

Illustration of ISP for each bug-class

MPI Happens-before: how MPI supports Out-of-order execution

The POE algorithm of ISP explained using MPI happens-before

About 30 minutes - by Ganesh

Bug Classes Caught by ISP

- Deadlocks
 - Show how a collective deadlock is caught
- Resource Leaks
 - Demonstrate MPI_hb_different_comm that shows
 - Coloration in Java GUI depends on communicators
 - The relaxed HB edges
 - None between different sends from the same process
 - » They target the same destination
 - » but use different communicators
- Buffer Sensitive Deadlocks
 - Run MPI_BufferingSensitiveDeadlock
 - With Windows->Preferences->ISP->Use Blocking Sends = true
 - With the above = false
- Assertion violation
 - Run MPI_AssertTest (red_blue_assert.c) with 4 procs.

Intuitive GUI Display Capabilities

- We have already seen the Internal Issue Order
- Witness the experimental Time Order stepping feature
- Iprobes are interesting!
 - We can probe one send / receive
 - But, we can actually communicate with a different send / receive
 - The ISP GUI shows this really vividly!
 - Try out `MPI_Iprobeillustration`, file `i-probe-illustration.c`
- ISP's execution is directly guided by `MPI-hb`
 - `MPI_HappensBeforeillustration`
- ISP is very mindful of wildcard dependencies that may arise due to continued execution
 - Try `MPI_CrossCoupledWildcardDependency`
 - Discuss our “lazy algorithm” for handling this (not in ISP now)

Formalization of MPI Happens-before

It really is

matches-before + completes-before

MPI guarantees point-to-point non-overtaking

P0

S(to:1, msg1, h1);

...

S(to:1, msg2, h2);

...

W(h1);

...

W(h2);

P1

R(from:0, buf1, h3);

...

R(from:0, buf2, h4);

...

W(h3);

...

W(h4);

MPI guarantees point-to-point non-overtaking

Require S(..msg1..) to match a potential receive from P1

BEFORE S(..msg2..) is allowed to match

P0

S(to:1, msg1, h1);

...

S(to:1, msg2, h2);

...

W(h1);

...

W(h2);

P1

R(from:0, buf1, h3);

...

R(from:0, buf2, h4);

...

W(h3);

...

W(h4);

MPI guarantees point-to-point non-overtaking

Require S(..msg1..) to match a potential receive from P1

BEFORE S(..msg2..) is allowed to match

P0	P1
---	---
S(to:1, msg1, h1);	R(from:0, buf1, h3);
...	...
S(to:1, msg2, h2);	R(from:0, buf2, h4);
...	...
W(h1);	W(h3);
...	...
W(h2);	W(h4);

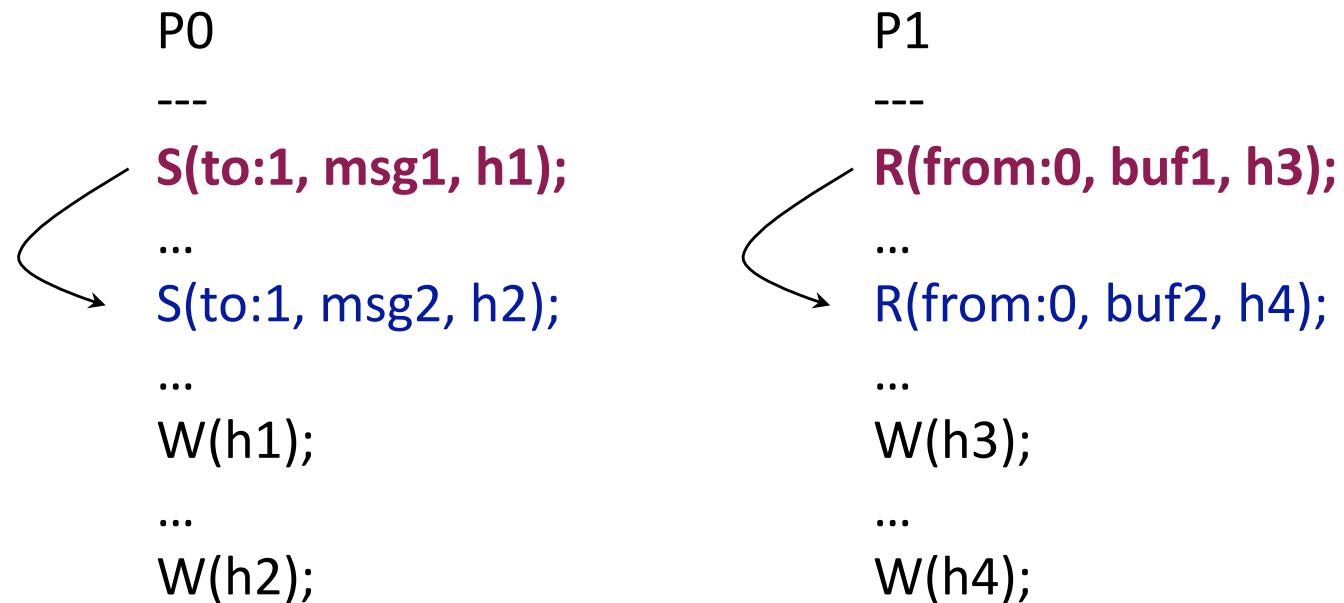
Require R(..buf1..) to match a potential send from P0

