

# Practical Formal Verification of MPI and Thread Programs

Sarvani Vakkalanka

Anh Vo\*

Michael DeLisi

Sriram Ananthakrishnan

Alan Humphrey

Christopher Derrick

Yu Yang

Ganesh Gopalakrishnan\*

Robert M. Kirby\*

\* = presenters

*School of Computing, University of Utah,*

*Salt Lake City, UT 84112, USA*

[http:// www.cs.utah.edu / formal\\_verification / europvm09-tutorial-mpi-threading-fv](http://www.cs.utah.edu/formal_verification/europvm09-tutorial-mpi-threading-fv)

Supported by NSF CNS 0509379, CCF 0811429, CCF 0903408,

SRC tasks TJ 1847.001 and TJ 1993, and Microsoft

# Additional Acknowledgements for this tutorial

---

- **Other students involved:**

Salman Pervez, Robert Palmer, Guodong Li, Geof Sawaya, Subodh Sharma, Grzegorz Szubzda, Jason Williams, Simone Atzeni, Wei-Fan Chiang

- **External Collaborators:**

**ANL / UIUC** : Rajeev Thakur, Bill Gropp, Rusty Lusk

**IBM** : Beth Tibbits

**LLNL** : Bronis de Supinski, Martin Schulz, Dan Quinlan

**Microsoft** : Robert Palmer, Dennis Crain, Shahrokh Mortazavi

## 9:00 to 10:30

---

- Overview of Formal Verification, especially Dynamic Verification
- Overview of MPI
- Demo of our tool ISP
- Architecture of ISP
- Presentation of Any\_src\_can\_deadlock (from Umpire test suite)
- Our algorithm POE (Partial Order avoiding Elusive interleavings)
- Presentation of POE-Illustration
- Present details of POE-Illustration: ISP's Eclipse framework and GUI
- Boot into LiveDVD and practice on POE-Illustration

## 10:30 to 11:00

---

- Coffee Break
- IMPORTANT : Please give feedback before it is too late
  - Too fast ?
  - Too slow ?
  - Just right !! ?
  - Assuming a lot ?
  - Other suggestions ?
- We will TRY to take into account these valuable suggestions!

## 11:00 to 12:00

---

- Illustration of Resource Dependent Deadlocks, and Detection
- Illustration of Resource Leak, and Detection
- Iprobe behavior, and illustration using GUI
- Assertion Violation in Red/Blue Problem
- Audience Participation in Above Exercises
- ISP's Theory : MPI Happens-before
  - Also called “matches before, completes before” in the tool

## 12:00 to 12:30

---

- Example of Matrix Multiplication: Four Variations
- Analysis of these variations using ISP, with Audience Participation

14:00 to 15:00

---

- Assisted Problem Solving by Audience

15:00 to 15:30

---

- Overview of Dynamic Verification of Shared Memory Thread Programs

## 16:00 to 17:30

---

- Dynamic Verification of Thread Programs using Inspect
- Concluding Remarks

# Overview of Formal Verification methods for Validating Concurrent Systems

About 30 minutes - by Ganesh



For many important reasons, we advocate  
**Dynamic Formal Verification** methods

---

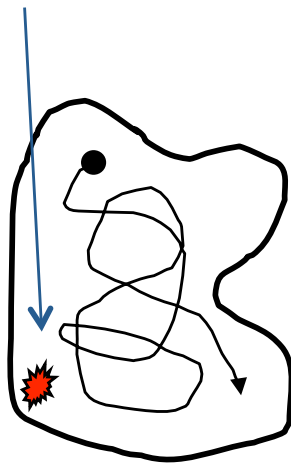
- Designers require a push-button debugger-like interface
  - But one that offers coverage guarantees and deeper insights

# For many important reasons, we advocate **Dynamic Formal Verification** methods

---

- Testing methods suffer from bug omissions

Bug Omissions



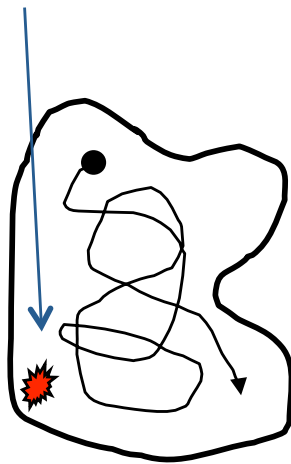
**X**

# For many important reasons, we advocate **Dynamic Formal Verification** methods

---

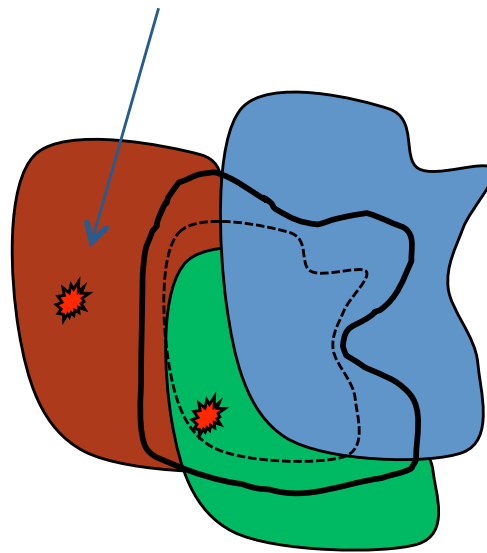
- Testing methods suffer from bug omissions
- Static analysis methods generate many false alarms

Bug Omissions



**X**

False Alarms



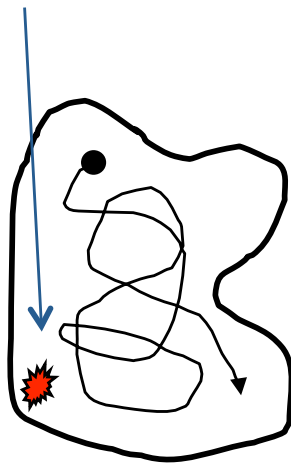
**X**

# For many important reasons, we advocate **Dynamic Formal Verification** methods

---

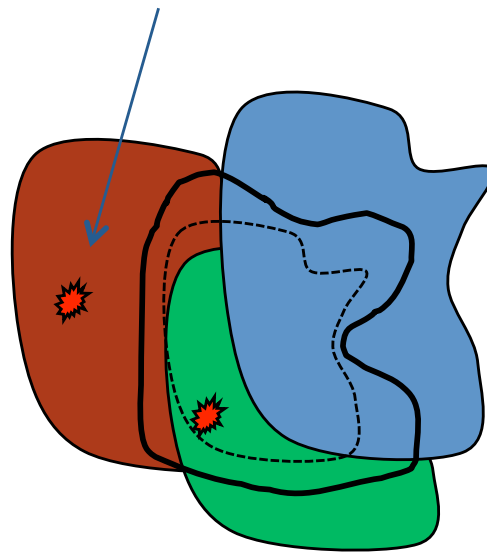
- Testing methods suffer from bug omissions
- Static analysis methods generate many false alarms
- Model based verification requires tedious model building

