

“ Transforming Sources to Petri Nets: A Way to Analyse Execution of Parallel Programs ”

International Workshop on Petri Net Tools and APplications

March 3, 2008 - Marseille - France

- Université Pierre & Marie Curie, LIP6/MoVe, Paris, France

- **Fabrice Kordon**
- **Jean-Baptiste Voron**

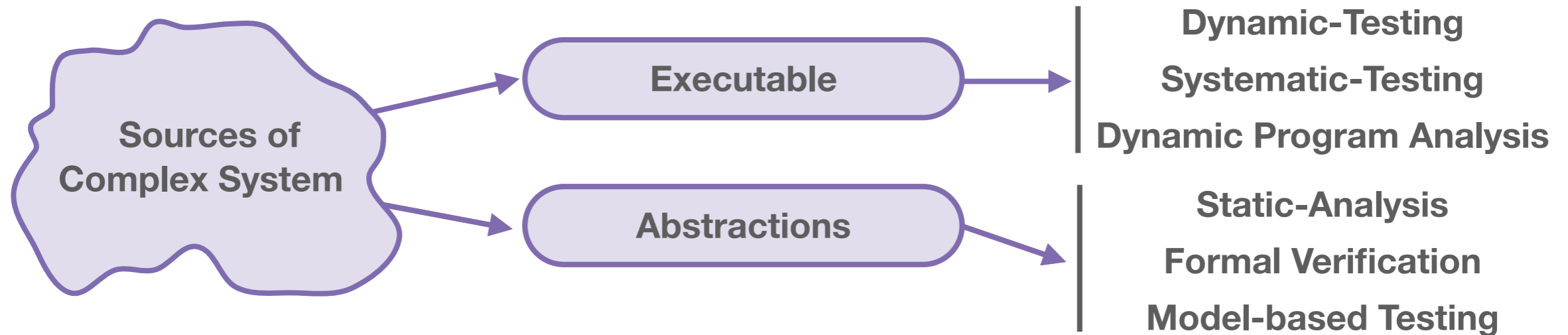
PNTAP'08

Objectives...

- **Intrusion Detection System** Context
 - Guarantee the “correct” behavior of a program
- Double check
 - During the execution : **Online Analysis**
 - Before the execution : **Offline Analysis**
- Without code instrumentation
- **Automatic process**

Software Analysis for Concurrent Systems

- Objective : **Producing correct software**



- Behavioral analysis

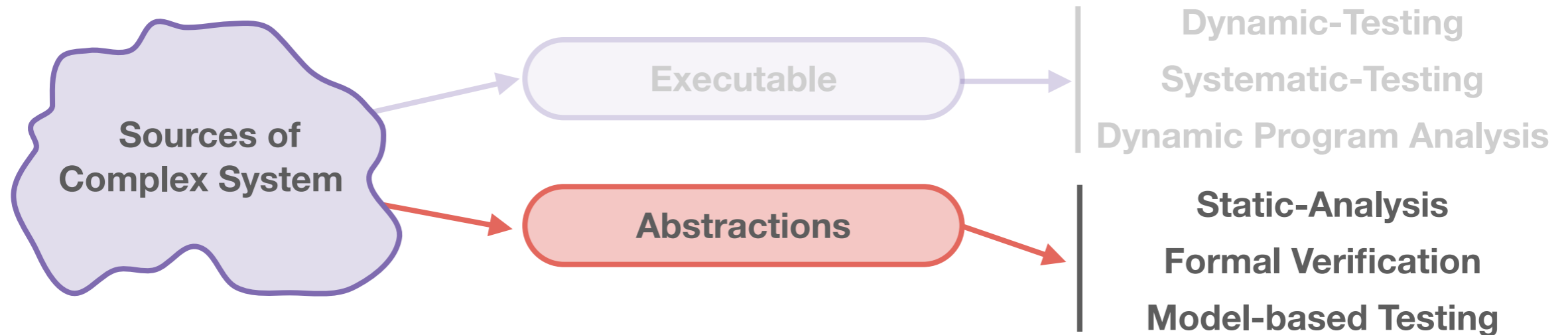
- Dealing with **very large** state-spaces !
- Formal specifications are **expensive** and **difficult to handle** for engineers

