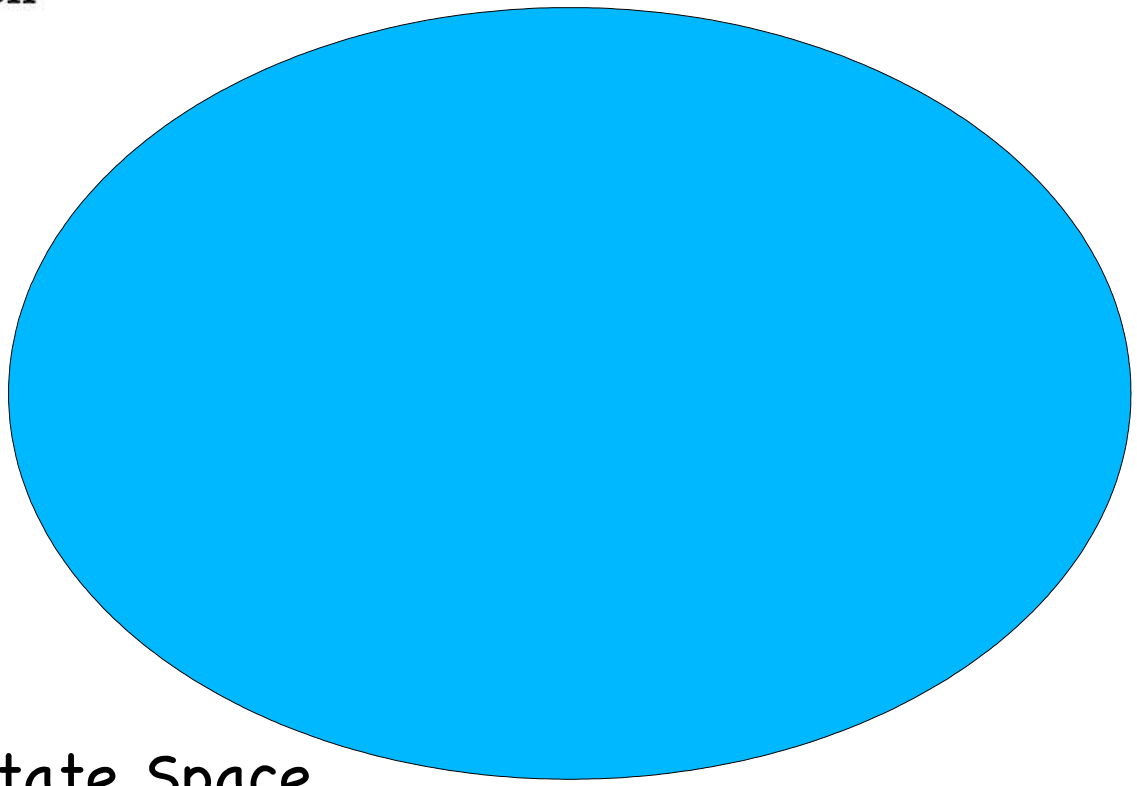
Distributed TA Reachability

$waiting_A = \{(l_0, Z_0 \wedge I(l_0)) \mid h(l_0) = A\}$
$passed_A = \varnothing$
while $\neg terminated$ **do**
  select and remove a state (l, Z) from $waiting_A$
  if $\forall (l, Y) \in passed_A : Z \not\subseteq Y$ **then**
    $passed_A = passed_A \cup \{(l, Z)\}$
    for all successors (l', Z') of (l, Z) **do**
      $d = h(l')$
      if $\forall (l', Y') \in waiting_d : Z' \not\subseteq Y'$ **then**
        $waiting_d = waiting_d \cup \{(l', Z')\}$
      endif
    done
  endif
done

Distributed Reachability

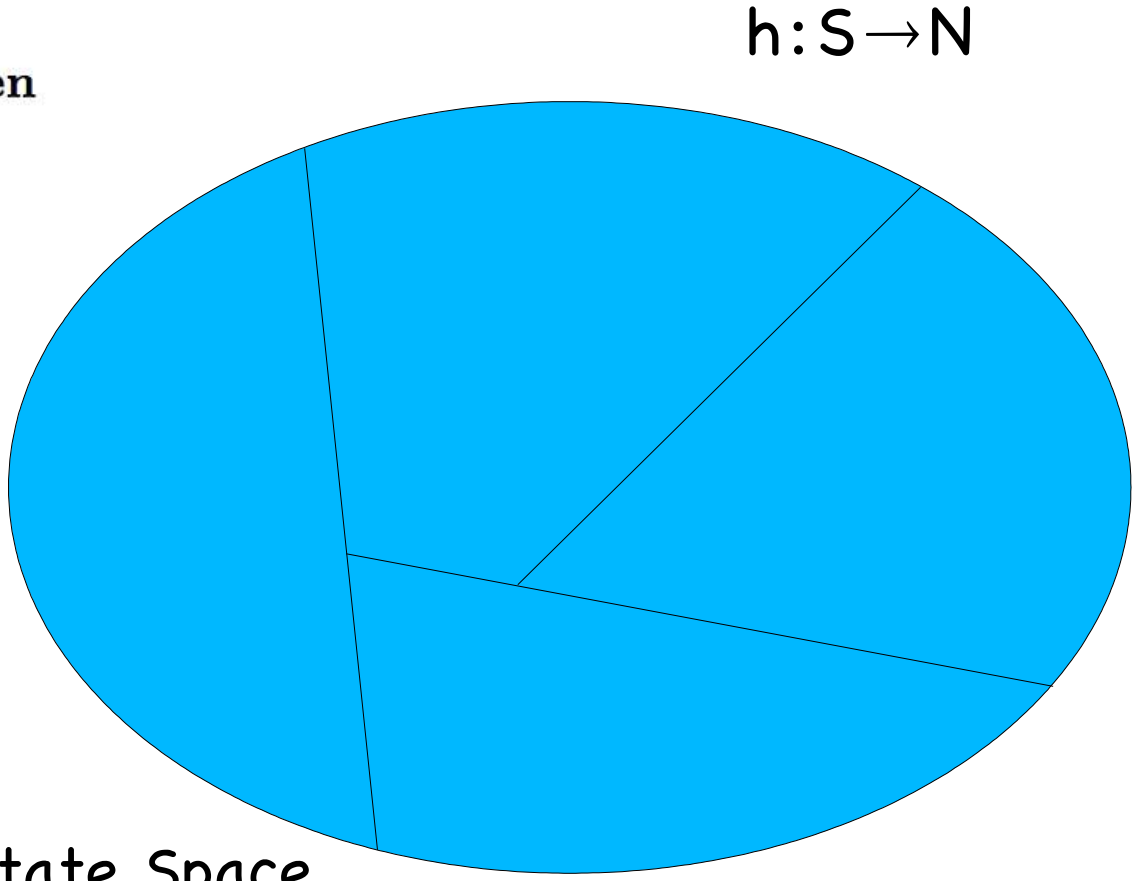
$waiting_A = \{(l_0, Z_0 \wedge I(l_0)) \mid h(l_0) = A\}$
$passed_A = \varnothing$
while $\neg terminated$ **do**
  select and remove a state (l, Z) from $waiting_A$
  if $\forall (l, Y) \in passed_A : Z \not\subseteq Y$ **then**
    $passed_A = passed_A \cup \{(l, Z)\}$
    for all successors (l', Z') of (l, Z) **do**
      $d = h(l')$
      if $\forall (l', Y') \in waiting_d : Z' \not\subseteq Y'$ **then**
        $waiting_d = waiting_d \cup \{(l', Z')\}$
      endif
    done
  endif
done



State Space

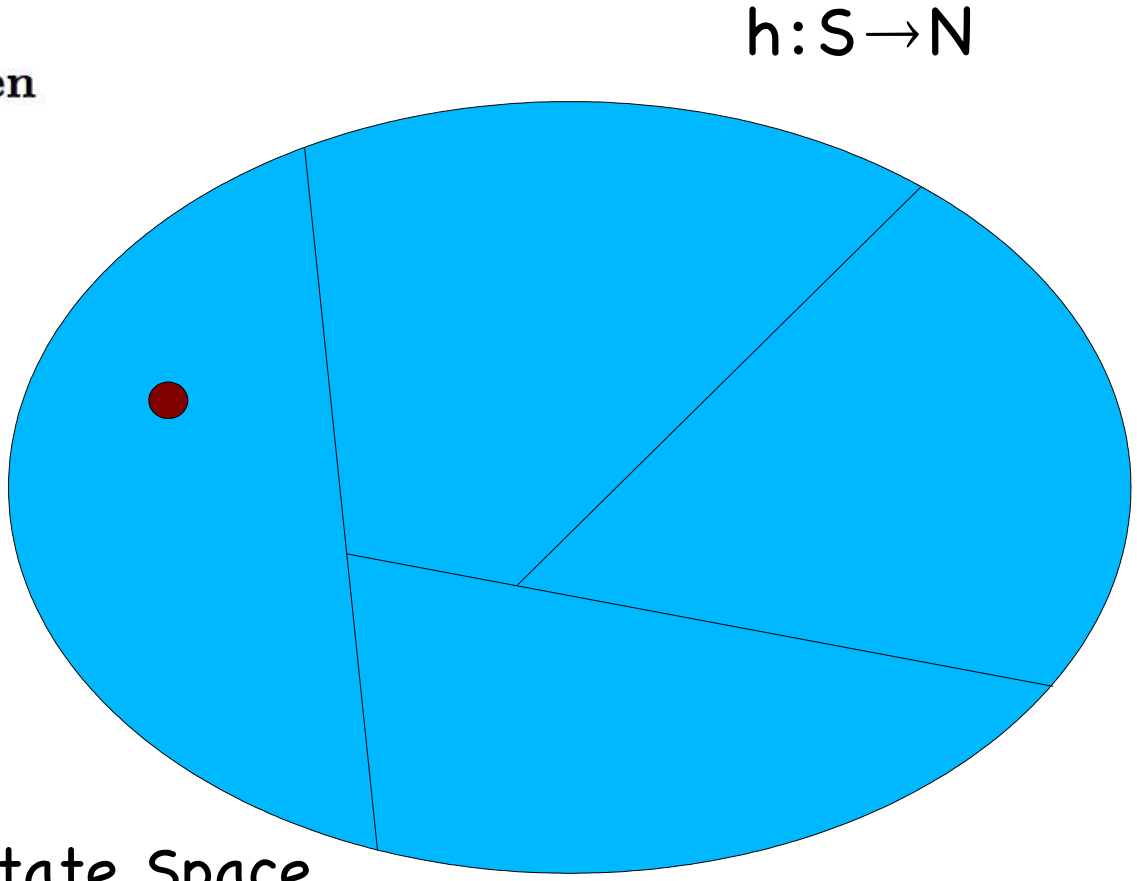
Distributed Reachability

$waiting_A = \{(l_0, Z_0 \wedge I(l_0)) \mid h(l_0) = A\}$
$passed_A = \varnothing$
while $\neg terminated$ **do**
  select and remove a state (l, Z) from $waiting_A$
  if $\forall (l, Y) \in passed_A : Z \not\subseteq Y$ **then**
    $passed_A = passed_A \cup \{(l, Z)\}$
    for all successors (l', Z') of (l, Z) **do**
      $d = h(l')$
      if $\forall (l', Y') \in waiting_d : Z' \not\subseteq Y'$ **then**
        $waiting_d = waiting_d \cup \{(l', Z')\}$
      endif
    done
  endif
done