Bug Omissions



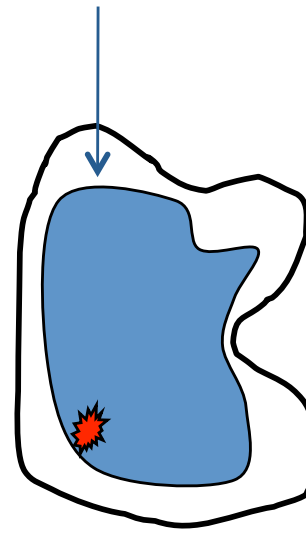
**X**

False Alarms



**X**

Tedious Modeling



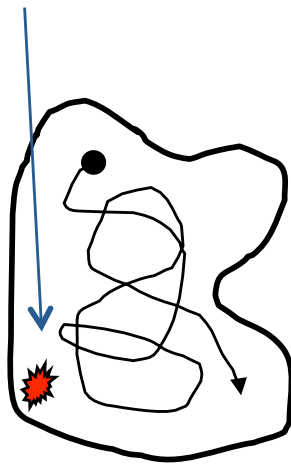
**X**

# For many important reasons, we advocate **Dynamic Formal Verification** methods

---

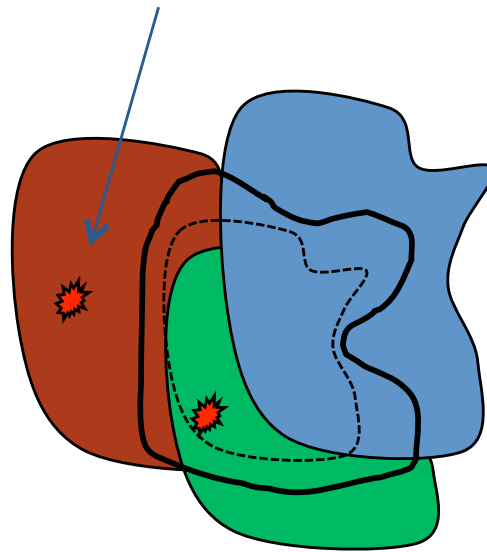
- Testing methods suffer from bug omissions
- Static analysis methods generate many false alarms
- Model based verification requires tedious model building
- **Dynamic verification methods are ideal for designers!**

Bug Omissions



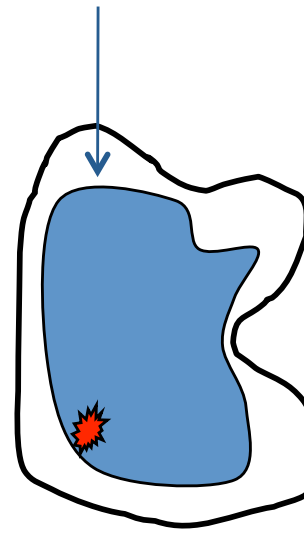
**X**

False Alarms



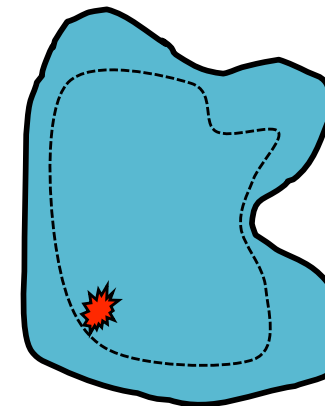
**X**

Tedious Modeling



**X**

- **No omissions**
- **No false alarms**
- **No need for modeling**



**✓**

# Growing Importance of Dynamic Verification

---

**Code written using mature libraries  
(MPI, OpenMP, PThreads, ...)**

**API calls made from real  
programming languages  
(C, Fortran, C++)**

**Runtime semantics determined by  
realistic compilers and runtimes**

***Dynamic Verification  
Methods are going to  
be very important for  
real engineers !***

***(static analysis and model  
based verification can  
play important  
supportive roles)***

# A Brief Survey of Dynamic Verification tools

---

- Verisoft Project
  - Used for telephone switch software verification in Bell Labs
  - Available

# A Brief Survey of Dynamic Verification tools

---

- Verisoft Project
  - Used for telephone switch software verification in Bell Labs
  - Available
- **The Java Pathfinder Project**
  - Developed at NASA for Java Control Software
  - On SourceForge

# A Brief Survey of Dynamic Verification tools

---

- Verisoft Project
  - Used for telephone switch software verification in Bell Labs
  - Available
- The Java Pathfinder Project
  - Developed at NASA for Java Control Software
  - On SourceForge
- **The CHES Project**
  - **Microsoft Research ; available for academic institutions**
  - **In use within Microsoft product groups, and used by academics**

# A Brief Survey of Dynamic Verification tools

---

- Verisoft Project
  - Used for telephone switch software verification in Bell Labs
  - Available
- The Java Pathfinder Project
  - Developed at NASA for Java Control Software
  - On SourceForge
- The CHES Project
  - Microsoft Research ; available for academic institutions
  - In use within Microsoft product groups, and used by academics
- **Inspect : Our fairly unique Pthread / C verifier**
  - **Discussed in this tutorial**

# A Brief Survey of Dynamic Verification tools

---

- Verisoft Project
  - Used for telephone switch software verification in Bell Labs
  - Available
- The Java Pathfinder Project
  - Developed at NASA for Java Control Software
  - On SourceForge
- The CHES Project
  - Microsoft Research ; available for academic institutions
  - In use within Microsoft product groups, and used by academics
- **Inspect : Our fairly unique Pthread / C verifier**
  - **Discussed in this tutorial**
- **ISP : Our very unique MPI / C program verifier**
  - **Main focus of THIS TUTORIAL !!**

# Example : How ISP Effects Dynamic Verification

---

## Example : How ISP Effects Dynamic Verification

---

- Somehow Instruments the Source / Binary

## Example : How ISP Effects Dynamic Verification

---

- Somehow Instruments the Source / Binary
  - **Through PMPI at source level**

## Example : How ISP Effects Dynamic Verification

---

- Somehow Instruments the Source / Binary
  - **Through PMPI**
- Runs the code under a *verification scheduler*