- Our approach

- **Automatic transformation** from source code to Petri nets
- Modular construction using only **relevant information** for analysis
- Reuse of **existing tools** dedicated to Petri net analysis

Software Analysis for Concurrent Systems

- Objective : **Producing correct software**



- Behavioral analysis

- Dealing with **very large** state-spaces !
- Formal specifications are **expensive** and **difficult to handle** for engineers

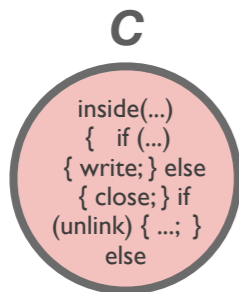
- Our approach

- **Automatic transformation** from source code to Petri nets
- Modular construction using only **relevant information** for analysis
- Reuse of **existing tools** dedicated to Petri net analysis

Model-Based Static Analysis

◎ Several steps :

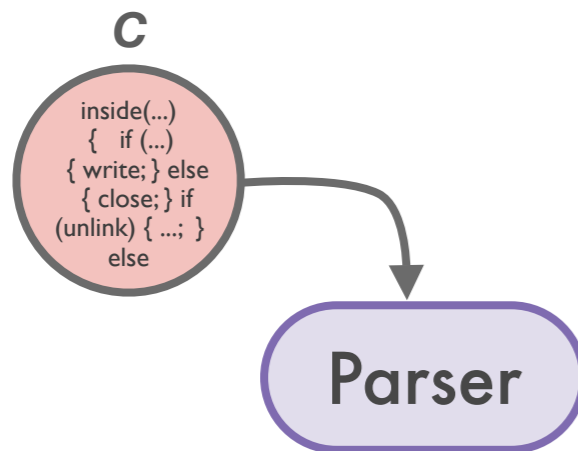
- **Extracting relevant information** from source code
 - *Structural information*
 - *Other dedicated information*
- **Building a complete Petri net model**
- **Reducing** and optimizing the produced net
- **Analysing** the final net



Model-Based Static Analysis

◎ Several steps :

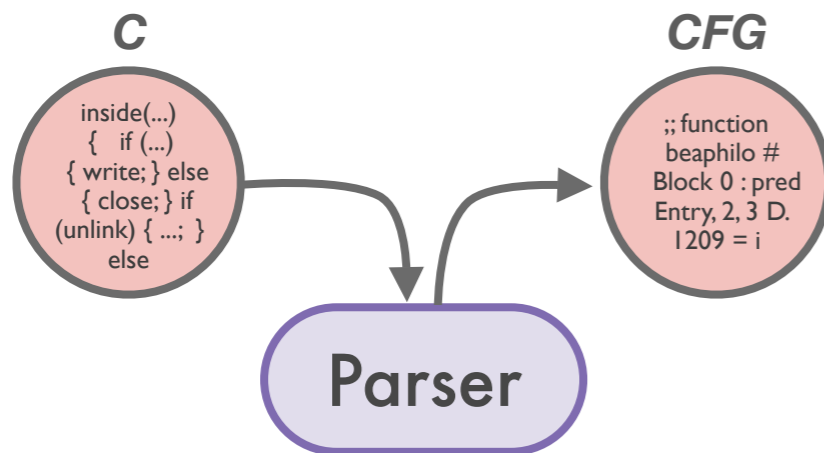
- **Extracting relevant information** from source code
 - *Structural information*
 - *Other dedicated information*
- **Building a complete Petri net model**
- **Reducing** and optimizing the produced net
- **Analysing** the final net



Model-Based Static Analysis

◎ Several steps :