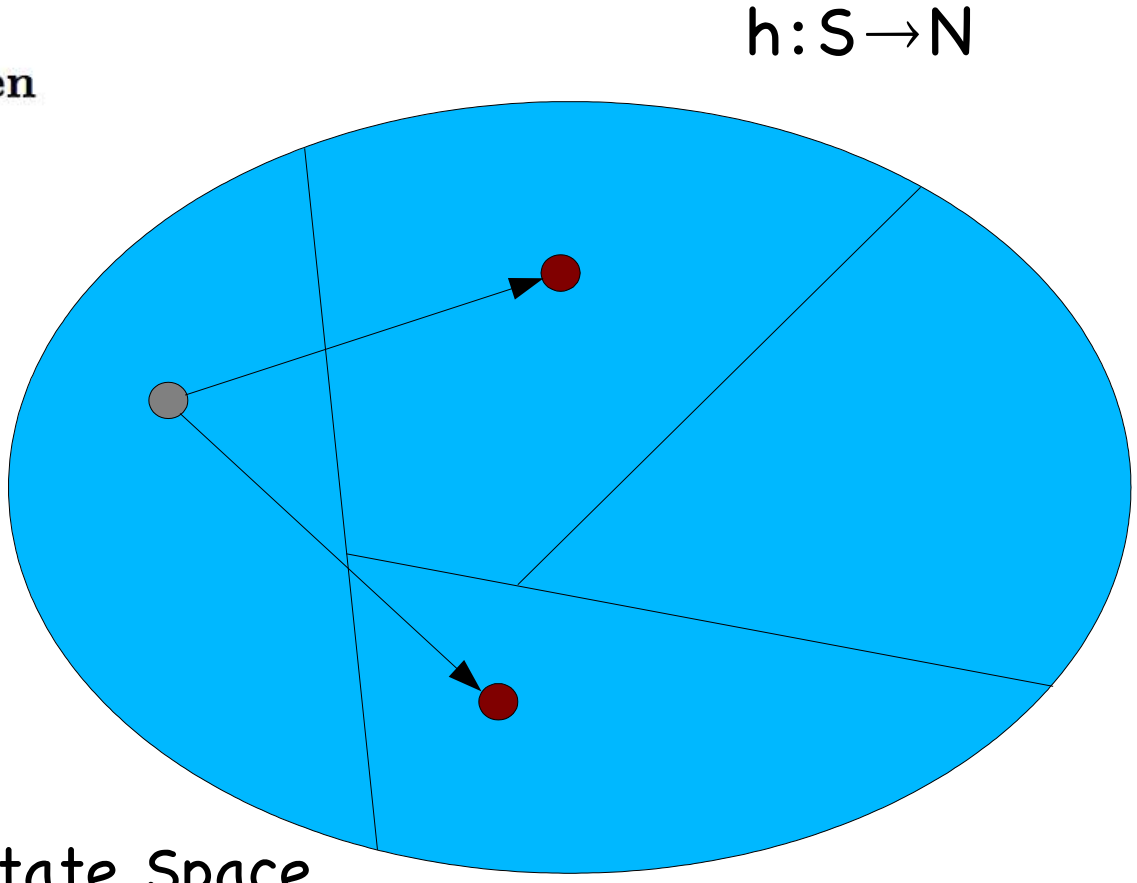
Distributed Reachability

$waiting_A = \{(l_0, Z_0 \wedge I(l_0)) \mid h(l_0) = A\}$
$passed_A = \varnothing$
while $\neg terminated$ **do**
  select and remove a state (l, Z) from $waiting_A$
  if $\forall (l, Y) \in passed_A : Z \not\subseteq Y$ **then**
    $passed_A = passed_A \cup \{(l, Z)\}$
    for all successors (l', Z') of (l, Z) **do**
      $d = h(l')$
      if $\forall (l', Y') \in waiting_d : Z' \not\subseteq Y'$ **then**
        $waiting_d = waiting_d \cup \{(l', Z')\}$
      endif
    done
  endif
done



Distributed Reachability

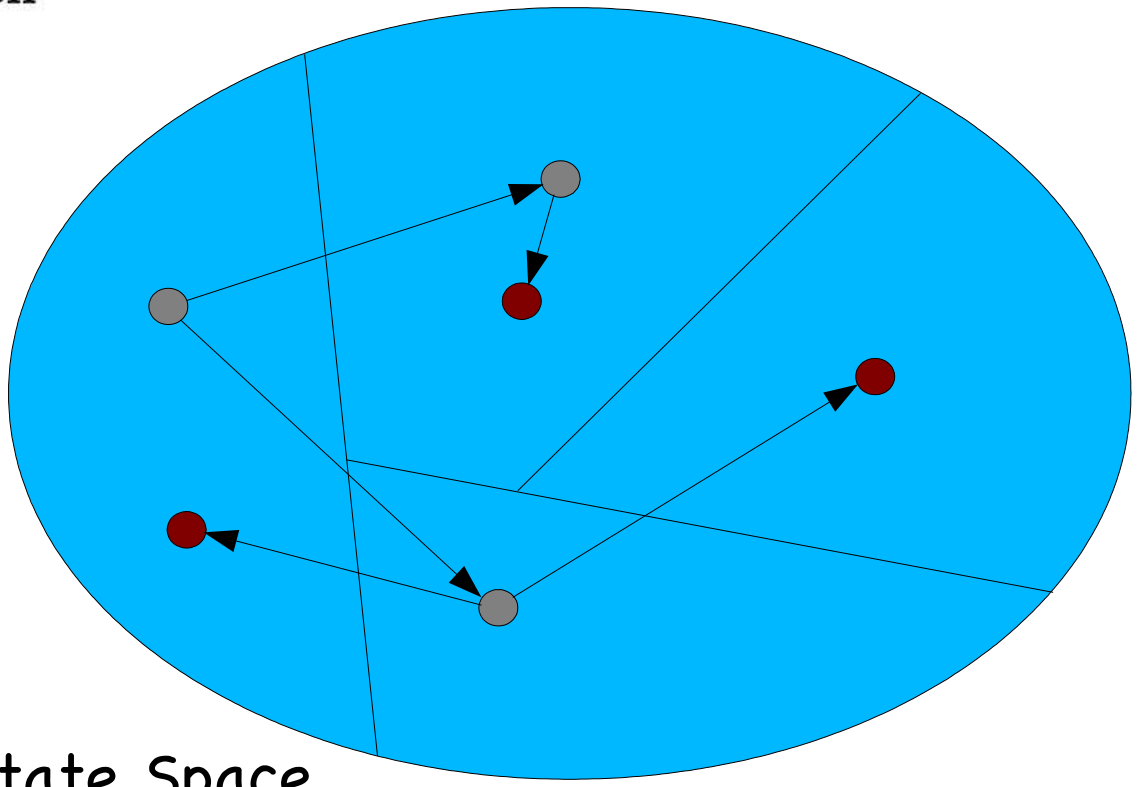
$waiting_A = \{(l_0, Z_0 \wedge I(l_0)) \mid h(l_0) = A\}$
$passed_A = \varnothing$
while $\neg terminated$ **do**
  select and remove a state (l, Z) from $waiting_A$
  if $\forall (l, Y) \in passed_A : Z \not\subseteq Y$ **then**
    $passed_A = passed_A \cup \{(l, Z)\}$
    for all successors (l', Z') of (l, Z) **do**
      $d = h(l')$
      if $\forall (l', Y') \in waiting_d : Z' \not\subseteq Y'$ **then**
        $waiting_d = waiting_d \cup \{(l', Z')\}$
      endif
    done
  endif
done



Distributed Reachability

$waiting_A = \{(l_0, Z_0 \wedge I(l_0)) \mid h(l_0) = A\}$
$passed_A = \varnothing$
while $\neg terminated$ **do**
  select and remove a state (l, Z) from $waiting_A$
  if $\forall (l, Y) \in passed_A : Z \not\subseteq Y$ **then**
    $passed_A = passed_A \cup \{(l, Z)\}$
    for all successors (l', Z') of (l, Z) **do**
      $d = h(l')$
      if $\forall (l', Y') \in waiting_d : Z' \not\subseteq Y'$ **then**
        $waiting_d = waiting_d \cup \{(l', Z')\}$
      endif
    done
  endif
done

h: S → N

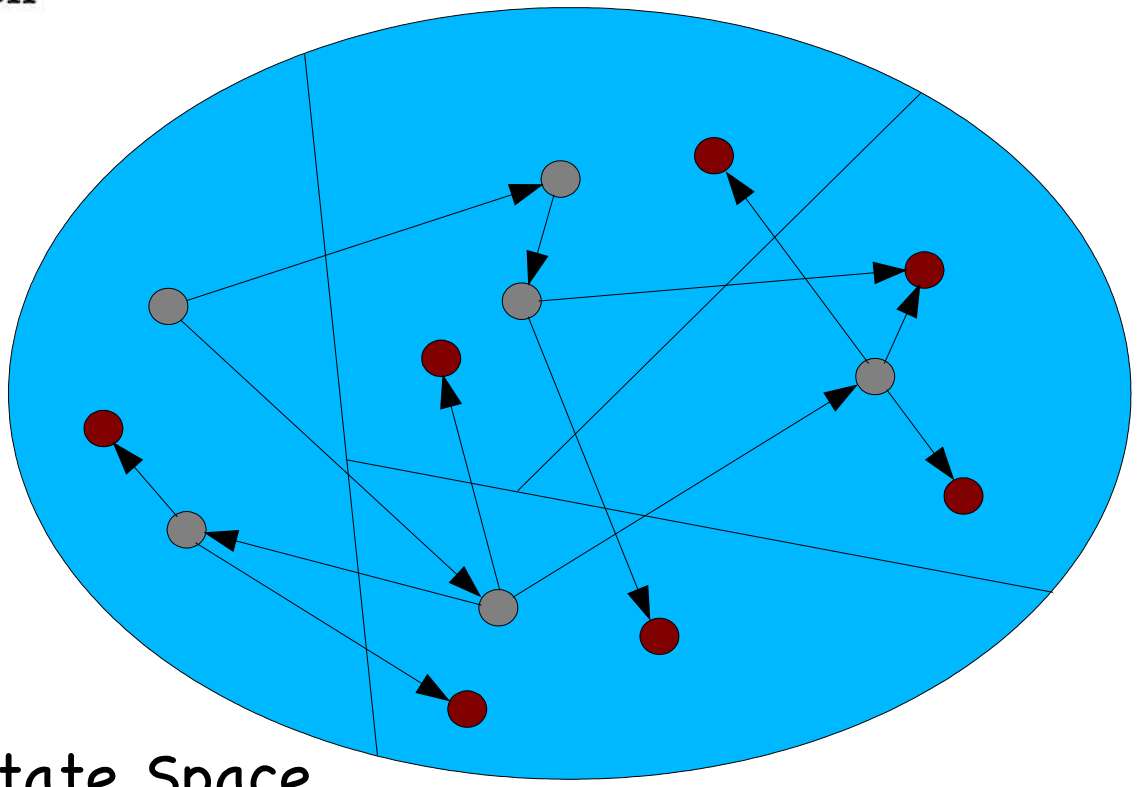


State Space

Distributed Reachability

$waiting_A = \{(l_0, Z_0 \wedge I(l_0)) \mid h(l_0) = A\}$
$passed_A = \varnothing$
while $\neg terminated$ **do**
  select and remove a state (l, Z) from $waiting_A$
  if $\forall (l, Y) \in passed_A : Z \not\subseteq Y$ **then**
    $passed_A = passed_A \cup \{(l, Z)\}$
    for all successors (l', Z') of (l, Z) **do**
      $d = h(l')$
      if $\forall (l', Y') \in waiting_d : Z' \not\subseteq Y'$ **then**
        $waiting_d = waiting_d \cup \{(l', Z')\}$
      endif
    done
  endif
done