BEFORE R(..buf2..) is allowed to match

MPI guarantees point-to-point non-overtaking

Require S(..msg1..) to match a potential receive from P1

BEFORE S(..msg2..) is allowed to match



Require R(..buf1..) to match a potential send from P0

BEFORE R(..buf2..) is allowed to match

Achieved by ensuring the above matches-before relations during scheduling

But, do not enforce matches-before needlessly

Must we force sends (S) to finish in order in all cases ?

Certainly not! (unless you love poor performance)

P0	P1	P1
---	---	---
S(to:1, big-message, h1);	R(from:1, buf1, h3);	R(from:2, buf2, h4);
...
S(to:2, small-message, h2);	W(h3);	W(h4);
...		
W(h2);		
...		
W(h1);		

But, do not enforce matches-before needlessly

Must we force sends (S) to finish in order in all cases ?

Certainly not! (unless you love poor performance)

P0	P1	P1
---	---	---
S(to:1, big-message, h1);	R(from:1, buf1, h3);	R(from:2, buf2, h4);
...
S(to:2, small-message, h2);	W(h3);	W(h4);
...		
W(h2);		
...		
W(h1);		

Non-blocking sends (S) allow later actions of the same process (including “unrelated” non-blocking sends) to be concurrent.

But, do not enforce matches-before needlessly

Must we force sends (S) to finish in order in all cases ?

Certainly not! (unless you love poor performance)

P0	P1	P1
---	---	---
S(to:1, big-message, h1);	R(from:1, buf1, h3);	R(from:2, buf2, h4);
...
S(to:2, small-message, h2);	W(h3);	W(h4);
...		
W(h2);		
...		
W(h1);		

Non-blocking sends (S) allow later actions of the same process (including “unrelated” non-blocking sends) to be concurrent.

Achieved by NOT having matches-before between the S within P0

MPI is tricky... till you see how it really works!

Will this single-process example called “Auto-send” deadlock ?

P0 : R(from:0, h1); B; S(to:0, h2); W(h1); W(h2);

MPI is tricky... till you see how it really works!

Will this single-process example called “Auto-send” deadlock ?

P0 : R(from:0, h1); B; S(to:0, h2); W(h1); W(h2);