## Example : How ISP Effects Dynamic Verification

---

- Somehow Instruments the Source / Binary
  - Through PMPI
- Runs the code under a *verification scheduler*
  - ‘Hijacks’ MPI Function Calls

## Example : How ISP Effects Dynamic Verification

---

- Somehow Instruments the Source / Binary
  - Through PMPI
- Runs the code under a *verification scheduler*
  - ‘Hijacks’ MPI Function Calls
    - By interposing a profiler

## Example : How ISP Effects Dynamic Verification

---

- Somehow Instruments the Source / Binary
  - Through PMPI
- Runs the code under a *verification scheduler*
  - ‘Hijacks’ MPI Function Calls
    - By interposing a profiler
  - Exerts its own Interleaving Generation Control

## Example : How ISP Effects Dynamic Verification

---

- Somehow Instruments the Source / Binary
  - Through PMPI
- Runs the code under a *verification scheduler*
  - ‘Hijacks’ MPI Function Calls
    - By interposing a profiler
  - Exerts its own Interleaving Generation Control
    - Selective replay, Dynamic Instruction Rewriting

## Example : How ISP Effects Dynamic Verification

---

- Somehow Instruments the Source / Binary
  - Through PMPI
- Runs the code under a *verification scheduler*
  - ‘Hijacks’ MPI Function Calls
    - By interposing a profiler
  - Exerts its own Interleaving Generation Control
    - Selective replay, Dynamic Instruction Rewriting
  - TRIES HARD to generate only RELEVANT interleavings

## Example : How ISP Effects Dynamic Verification

---

- Somehow Instruments the Source / Binary
  - Through PMPI
- Runs the code under a *verification scheduler*
  - ‘Hijacks’ MPI Function Calls
    - By interposing a profiler
  - Exerts its own Interleaving Generation Control
    - Selective replay, Dynamic Instruction Rewriting
  - TRIES HARD to generate only RELEVANT interleavings
    - Only replays around “non-determinism”

## Example : How ISP Effects Dynamic Verification

---

- Somehow Instruments the Source / Binary
  - Through PMPI
- Runs the code under a *verification scheduler*
  - ‘Hijacks’ MPI Function Calls
    - By interposing a profiler
  - Exerts its own Interleaving Generation Control
    - Selective replay, Dynamic Instruction Rewriting
  - TRIES HARD to generate only RELEVANT interleavings
    - Only replays around “non-determinism”
  - Does ‘stateless’ (replay) verification

## Example : How ISP Effects Dynamic Verification

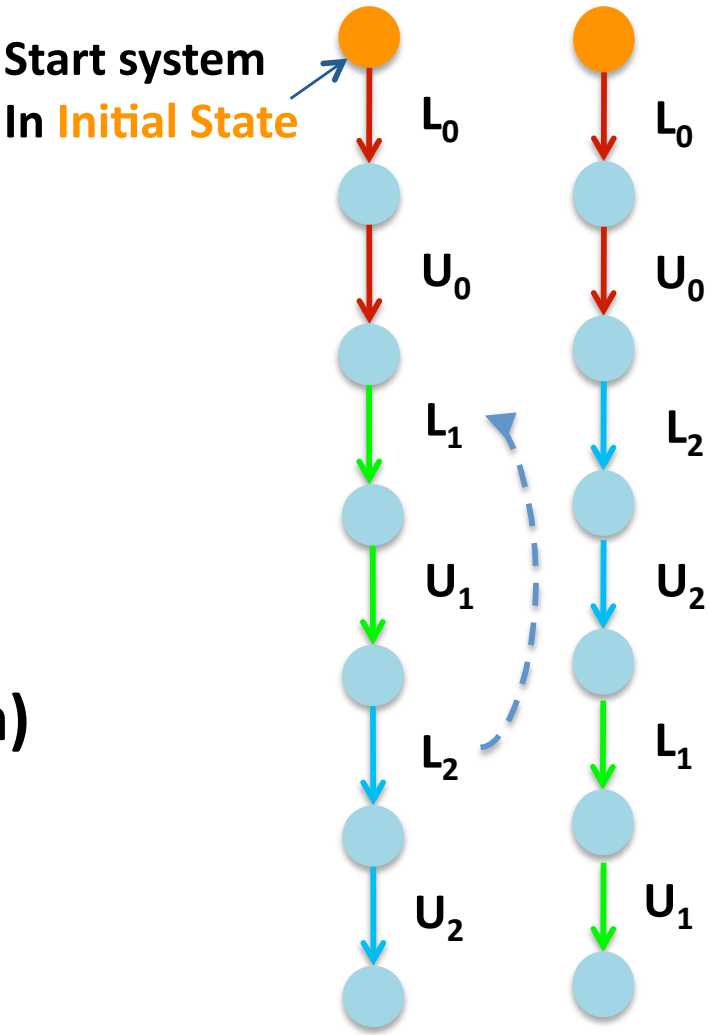
---

- Somehow Instruments the Source / Binary
  - Through PMPI
- Runs the code under a *verification scheduler*
  - ‘Hijacks’ MPI Function Calls
    - By interposing a profiler
  - Exerts its own Interleaving Generation Control
    - Selective replay, Dynamic Instruction Rewriting
  - TRIES HARD to generate only RELEVANT interleavings
    - Only replays around “non-determinism”
  - Does ‘stateless’ (replay) verification
    - Restarts from MPI\_Init for each new interleaving