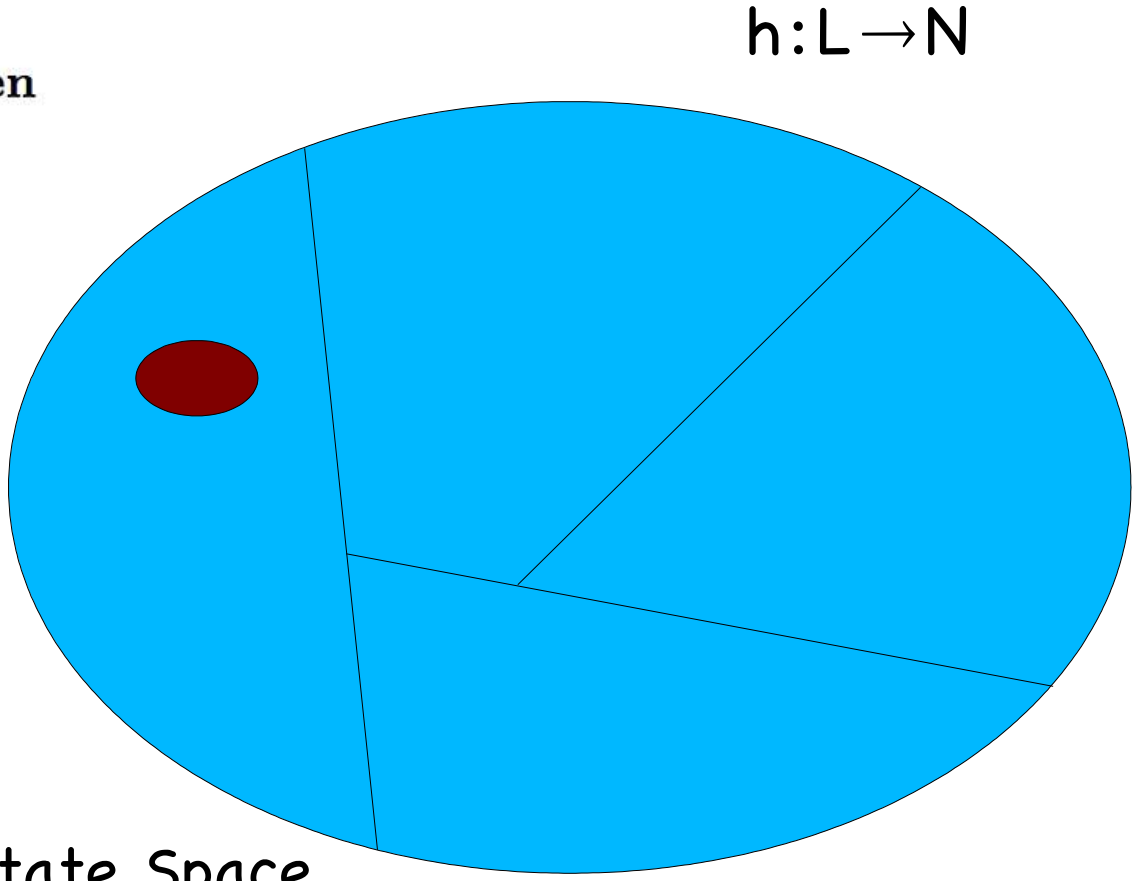
h: S → N



State Space

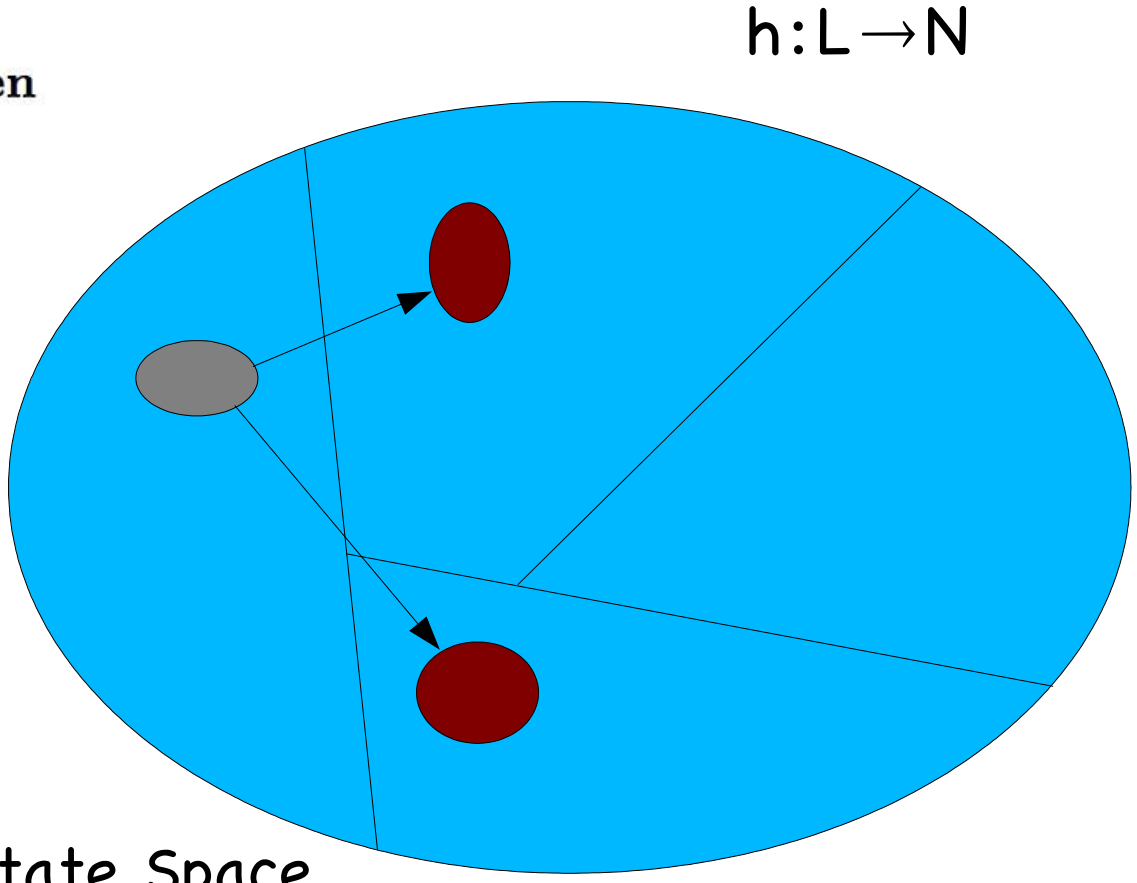
Distributed TA Reachability

$waiting_A = \{(l_0, Z_0 \wedge I(l_0)) \mid h(l_0) = A\}$
$passed_A = \varnothing$
while $\neg terminated$ **do**
  select and remove a state (l, Z) from $waiting_A$
  if $\forall (l, Y) \in passed_A : Z \not\subseteq Y$ **then**
    $passed_A = passed_A \cup \{(l, Z)\}$
    for all successors (l', Z') of (l, Z) **do**
      $d = h(l')$
      if $\forall (l', Y') \in waiting_d : Z' \not\subseteq Y'$ **then**
        $waiting_d = waiting_d \cup \{(l', Z')\}$
      endif
    done
  endif
done



Distributed TA Reachability

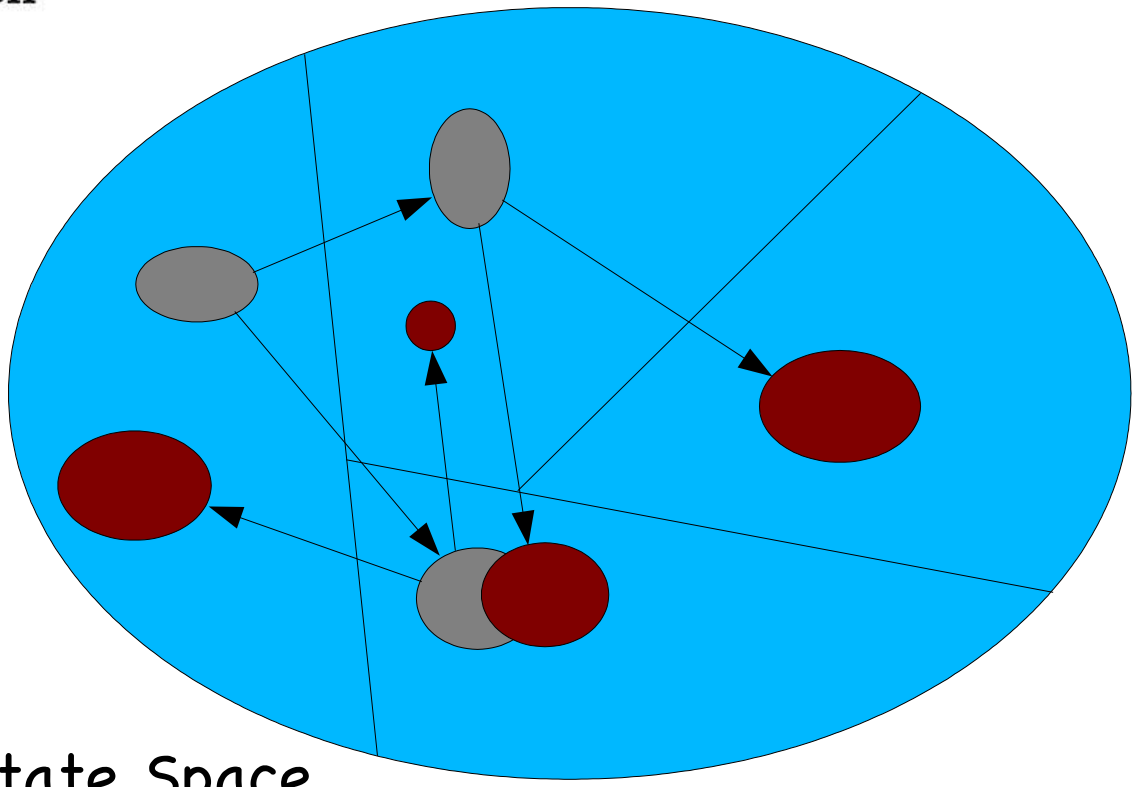
$waiting_A = \{(l_0, Z_0 \wedge I(l_0)) \mid h(l_0) = A\}$
$passed_A = \varnothing$
while $\neg terminated$ **do**
  select and remove a state (l, Z) from $waiting_A$
  if $\forall (l, Y) \in passed_A : Z \not\subseteq Y$ **then**
    $passed_A = passed_A \cup \{(l, Z)\}$
    for all successors (l', Z') of (l, Z) **do**
      $d = h(l')$
      if $\forall (l', Y') \in waiting_d : Z' \not\subseteq Y'$ **then**
        $waiting_d = waiting_d \cup \{(l', Z')\}$
      endif
    done
  endif
done