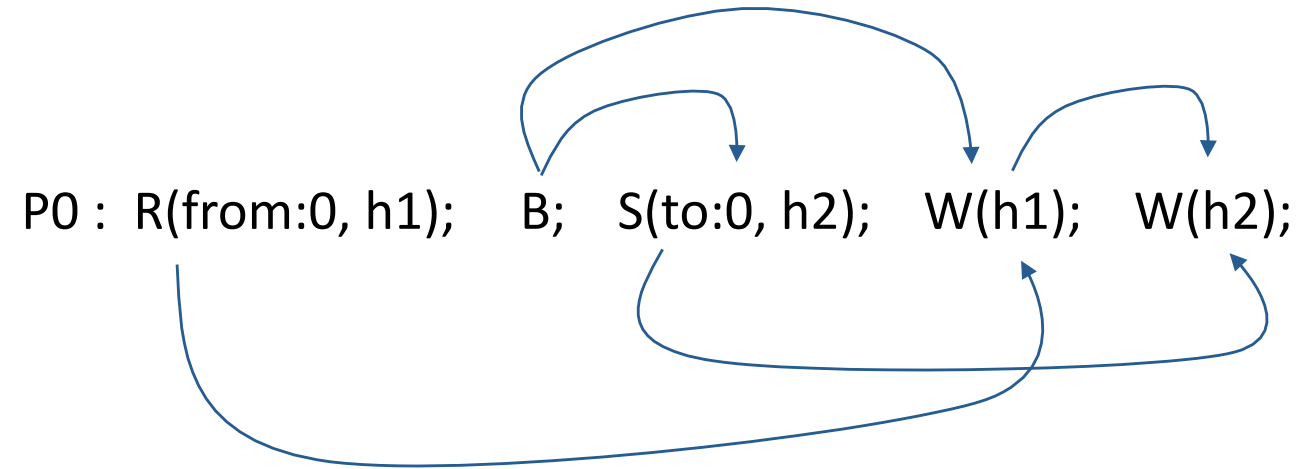
Again no! R (non-blocking receive) only initiates receipt – likewise non-blocking send (S) only initiates sending. These activities go on concurrently within the same process.

How Example Auto-send works

P0 : R(from:0, h1); B; S(to:0, h2); W(h1); W(h2);