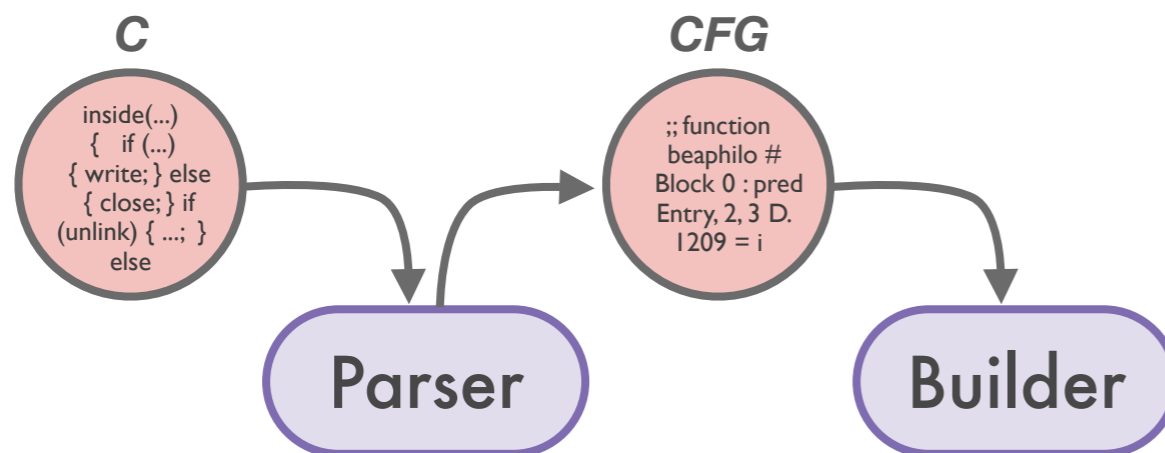
- **Extracting relevant information** from source code
 - *Structural information*
 - *Other dedicated information*
- **Building a complete Petri net model**
- **Reducing** and optimizing the produced net
- **Analysing** the final net



Model-Based Static Analysis

◎ Several steps :

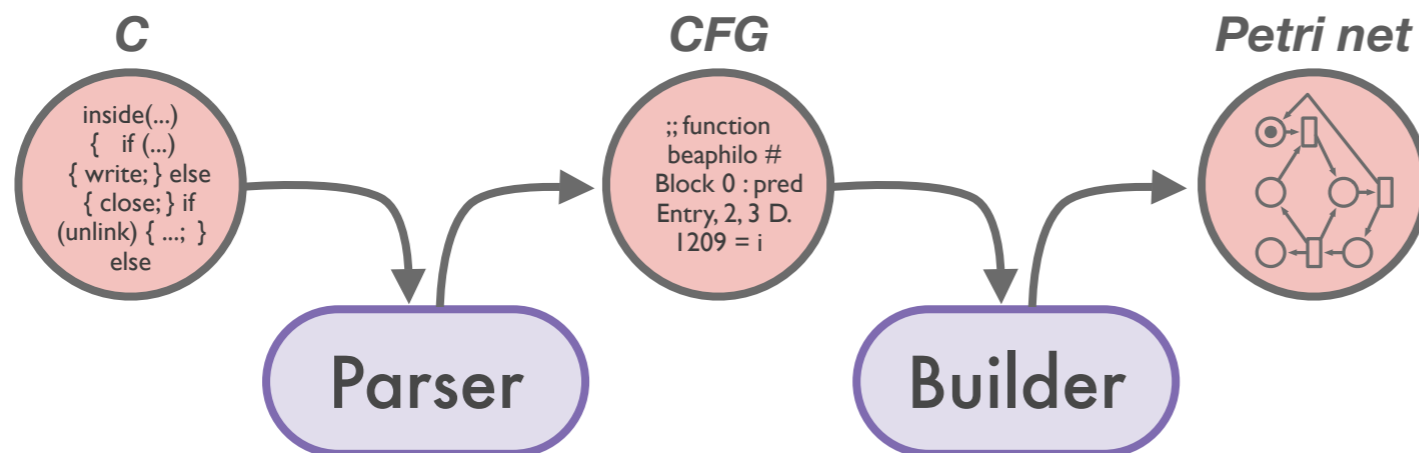
- **Extracting relevant information** from source code
 - *Structural information*
 - *Other dedicated information*
- **Building a complete Petri net model**
- **Reducing** and optimizing the produced net
- **Analysing** the final net



Model-Based Static Analysis

◎ Several steps :