Distributed TA Reachability

$waiting_A = \{(l_0, Z_0 \wedge I(l_0)) \mid h(l_0) = A\}$
$passed_A = \varnothing$
while $\neg terminated$ **do**
  select and remove a state (l, Z) from $waiting_A$
  if $\forall (l, Y) \in passed_A : Z \not\subseteq Y$ **then**
    $passed_A = passed_A \cup \{(l, Z)\}$
    for all successors (l', Z') of (l, Z) **do**
      $d = h(l')$
      if $\forall (l', Y') \in waiting_d : Z' \not\subseteq Y'$ **then**
        $waiting_d = waiting_d \cup \{(l', Z')\}$
      endif
    done
  endif
done

h:L→N

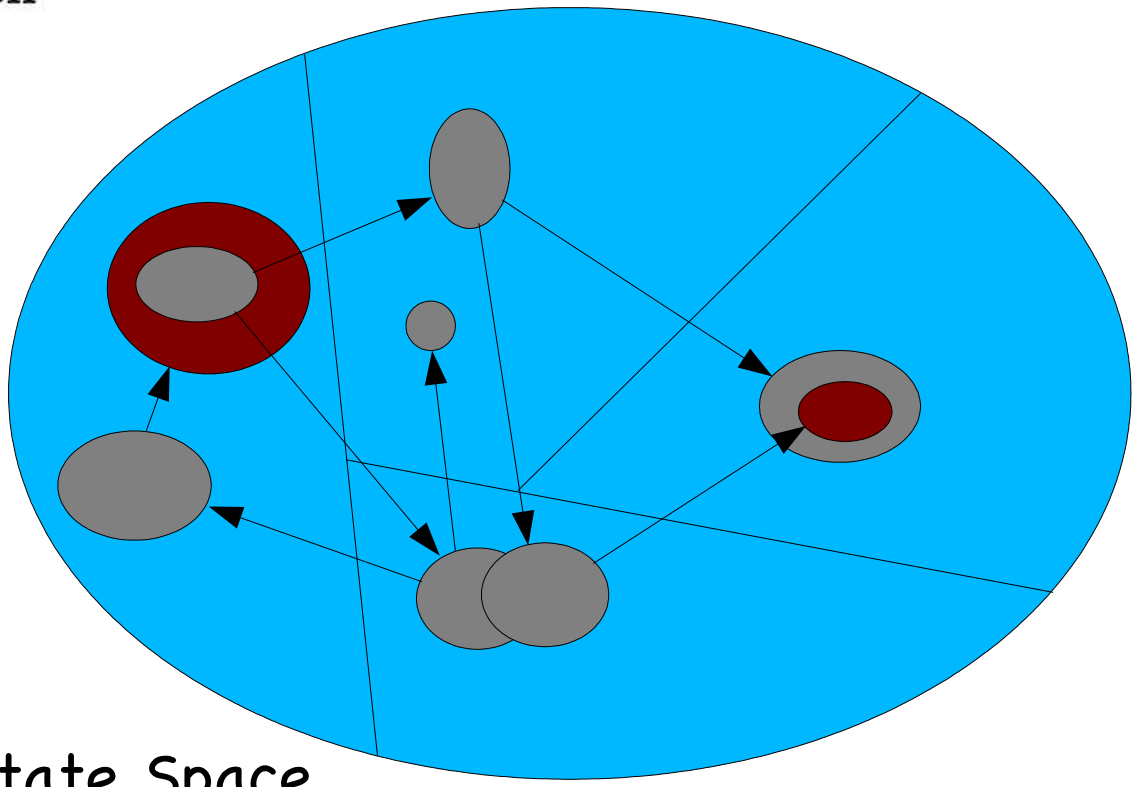


State Space

Distributed TA Reachability

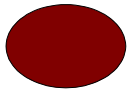
$waiting_A = \{(l_0, Z_0 \wedge I(l_0)) \mid h(l_0) = A\}$
$passed_A = \varnothing$
while $\neg terminated$ **do**
  select and remove a state (l, Z) from $waiting_A$
  if $\forall (l, Y) \in passed_A : Z \not\subseteq Y$ **then**
    $passed_A = passed_A \cup \{(l, Z)\}$
    for all successors (l', Z') of (l, Z) **do**
      $d = h(l')$
      if $\forall (l', Y') \in waiting_d : Z' \not\subseteq Y'$ **then**
        $waiting_d = waiting_d \cup \{(l', Z')\}$
      endif
    done
  endif
done

h:L→N

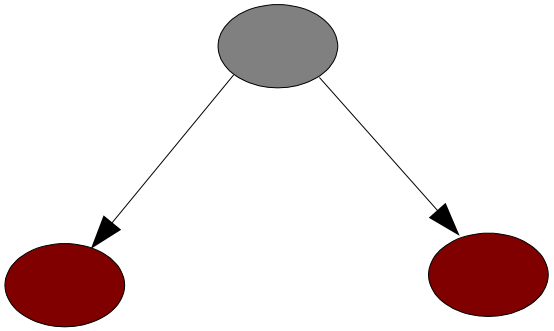


State Space

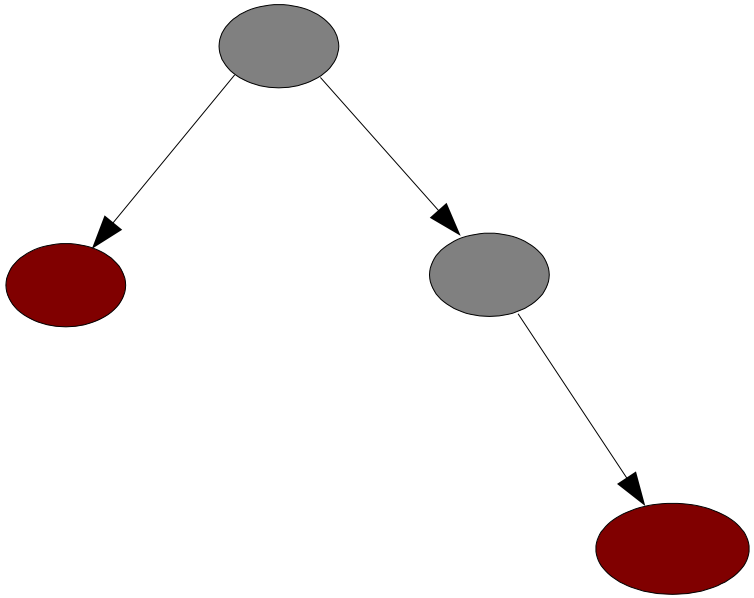
Why the search order matters



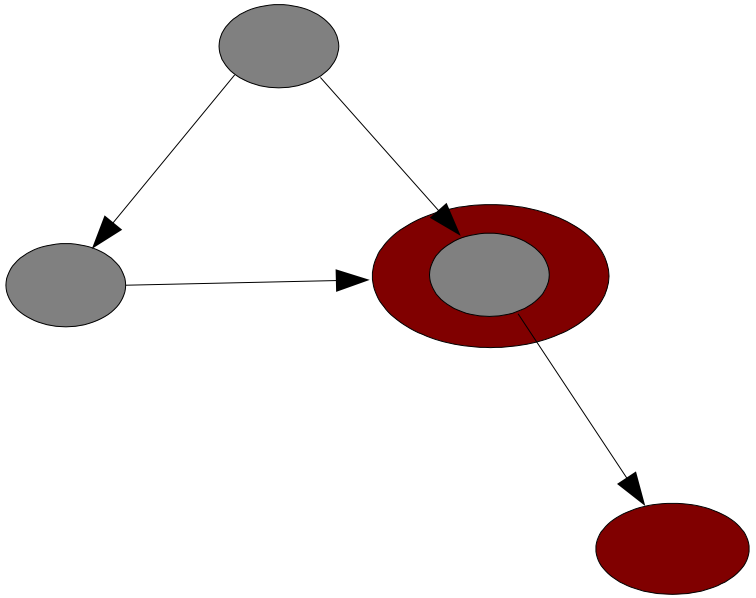
Why the search order matters



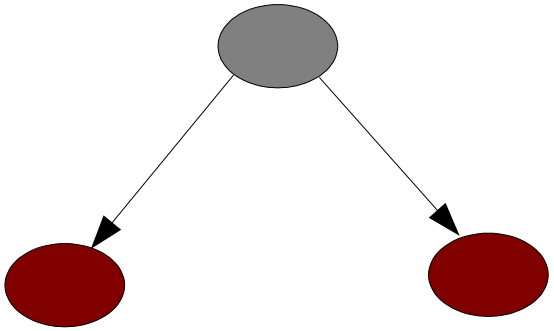
Why the search order matters



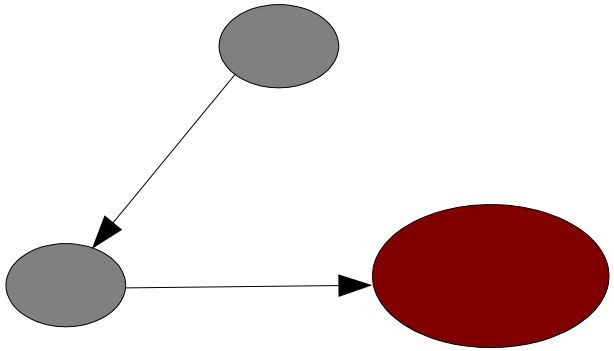
Why the search order matters



Why the search order matters



Why the search order matters



Why the search order matters

Breadth first order is pretty good for TA.

\+ Search order is non-deterministic for distributed reachability.

= The more nodes we add, the more work we get.