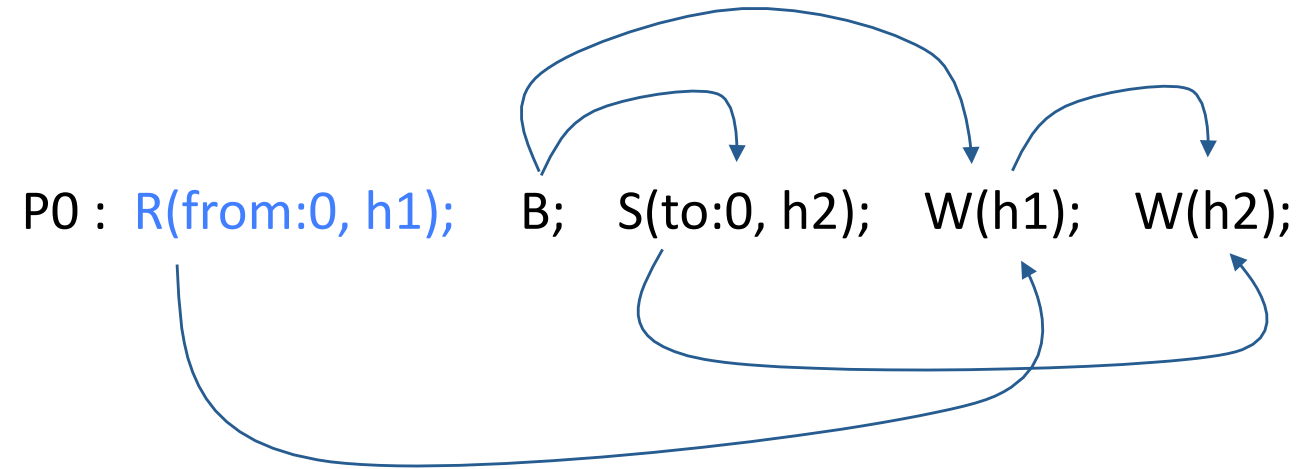
How Example Auto-send works

The HB



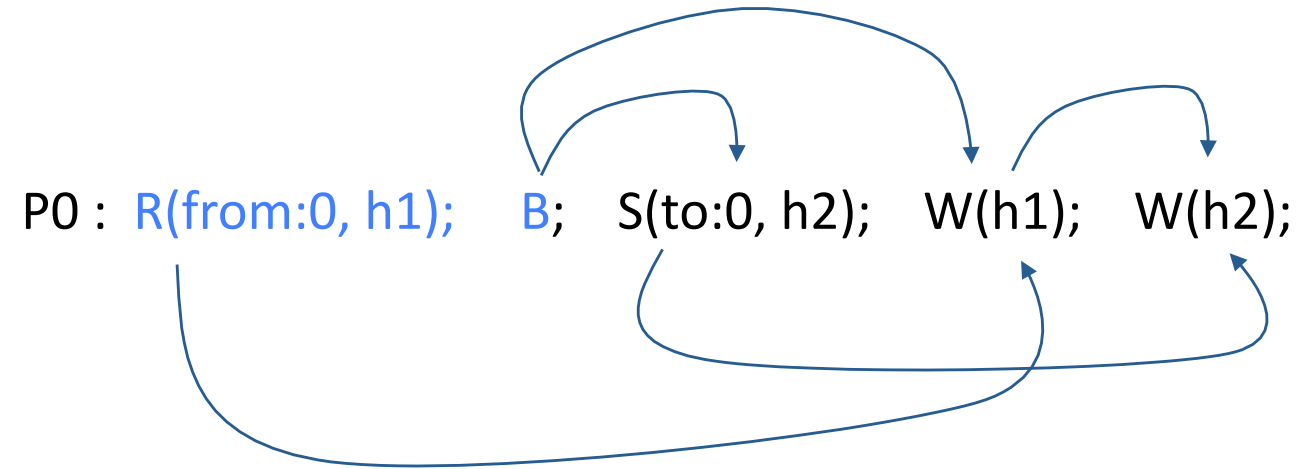
How Example Auto-send works

Issue R(from:0, h1), because prior to issuing R, P0 is not at a fence



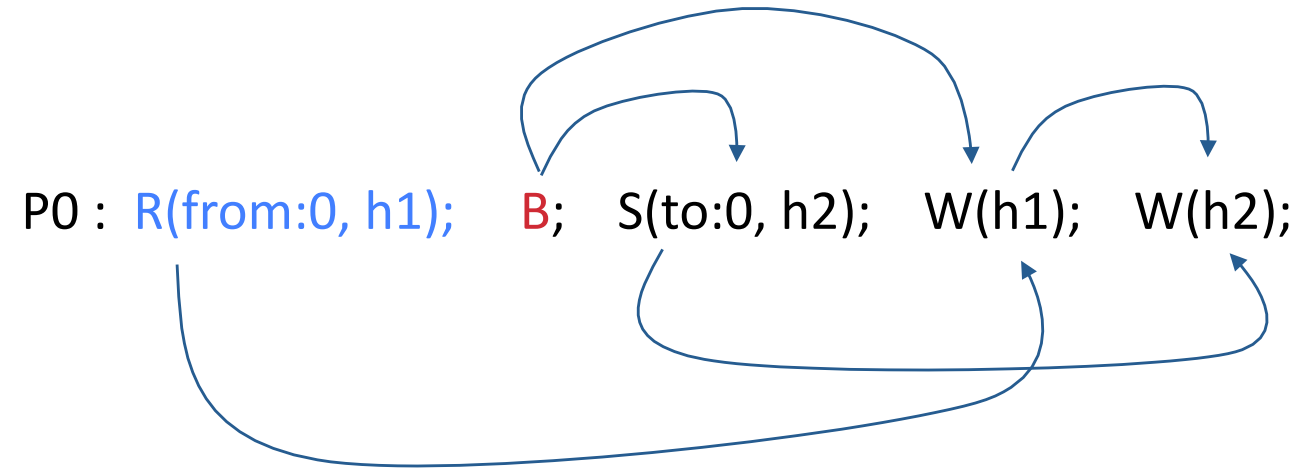