# Sketch of Stateless / Replay Verification

**Red, Green, and Blue** moves  
Belong to different processes

**Dotted arrow shows some  
Dependency  
(e.g., runtime non-determinism)**

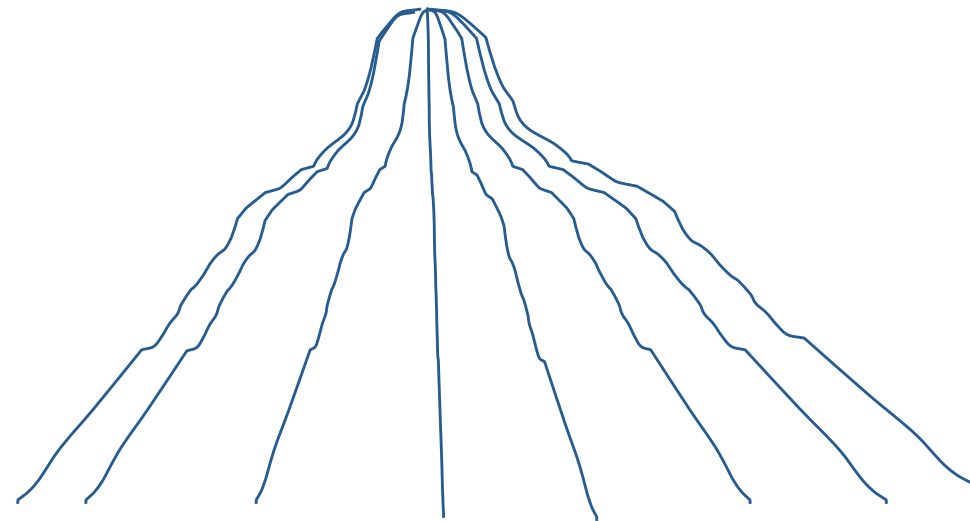
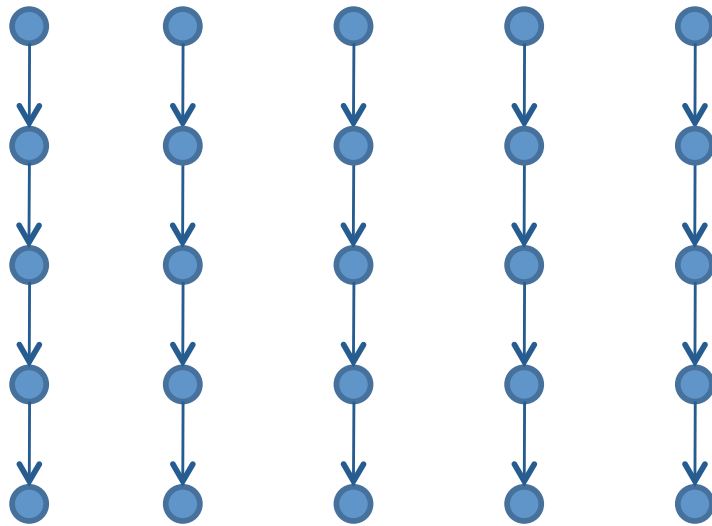


Exponential number of TOTAL Interleavings - most are EQUIVALENT - generate only RELEVANT ones !!

---



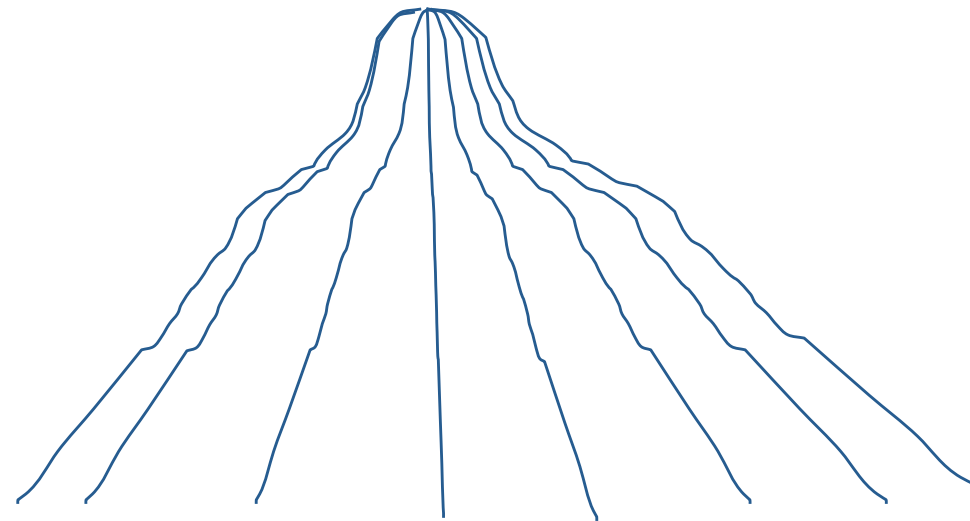
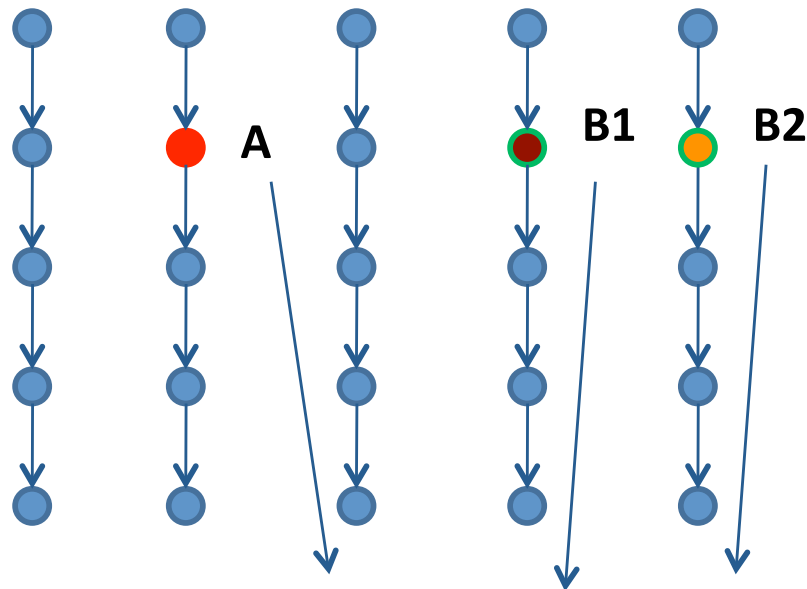
**TOTAL > 10 Billion Interleavings !!**



# Exponential number of TOTAL Interleavings - most are EQUIVALENT - generate only RELEVANT ones !!



TOTAL > 10 Billion Interleavings !!



These are the only dependent actions

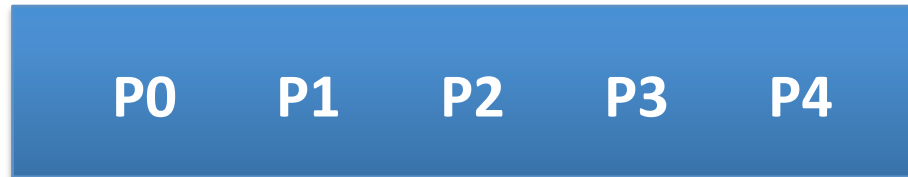
E.g. One ANY-SOURCE (wildcard) receive ● A