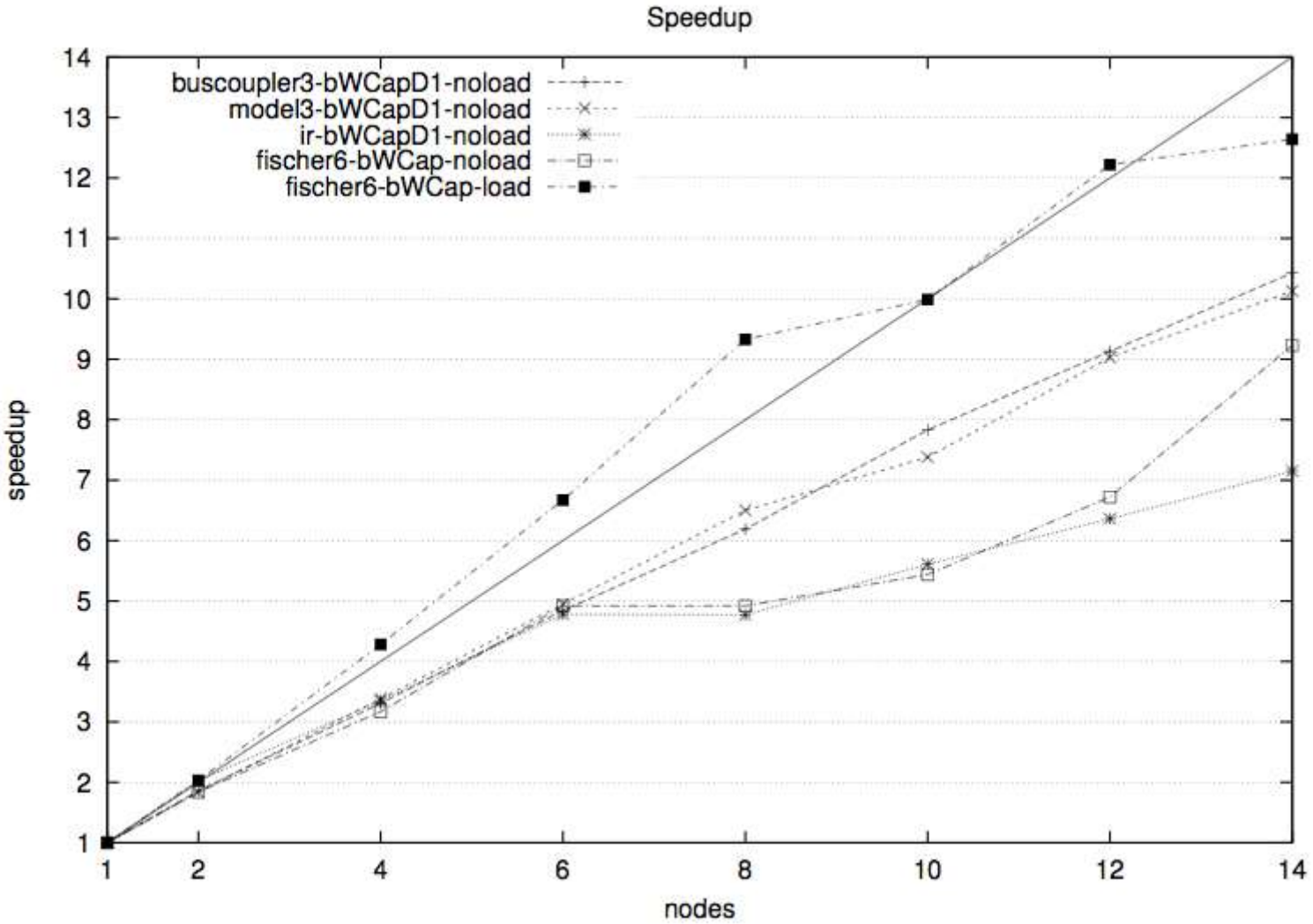
Why the search order matters

Solutions

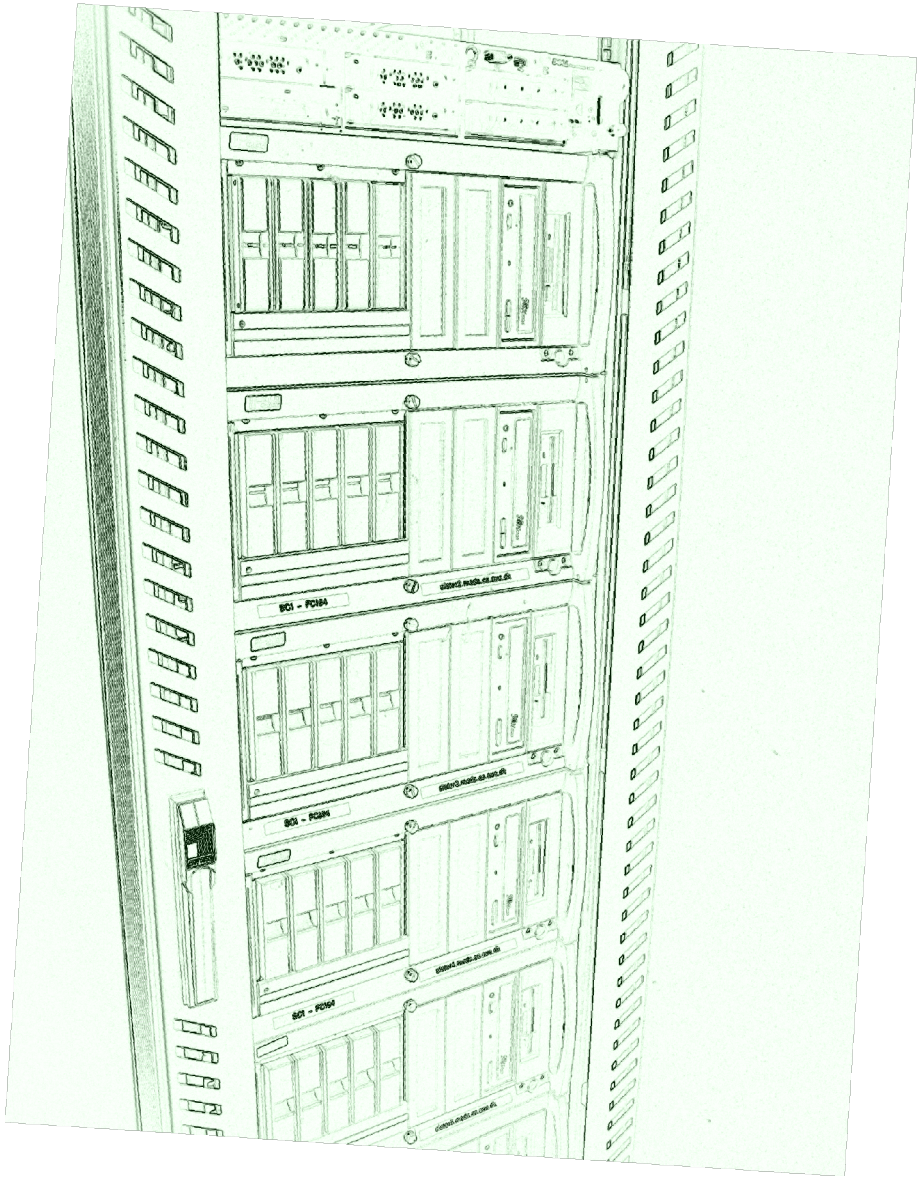
- Locally order states after depth.
- Locally search states (l, Z), where Z is the set of all clock valuations satisfying the invariants of l, first.

- E.g. 3,290,022 -> 5,741,661 with FIFO
  3,290,022 -> 3,021,411 when ordered.

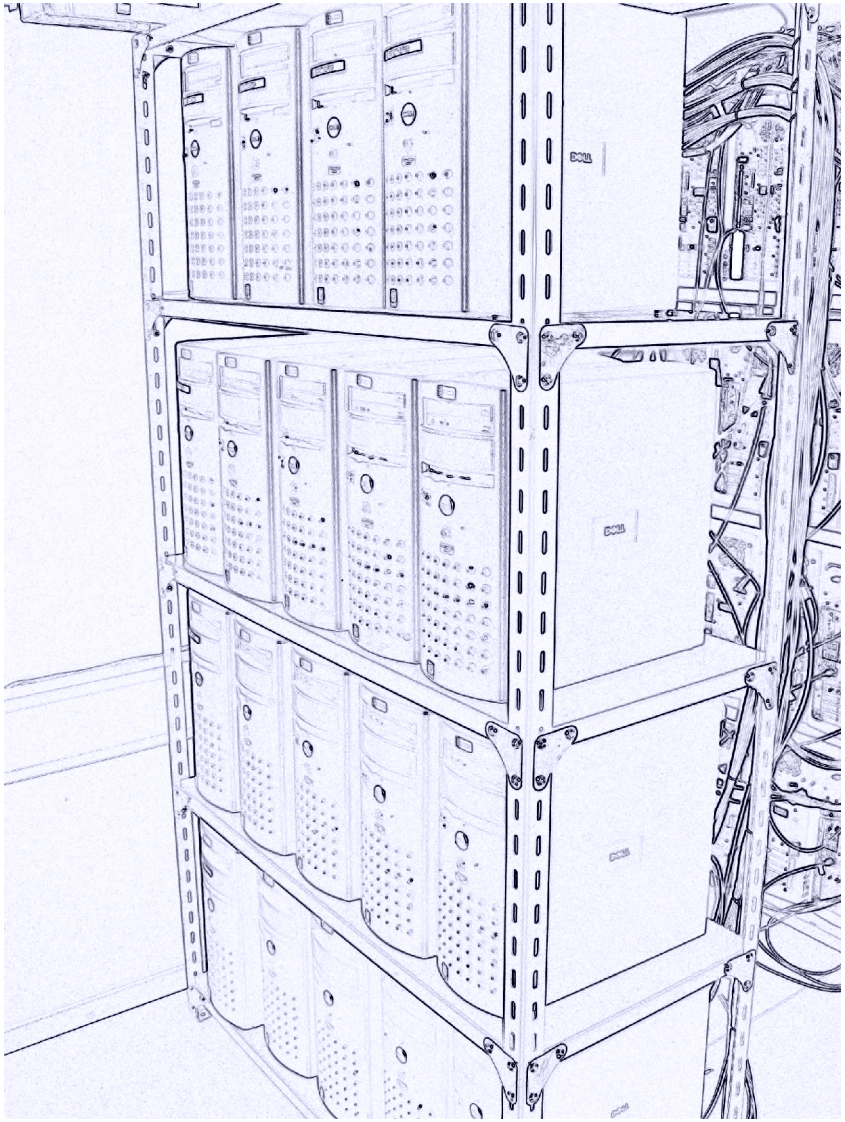
Speedup


Speedup
buscoupler3-bWCapD1-noload
model3-bWCapD1-noload
ir-bWCapD1-noload
fischer6-bWCap-noload
fischer6-bWCap-load
speedup
nodes
1
2
3
4
5
6
7
8
9
10
11
12
13
14

Heterogeneous clusters



+



7 x Dual 733MHz Pentium 3
2GB RAM

36 x 2.8 GHz Xeon Pentium 4
1GB RAM

First approach

Adjust hash function such that the new machines get more states!

First approach

Adjust hash function such that the new machines get more states!

$$\frac{4768\,\text{bogomips}}{1389\,\text{bogomips}} = 3.4$$

Thus we adjust h such that new machines get 3.4 times as many states.

First approach

Hence,

3.4 times the "load", good!

3.4 times as much memory, bad!

$$\frac{1\,\text{GB pr. new CPU}}{3.4} = 295\,\text{MB pr. old CPU}$$

First approach

Hence,

3.4 times the "load", good!

3.4 times as much memory, bad!

$$\frac{1\,\text{GB pr. new CPU}}{3.4} = 295\,\text{MB pr. old CPU}$$

CPU load and memory usage
are inherently linked!

Second approach

computes s
send to h(s)

got s
if new state then
store it

Second approach

computes s
send to h(s)

got s
if new state then
  store it

got reply for s
if reply = new then
   explore s

send back reply

Second approach

computes s
send to h(s)

got s
if new state then
  store it
  should I lie?
send back reply

got reply for s
if reply = new then
  explore s

if new and I lied then
  explore s

Second approach

computes s
send to h(s)

$\ldots s_4, s_3, s_2, s_1$ →

got s
if new state then
  store it
  should I lie?