How Example Auto-send works

Issue B, because after issuing R, P0 is not at a fence



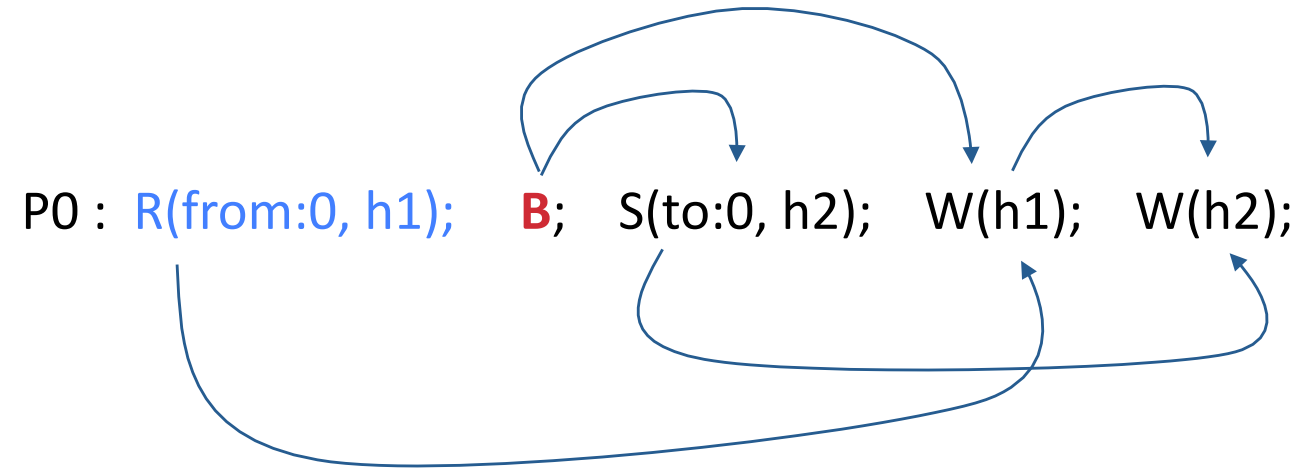
How Example Auto-send works

Form match set; Match-enabled set is {B}



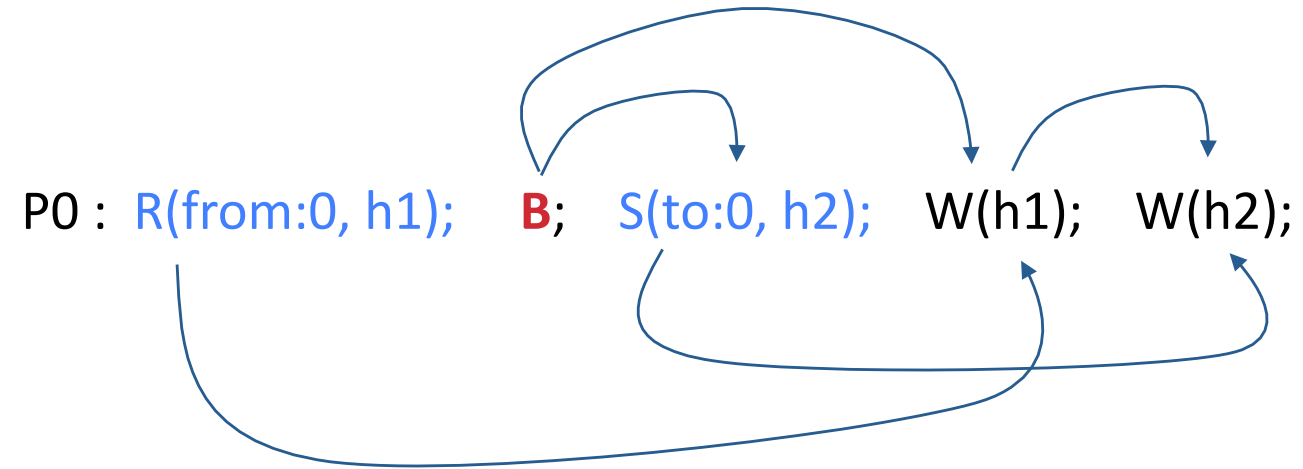