And two of its MATCHING SENDS ● B1

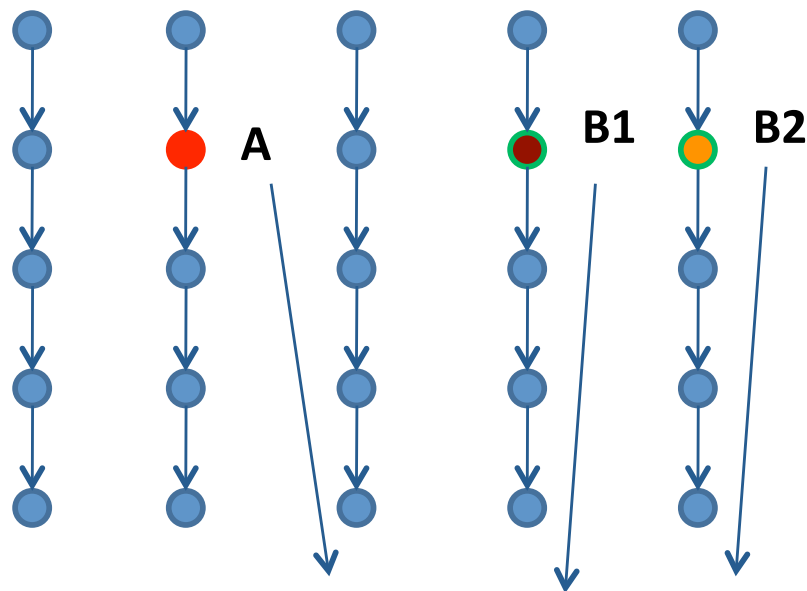
● B2

Point-to-point actions can be issued in ANY order

# Exponential number of TOTAL Interleavings - most are EQUIVALENT - generate only RELEVANT ones !!



TOTAL > 10 Billion Interleavings !!



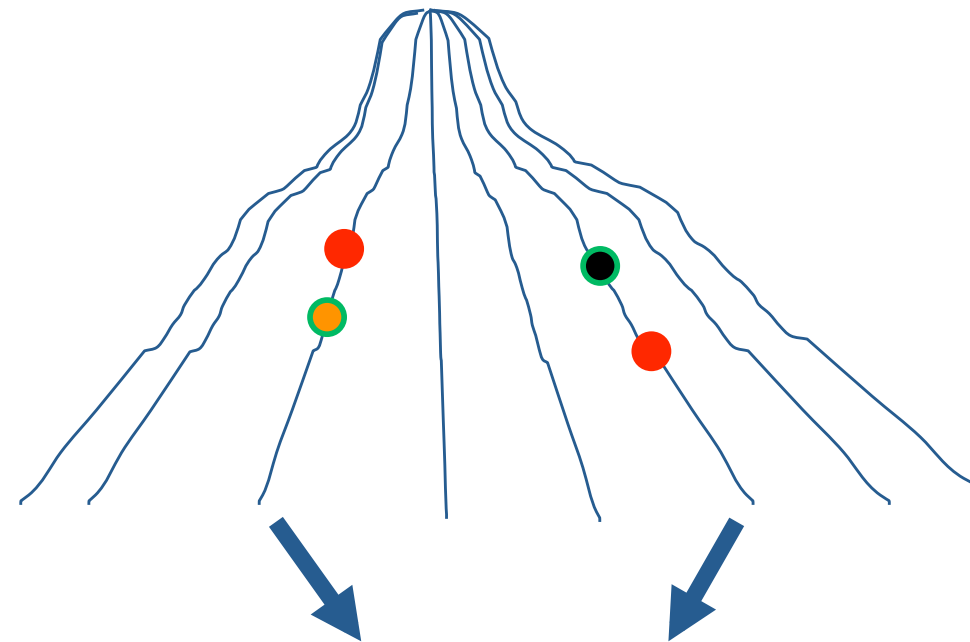
These are the only dependent actions

E.g. One ANY-SOURCE (wildcard) receive ● A

And two of its MATCHING SENDS ● B1

● B2

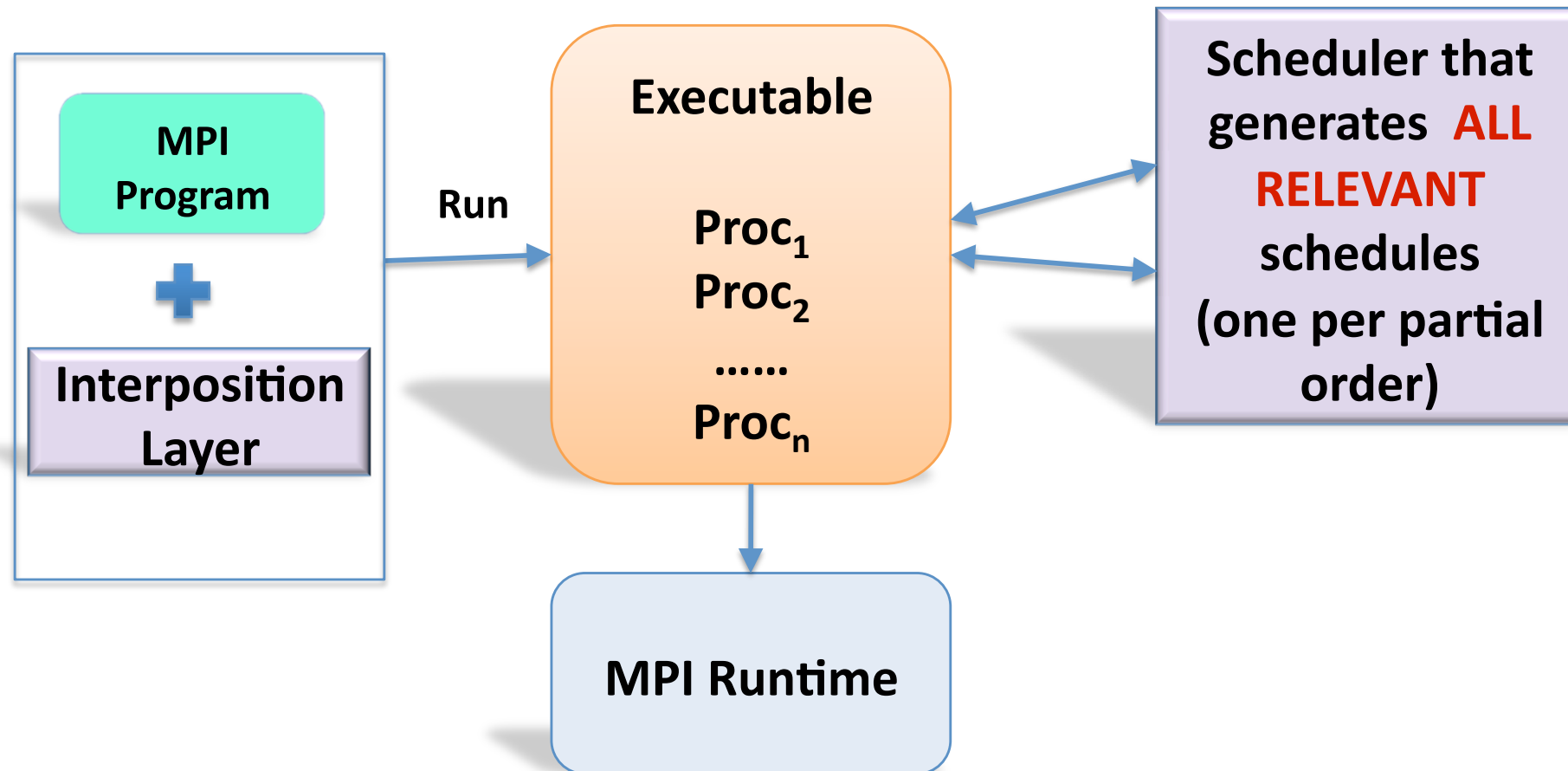
Point-to-point actions can be issued in ANY order