0110... ←

got reply for s
if reply = new then
  explore s

send back reply
if new and I lied then
  explore s

When to lie

Depends on several factors

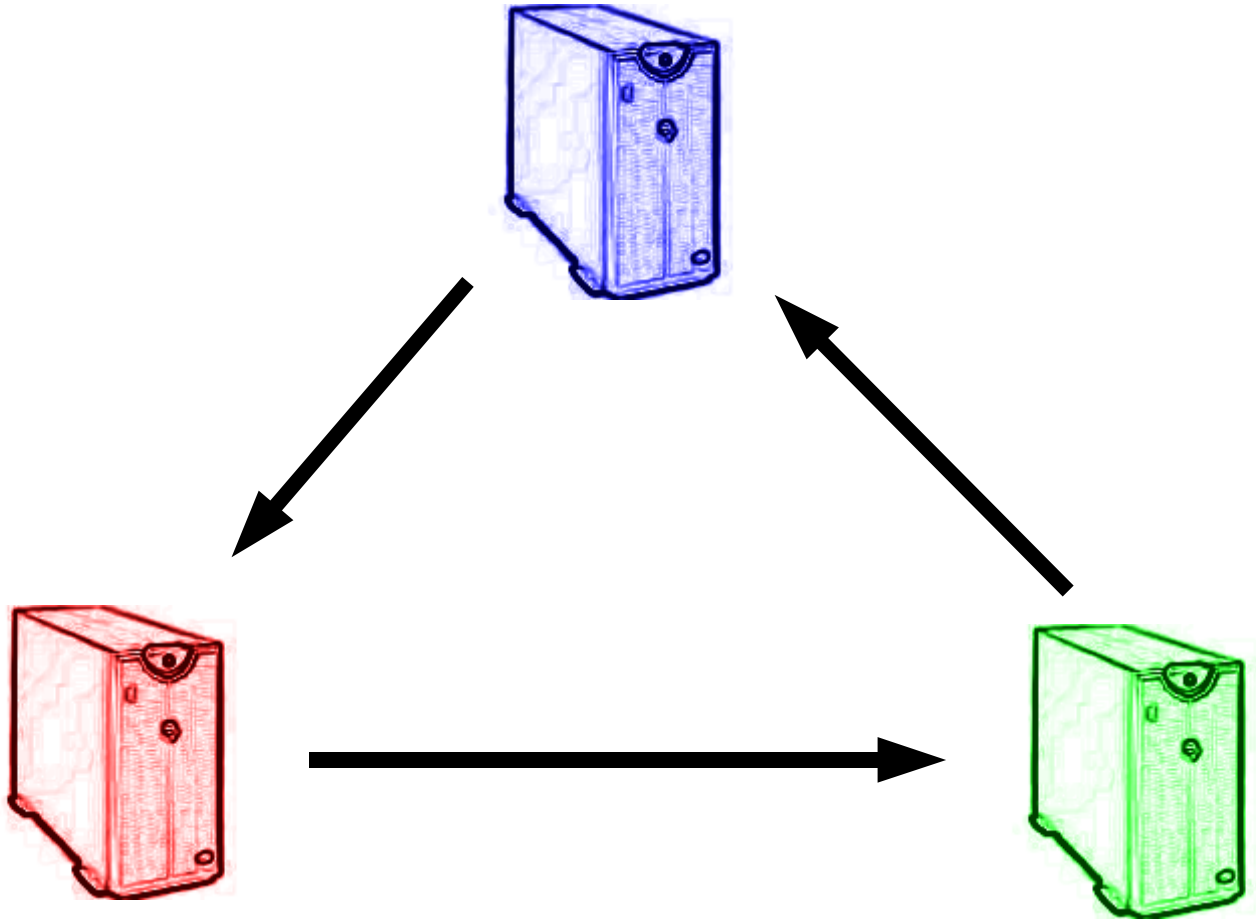
- The current load
- The current exploration rate

of myself, my peer, and all other nodes.

$$\forall i, j: \frac{|W_i|}{|R_i|} = \frac{|W_j|}{|R_j|}$$

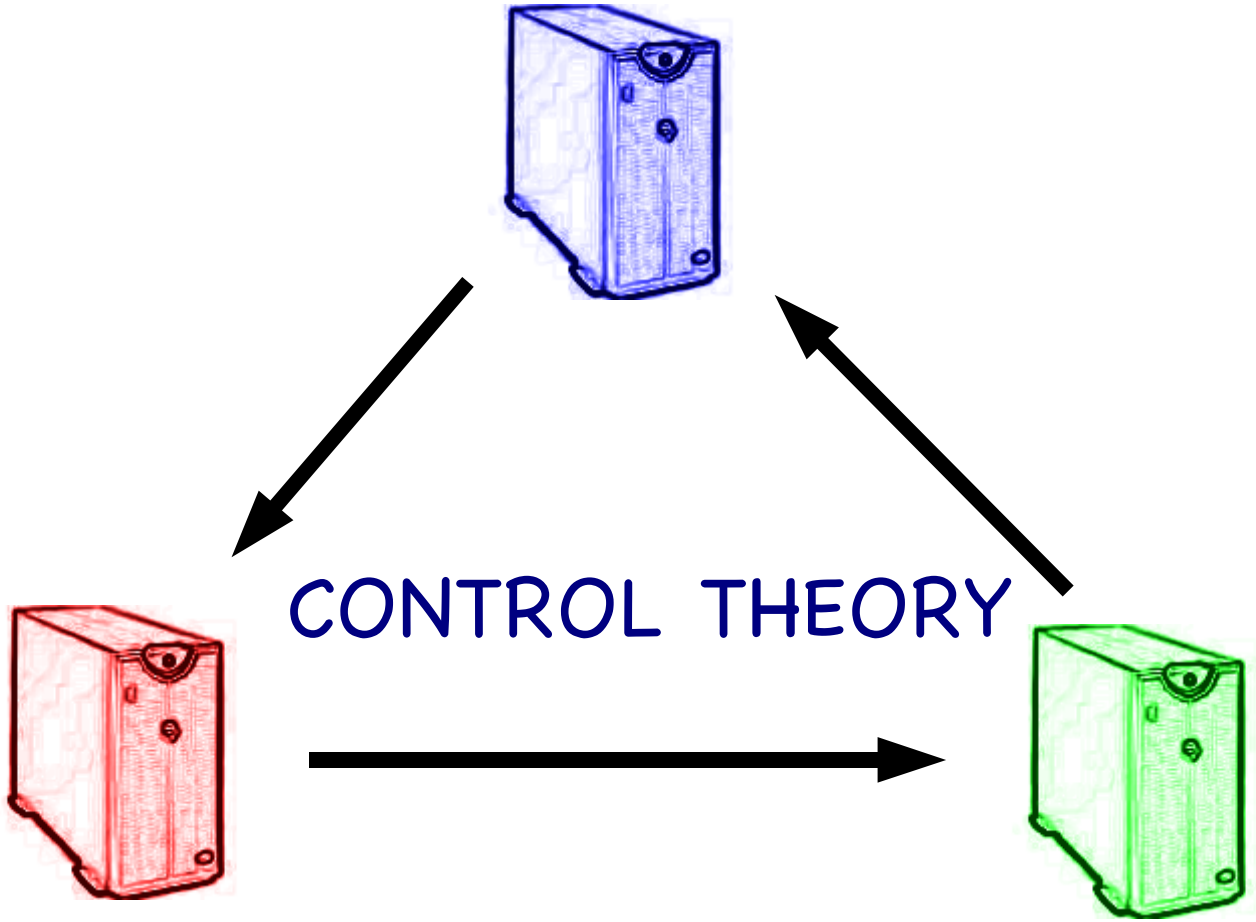
Other factors

Is the system stable or does it oscillate?



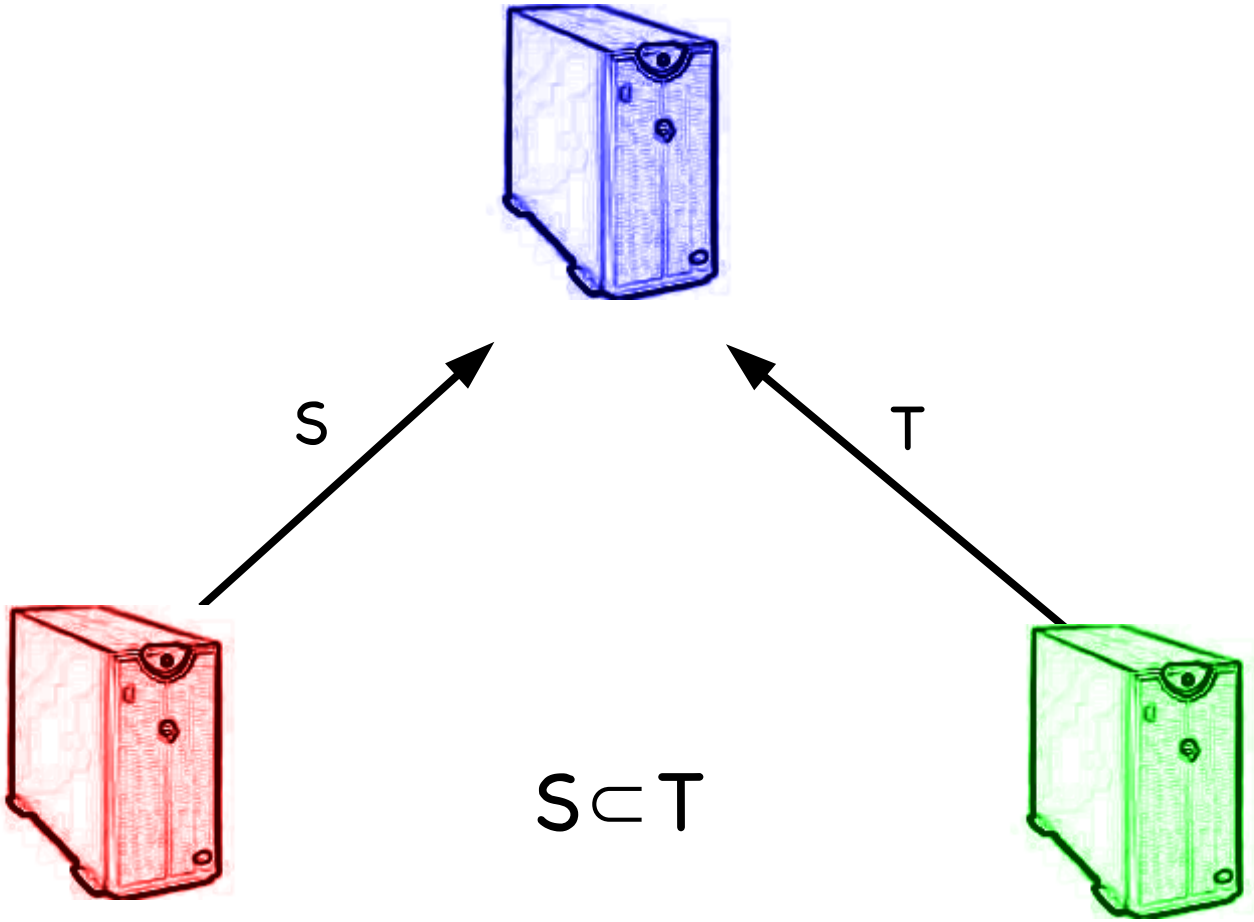
Other factors

Is the system stable or does it oscillate?