How Example Auto-send works

Fire Match-enabled set {B}



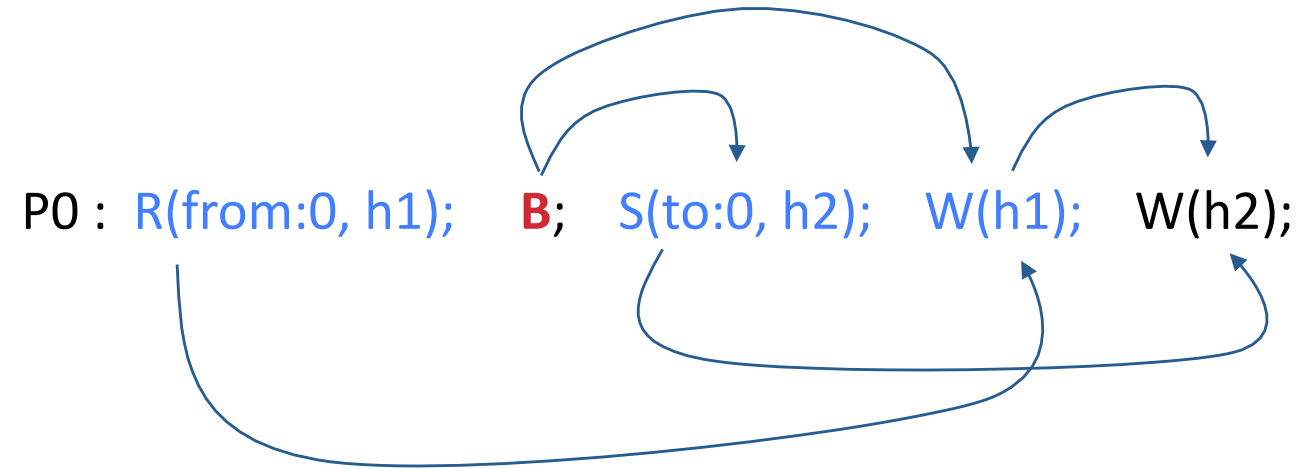
How Example Auto-send works

Issue $S(\text{to:0}, h2)$ because since B is gone, P0 is no longer at a fence



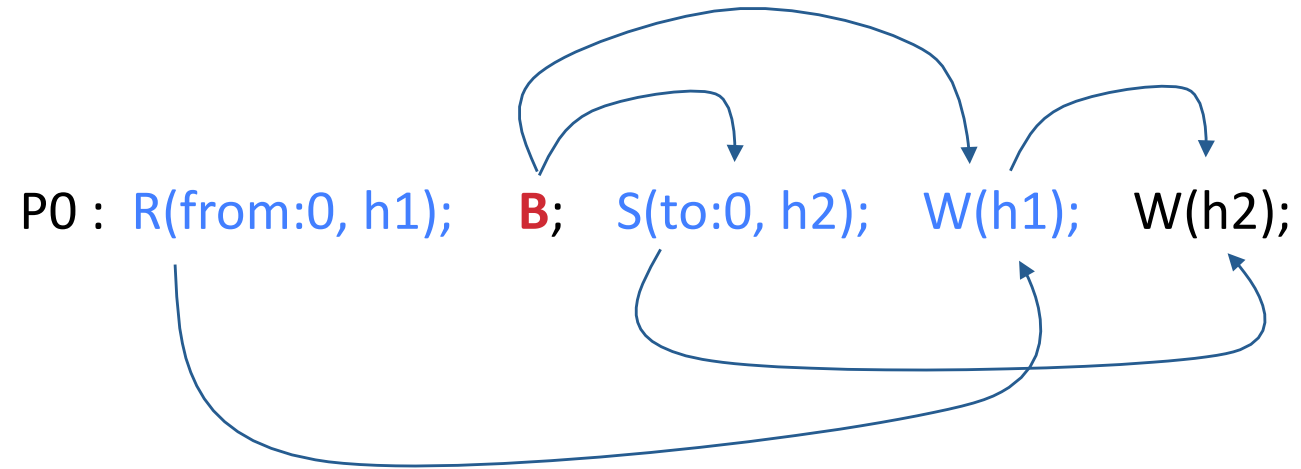
How Example Auto-send works