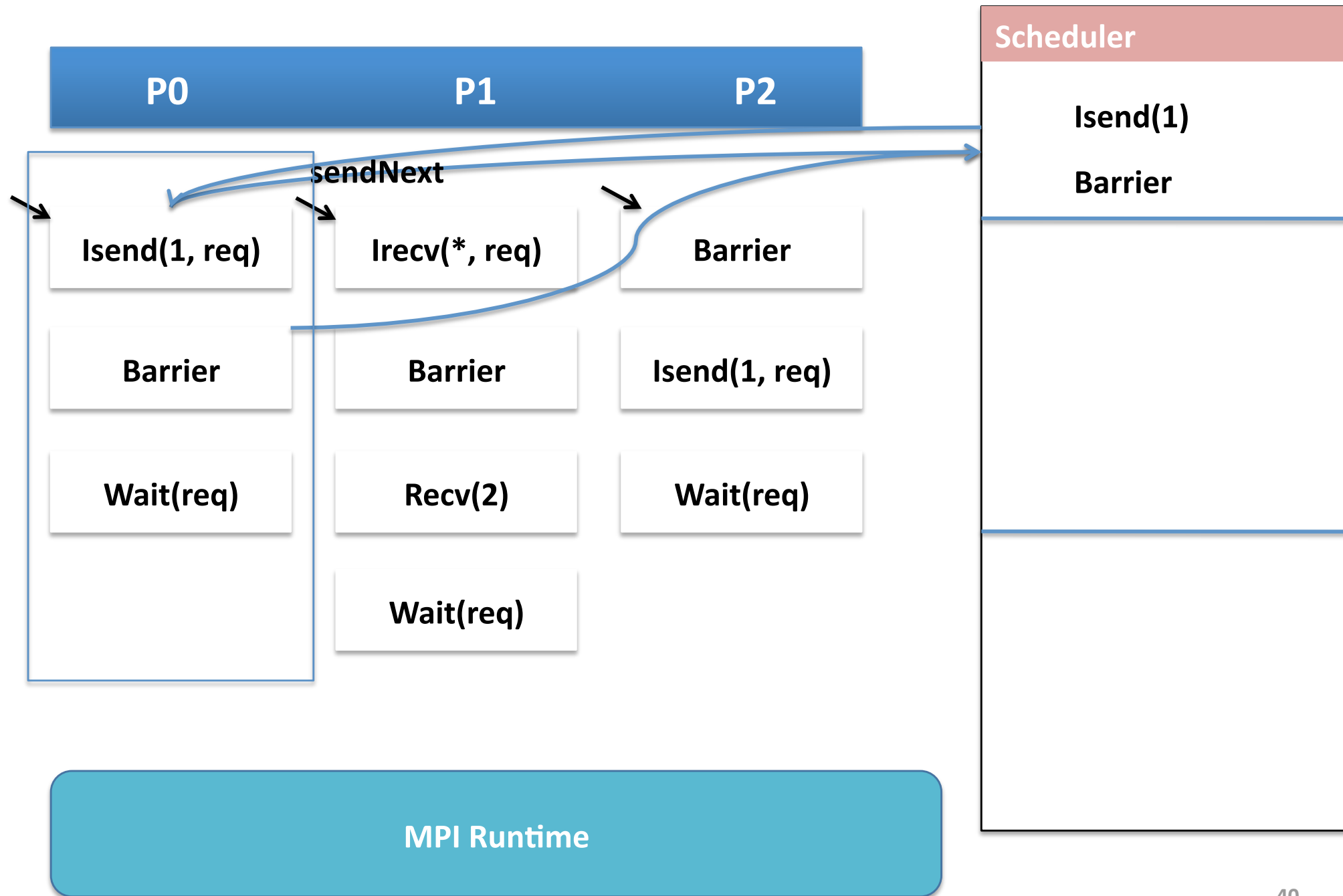
Only TWO RELEVANT Interleavings !