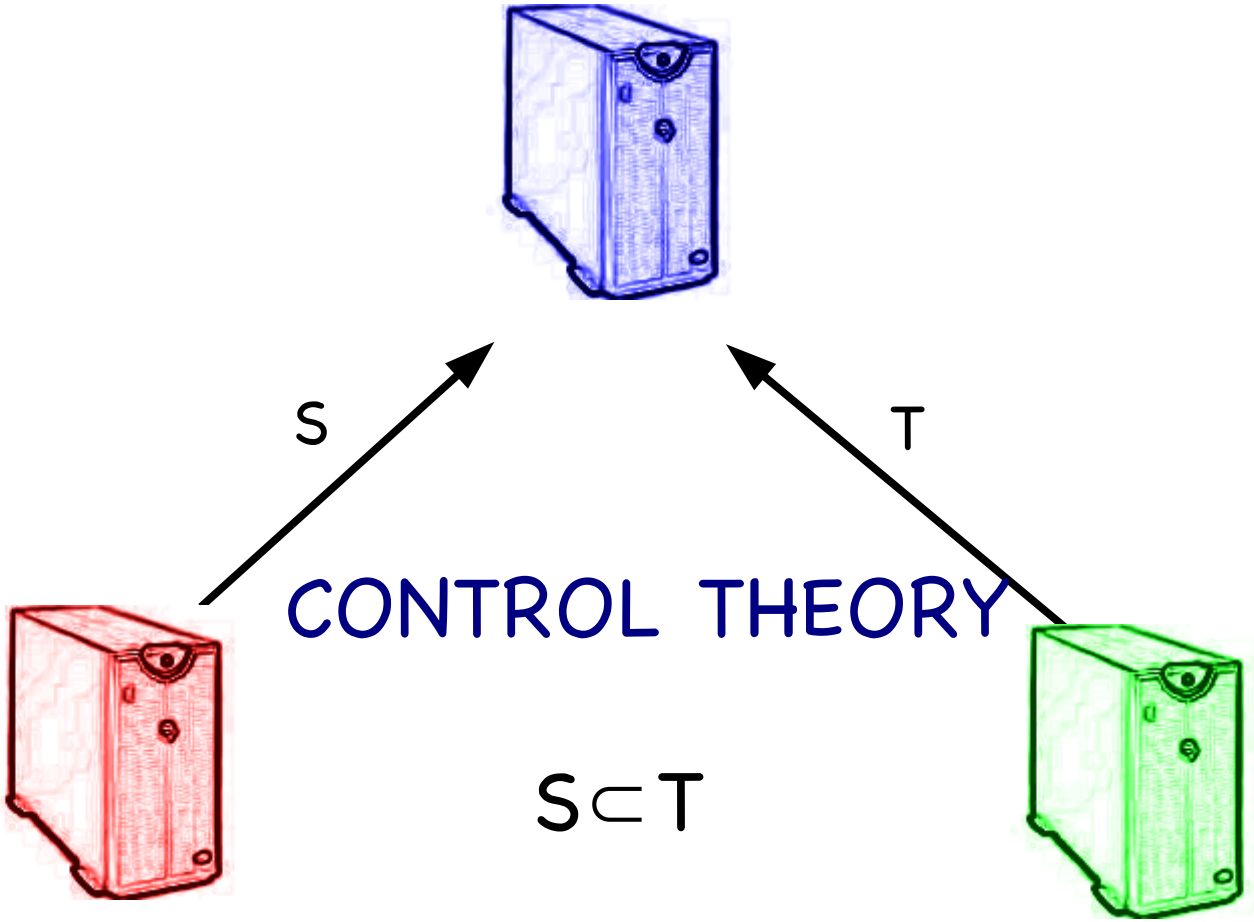
Other factors

With symbolic states, we would rather steal.

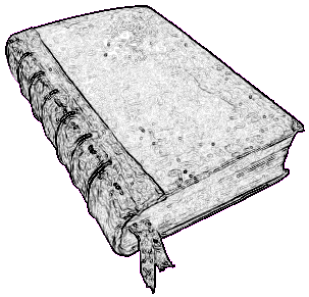


Other factors

With symbolic states, we would rather steal.



The controller

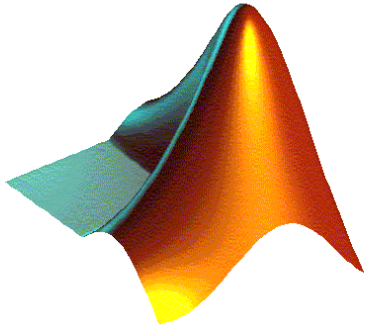


Cluster configuration

+

Henrik Schiøler

+

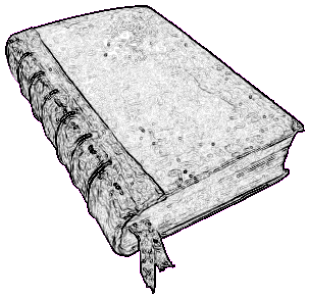


Matlab

=

Controller

The controller

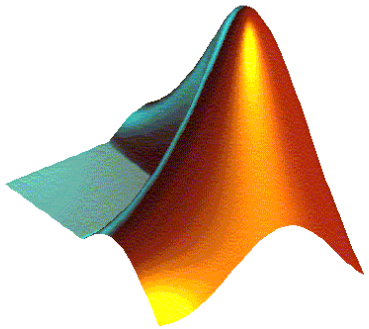


Cluster configuration

+

Henrik Schiøler

+



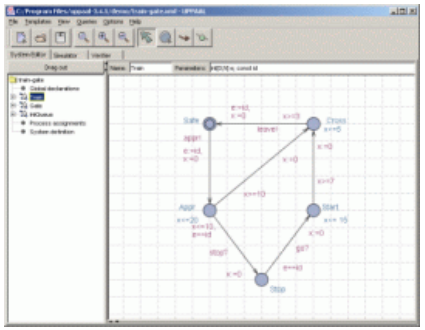
Matlab

=

Controller

Controller

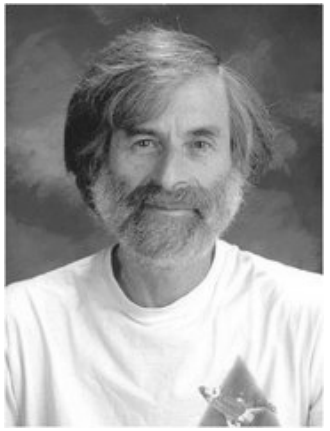
+



UPPAAL

=

&

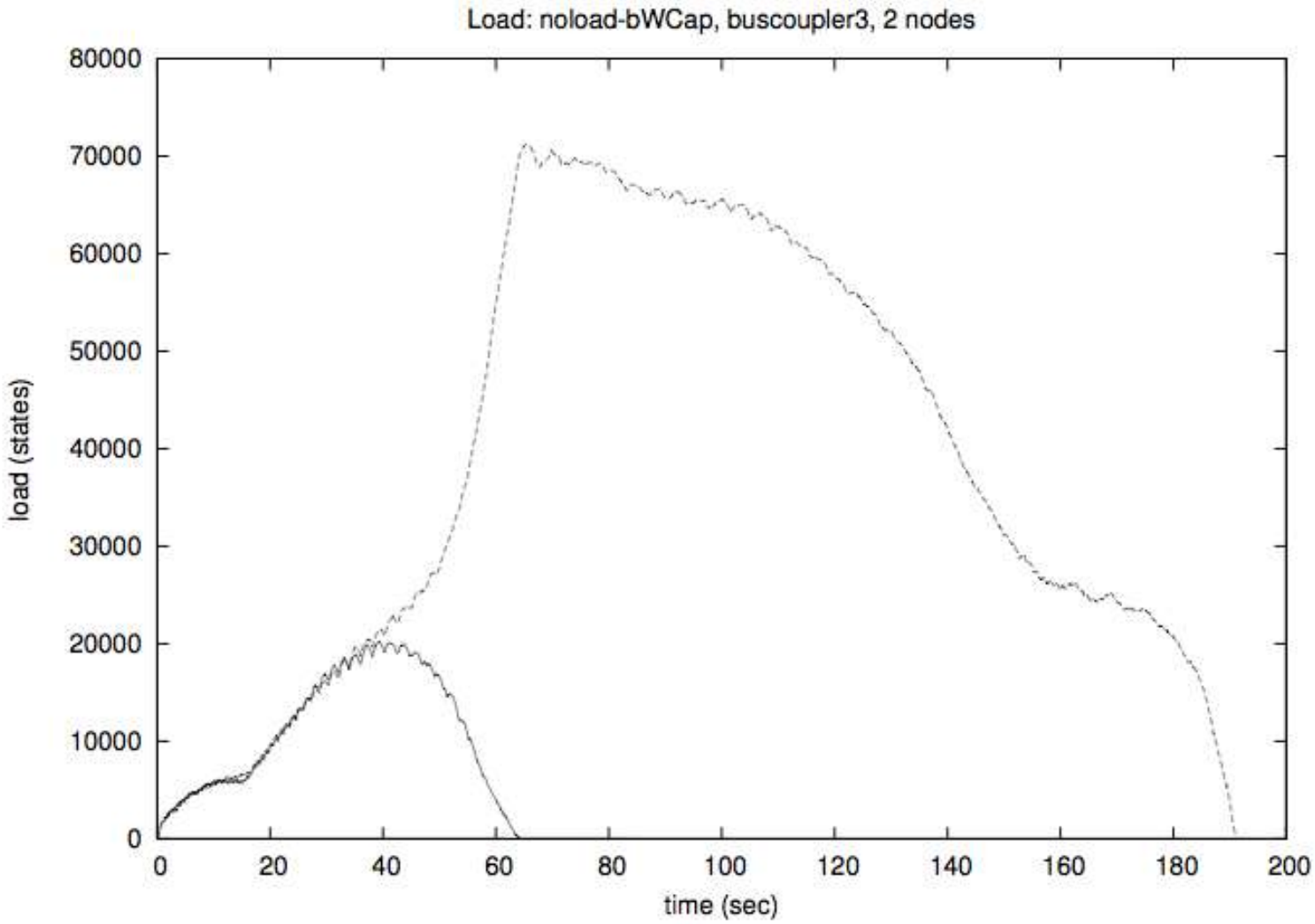


Happy users

Homogeneous clusters

The traditional algorithm is an instance of the new algorithm, where all states are stolen.

Load balancing homogeneous clusters



PDMC 2002,
STTT 2003

Load balancing homogeneous clusters

- Was thought to be TA specific, but
- Similar effects have been observed by
  - Kumar and Mercer, PDMC 2004
  - Jiri Barnat
- Why and why now?