Issue $W(h1)$ because after $S(to:0, h2)$, P0 is not at a fence



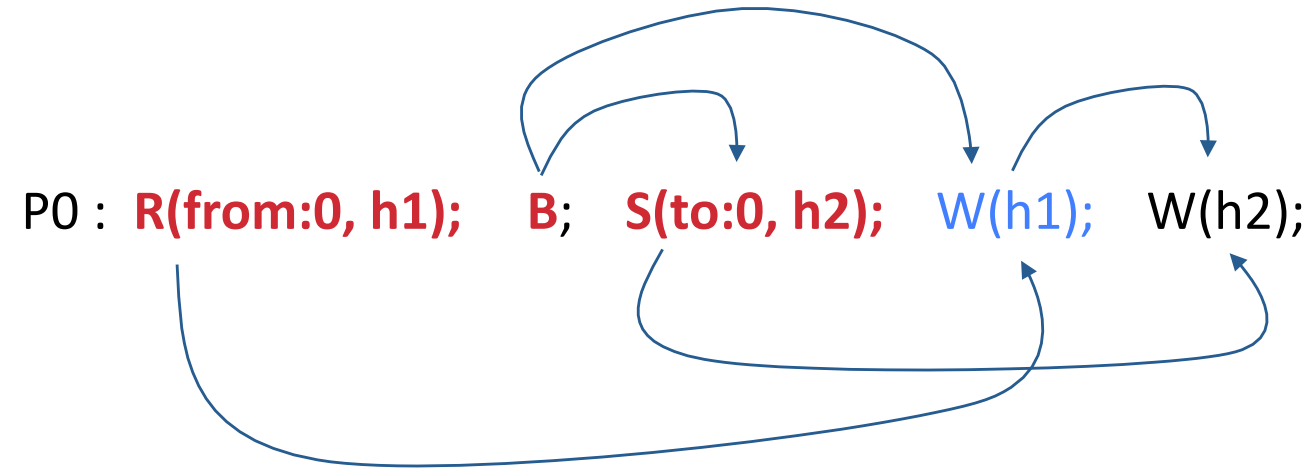
How Example Auto-send works

Can't form a $\{ W(h1) \}$ match set because it has an unmatched ancestor (namely $R(\text{from}:0, h1)$).