# Workflow of ISP

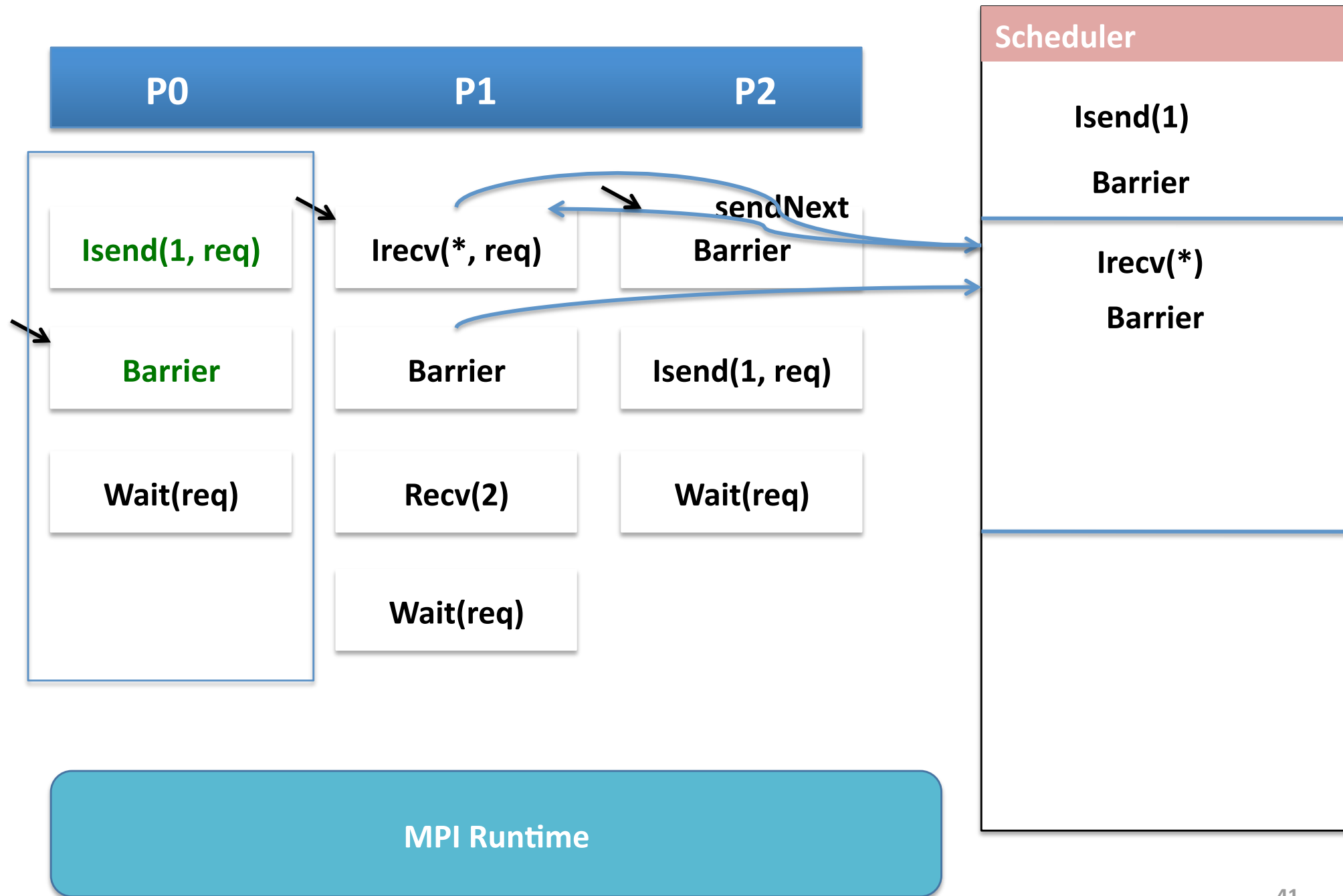
---



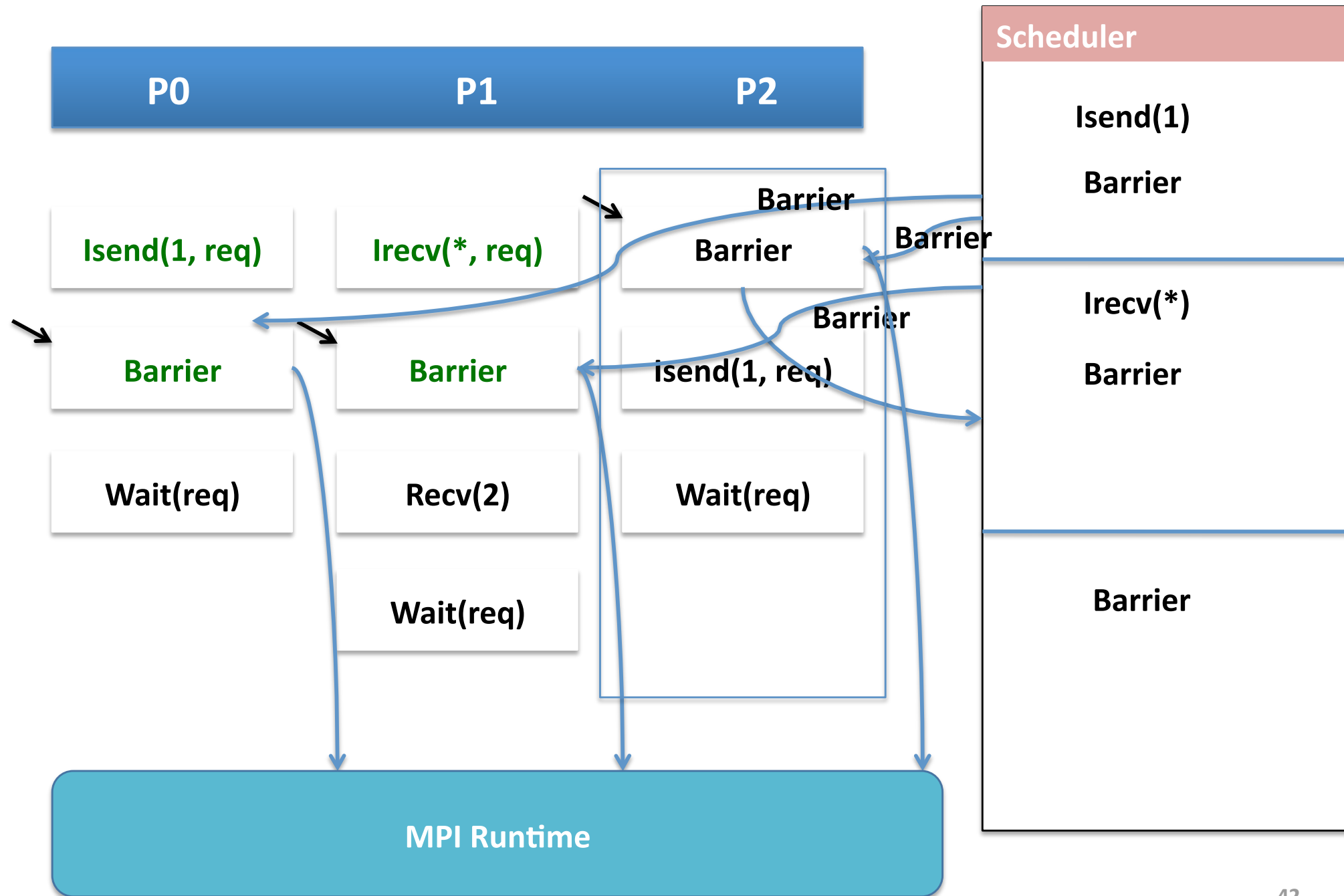
# Hijack Calls, Generate Relevant Interleavings