Load balancing homogeneous clusters

$$\dot{explored}_1 = f(CPU_1, \tfrac{1}{2}\dot{gen}_2, |Wait_1|)$$

$$\dot{gen}_1 = f'(\dot{explored}_1)$$

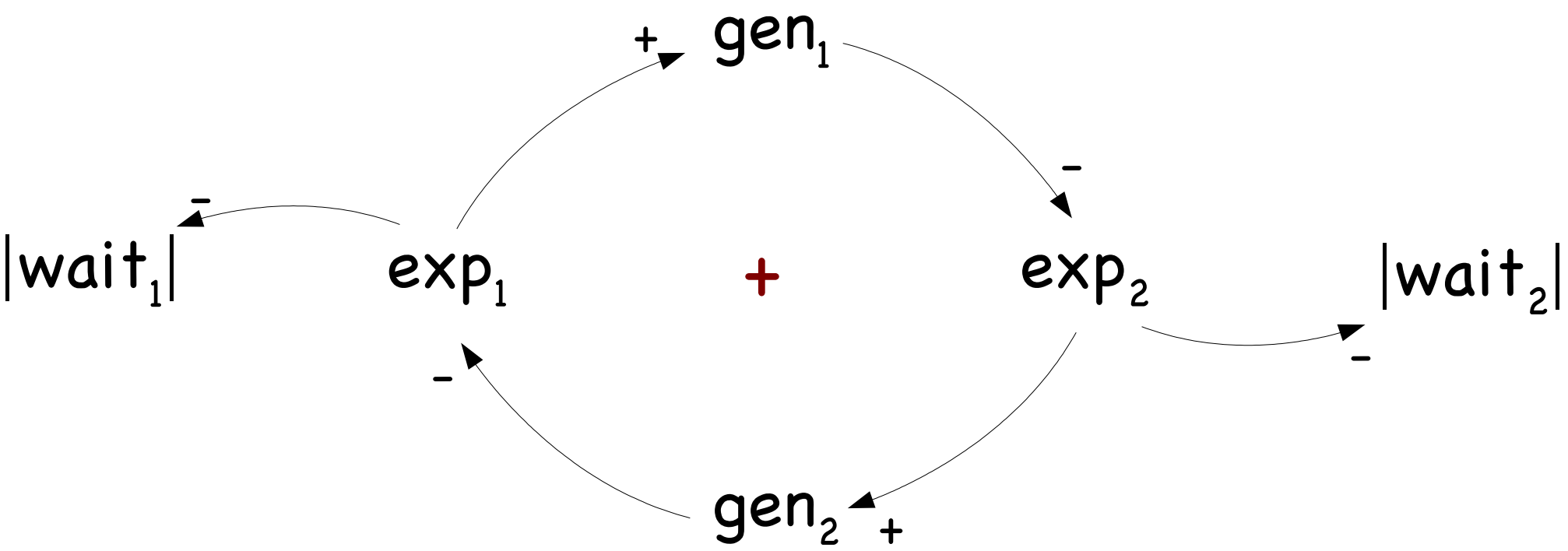
$$|\dot{Wait}_1| = \tfrac{1}{2}(\dot{gen}_1 + \dot{gen}_2) - \dot{exp}_1$$

$$\dot{explored}_2 = f(CPU_2, \tfrac{1}{2}\dot{gen}_1, |Wait_2|)$$

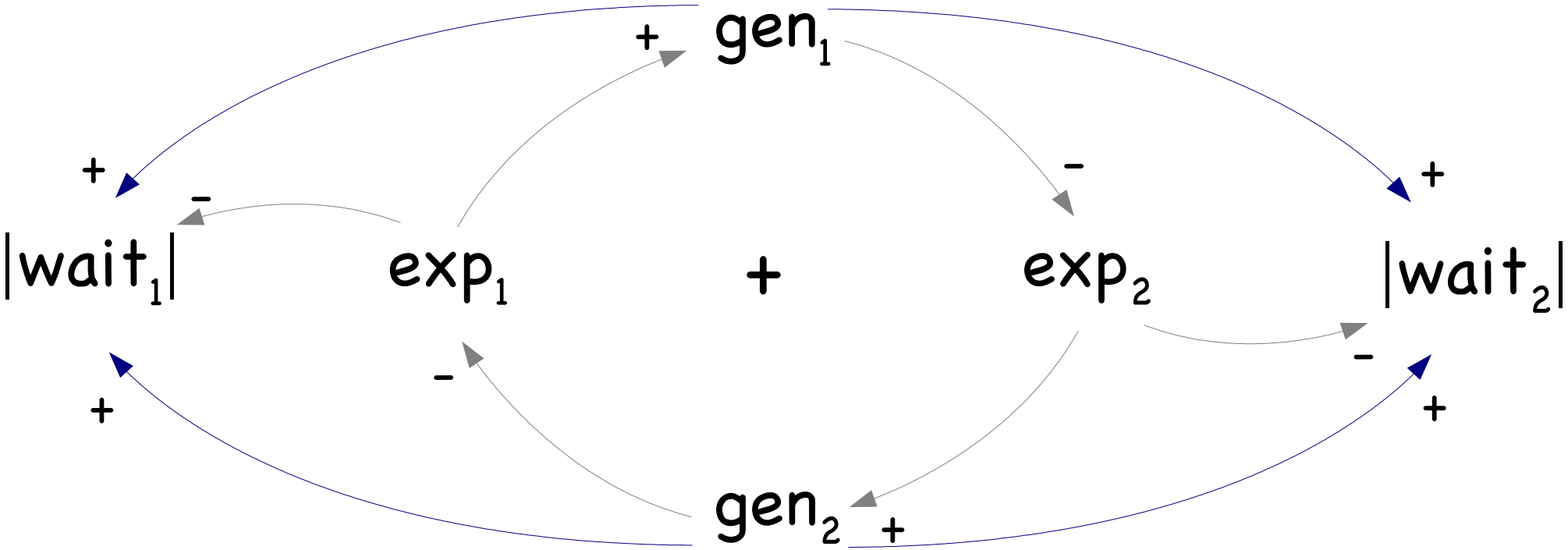
$$\dot{gen}_2 = f'(\dot{explored}_2)$$

$$|\dot{Wait}_2| = \tfrac{1}{2}(\dot{gen}_1 + \dot{gen}_2) - \dot{exp}_2$$

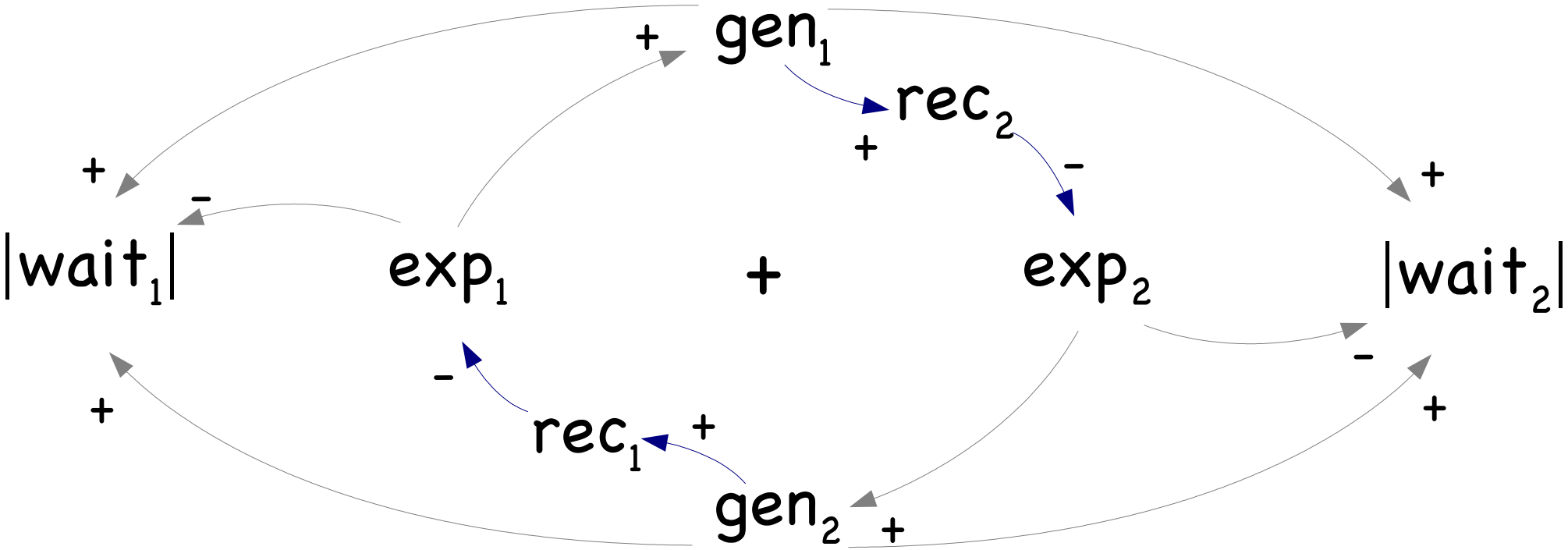
Positive feedback loop



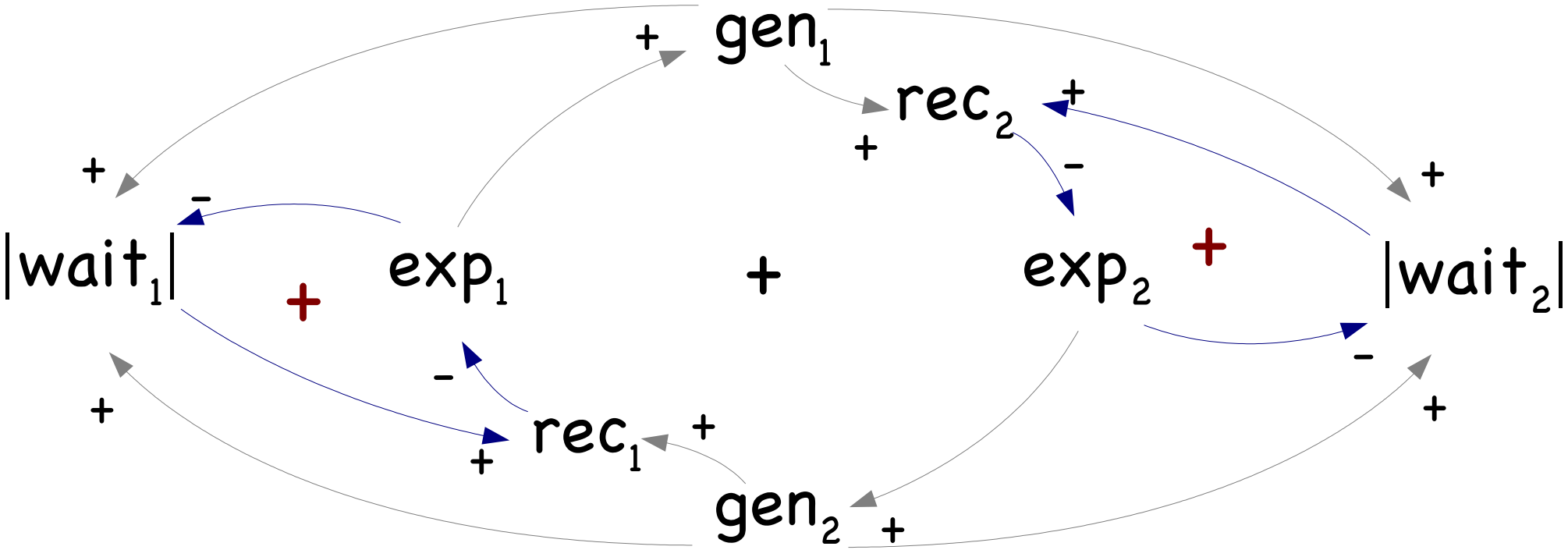
Positive feedback loop



Positive feedback loop



Positive feedback loop



What changed since the early days?

- Cluster of workstations rather than parallel machines.
- CPU speed
- Network bandwidth
- Network latency

Lesson learned?

- Problems related to control theory and systems dynamics.
- We must analyse the stability of our systems.

- The load balancing scheme for the heterogenous setup seems to work very well for the homogeneous setup.