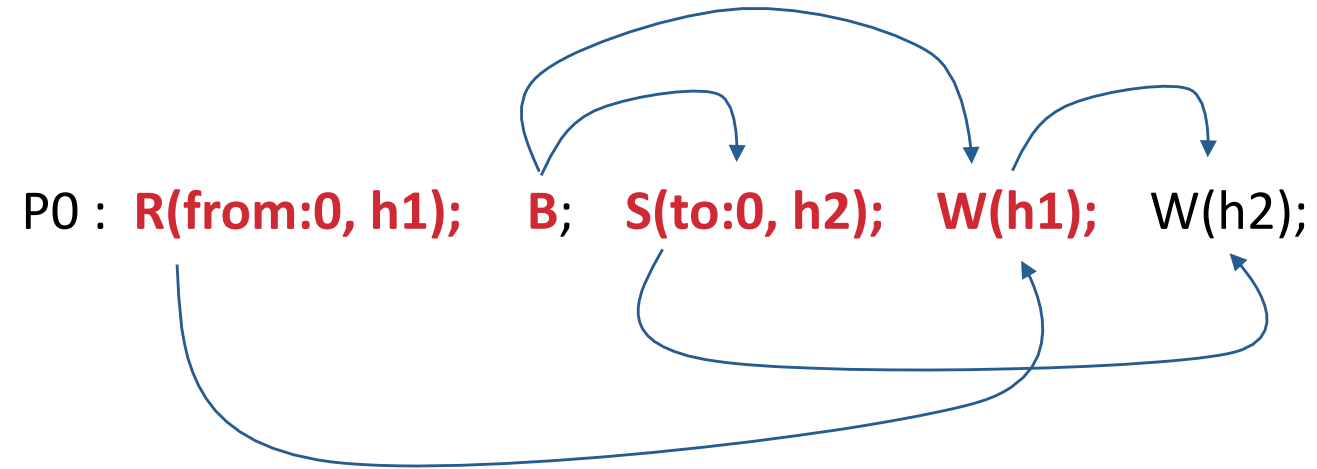
How Example Auto-send works

Form and **issue** the { R(from:0, h1), S(to:0, h2) } match set, and issue



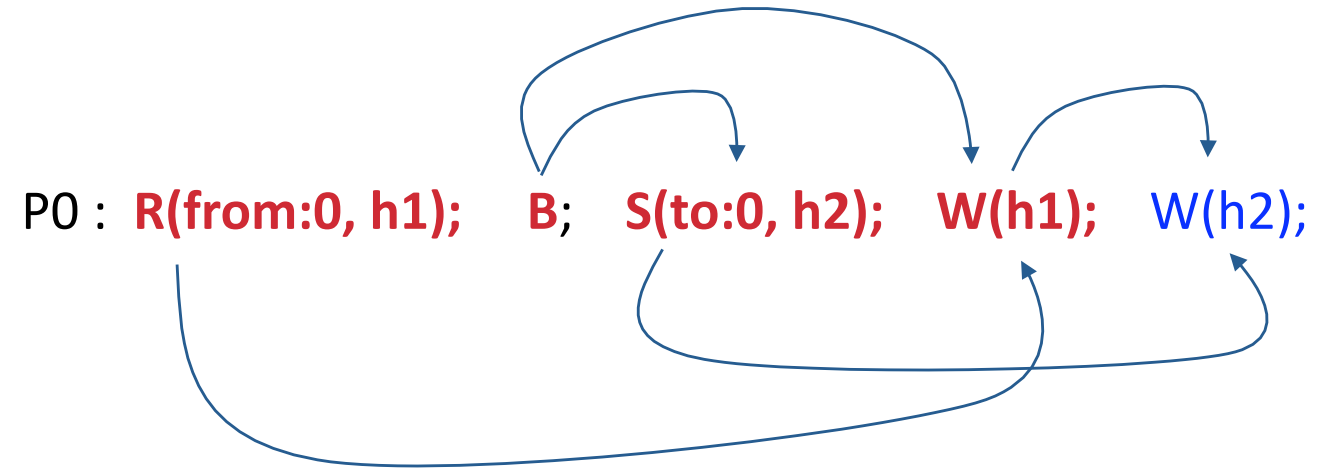
How Example Auto-send works

Now form and **issue** the match set { W(h1) }



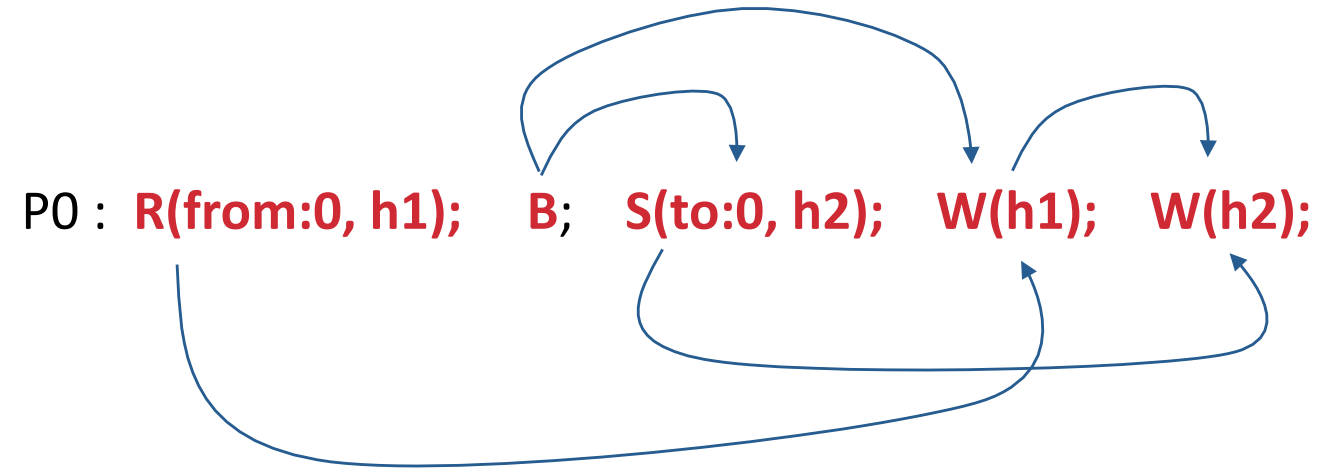
How Example Auto-send works

Now **issue** W(h2)



How Example Auto-send works

Form match set { W(h2) } and **fire** it. Done.



End of D