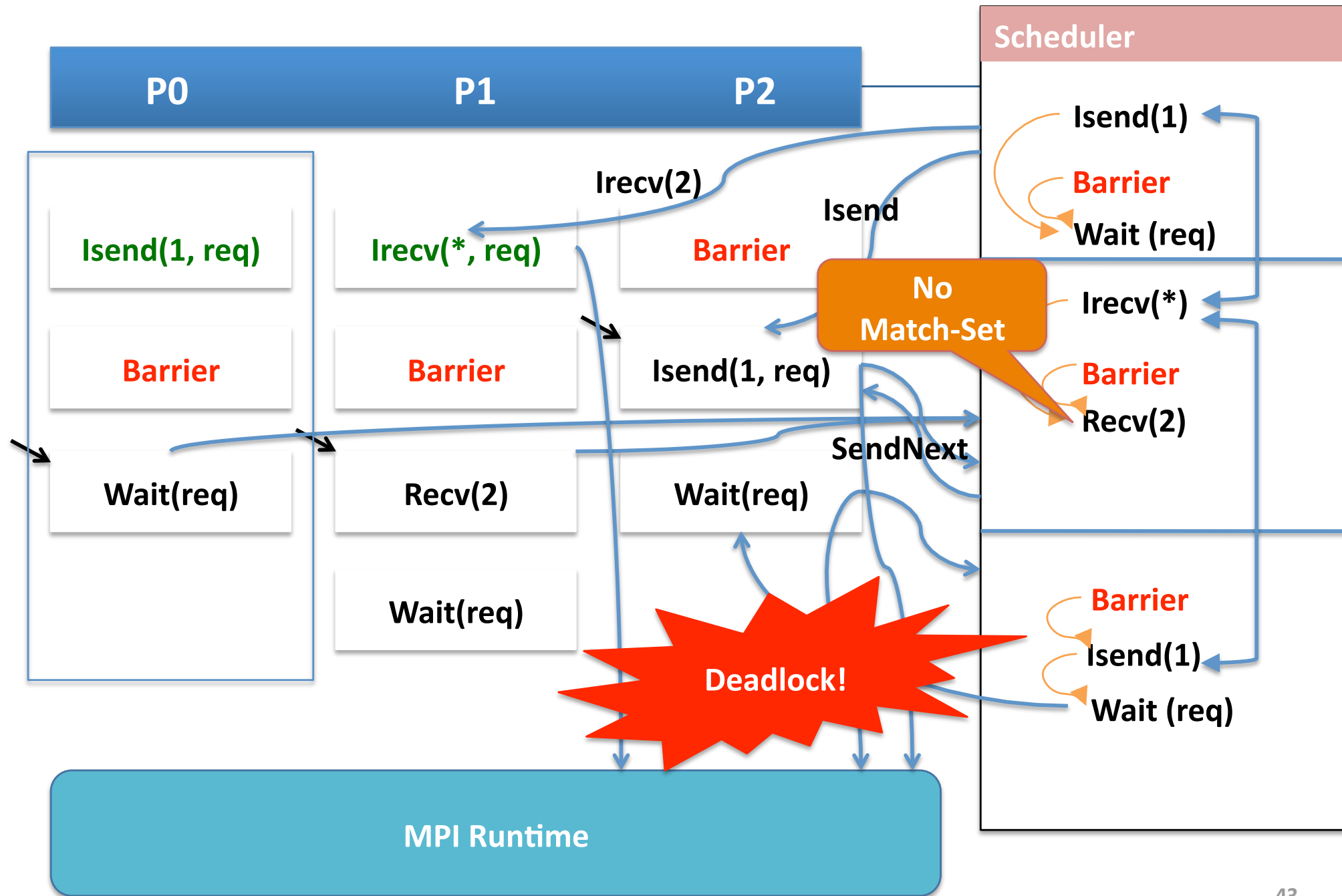
# Hijack Calls, Generate Relevant Interleavings



# Hijack Calls, Generate Relevant Interleavings



# Hijack Calls, Generate Relevant Interleavings



Let us see ISP in action on 'lucky.c' and 'unlucky.c'

---

- lucky.c has a deadlock that shows upon testing
- unlucky.c does not reveal a deadlock upon testing
- Testing is done using `mpicc ; mpirun`
- Verification is done using `ispcc ; isp`

## Example MPI program 'lucky.c' (lucky for tester)

---

### Process P0

R(from:\*, r1) ;

R(from:2, r2);

S(to:2, r3);

R(from:\*, r4);

All the Ws...

### Process P1

Sleep(3);

S(to:0, r1);

All the Ws...

### Process P2

//Sleep(3);

S(to:0, r1);

R(from:0, r2);

S(to:0, r3);

All the Ws...

## MPI program 'unlucky.c'

---

### Process P0

R(from:\*, r1) ;

R(from:2, r2);

S(to:2, r3);

R(from:\*, r4);

All the Ws...

### Process P1

// Sleep(3);

S(to:0, r1);

All the Ws...

### Process P2

Sleep(3);

S(to:0, r1);

R(from:0, r2);

S(to:0, r3);

All the Ws...

# Runs of lucky.c and unlucky.c on mpich using “standard testing” (“lucky” for tester)

---

```
mpicc lucky.c -o lucky.out
mpirun -np 3 ./lucky.out
(0) is alive on ganesh-desktop
(1) is alive on ganesh-desktop
(2) is alive on ganesh-desktop
Rank 0 did Irecv
Rank 2 did Send
Sleep over
Rank 1 did Send

[.. hang ..]
```

```
mpicc unlucky.c -o unlucky.out
mpirun -np 3 ./unlucky.out
(0) is alive on ganesh-desktop
(2) is alive on ganesh-desktop
(1) is alive on ganesh-desktop
Rank 0 did Irecv
Rank 1 did Send
Rank 0 got 11
Sleep over
Rank 2 did Send
(2) Finished normally
(1) Finished normally
(0) Finished normally

[.. OK ..]
```

# Runs of lucky.c and unlucky.c on mpich using “standard testing” (“lucky” for tester)

---

```
mpicc lucky.c -o lucky.out
mpirun -np 3 ./lucky.out
(0) is alive on ganesh-desktop
(1) is alive on ganesh-desktop
(2) is alive on ganesh-desktop
Rank 0 did Irecv
Rank 2 did Send
Sleep over
Rank 1 did Send

[.. hang ..]
```

```
mpicc unlucky.c -o unlucky.out
mpirun -np 3 ./unlucky.out
(0) is alive on ganesh-desktop
(2) is alive on ganesh-desktop
(1) is alive on ganesh-desktop
Rank 0 did Irecv
Rank 1 did Send
Rank 0 got 11
Sleep over
Rank 2 did Send
(2) Finished normally
(1) Finished normally
(0) Finished normally

[.. OK ..]
```

*ispcc ; isp will detect deadlock in both cases !!*

# Commands to verify lucky.c or unlucky.c

---

- With ISP at hand, WE ARE LUCKY IN BOTH CASES
  - Not just 'feeling lucky' !!
- COMMANDS RUN :
  - `lspcc lucky.c [ later try unlucky.c ]`
  - `lsp -n 3 -log /tmp/log1 ./a.out`
  - `ispUI /tmp/log1`

End of A