transition to get the DPI

Show PICASSO output for the Set and Iterators example.

Conclusion

We present an analysis to compute an over-approximation of the covering set for BDS.
Then we show that on top of answering coverability question the covering set can be used as starting point for more advanced analysis:

- numerical abstraction to prove (weak-fair) termination;
- interfaces for groups of objects.

References I

Abdulla, P. A., Cerans, K., Jonsson, B. and Tsay, Y.-K. (1996).
General Decidability Theorems for Infinite-State Systems.
In LICS pp. 313–321,.

Abdulla, P. A. and Jonsson, B. (1993).
Verifying Programs with Unreliable Channels.
In LICS pp. 160–170,.

Bardin, S., Finkel, A., Leroux, J. and Schnoebelen, P. (2005).
Flat Acceleration in Symbolic Model Checking.
In ATVA pp. 474–488,.

Cousot, P. and Cousot, R. (1977).
Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints.
In POPL pp. 238–252,.

Cousot, P. and Cousot, R. (1992).
Abstract Interpretation Frameworks.
Journal of Logic and Computation *2*, 511–547.

Finkel, A. and Goubault-Larrecq, J. (2009).
Forward Analysis for WSTS, Part I: Completions.
In STACS vol. 09001, of Dagstuhl Sem. Proc. pp. 433–444,.

References II

Finkel, A. and Schnoebelen, P. (2001).
Well-structured transition systems everywhere!
Theor. Comput. Sci. *256*, 63–92.

Geeraerts, G., Raskin, J.-F. and Van Begin, L. (2006).
Expand, Enlarge and Check: New algorithms for the coverability problem of WSTS.
J. Comput. Syst. Sci. *72*, 180–203.

Meyer, R. (2008).
On Boundedness in Depth in the pi-Calculus.
In IFIP TCS vol. 273, of IFIP pp. 477–489, Springer.

Podelski, A. and Rybalchenko, A. (2004).
Transition invariants.
In LICS pp. 32–41,.

Wies, T., Zufferey, D. and Henzinger, T. A. (2010).
Forward Analysis of Depth-Bounded Processes.
In FoSSaCS 2010 vol. 4349, of LNCS pp. 94–108, Springer.