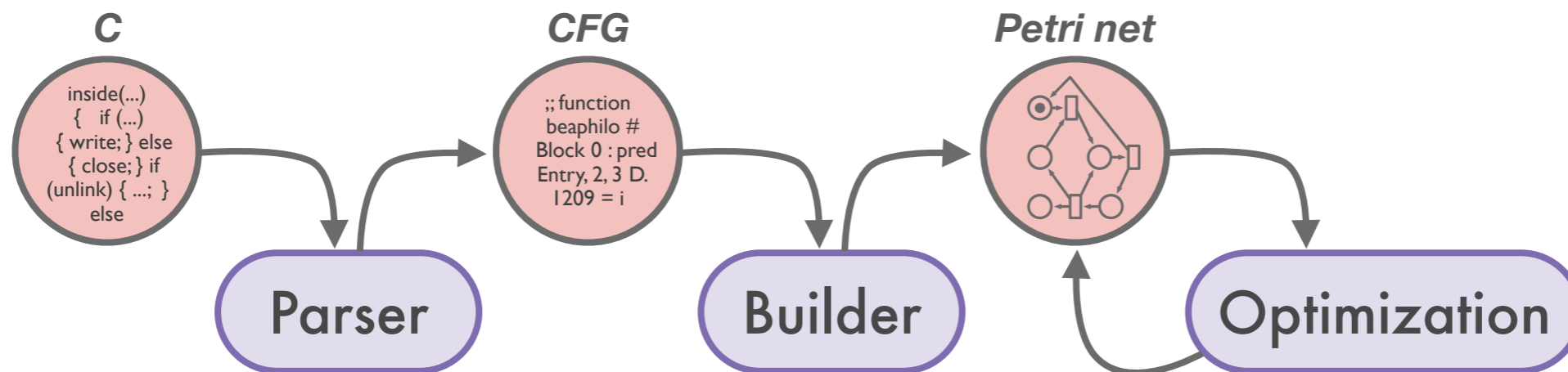
- **Extracting relevant information** from source code
 - *Structural information*
 - *Other dedicated information*
- **Building a complete Petri net model**
- **Reducing** and optimizing the produced net
- **Analysing** the final net



Model-Based Static Analysis

◎ Several steps :

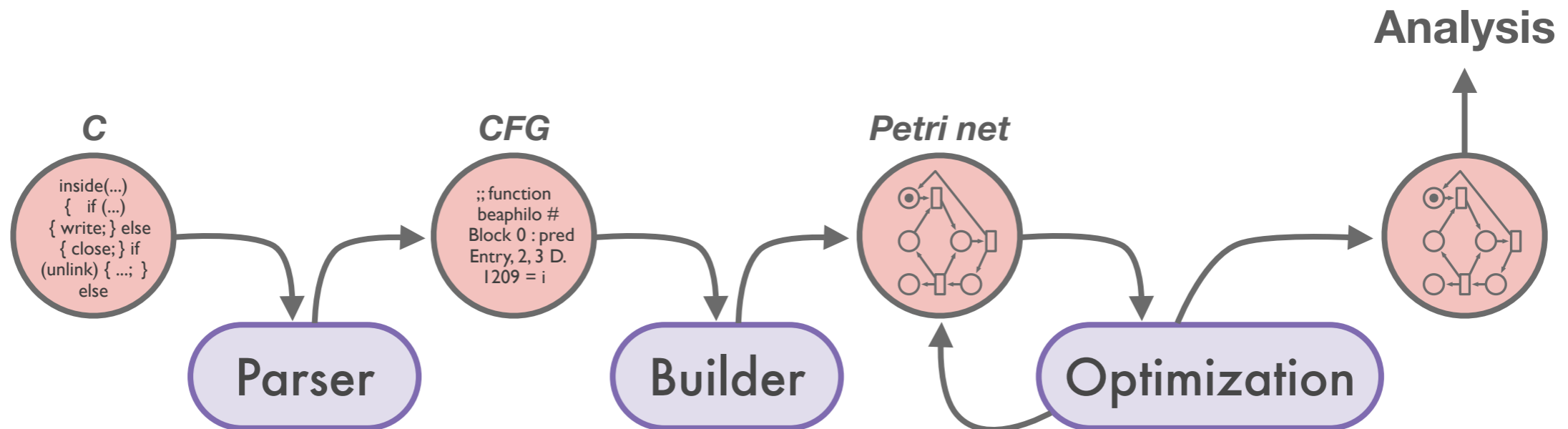
- **Extracting relevant information** from source code
 - *Structural information*
 - *Other dedicated information*
- **Building a complete Petri net model**
- **Reducing** and optimizing the produced net
- **Analysing** the final net



Model-Based Static Analysis

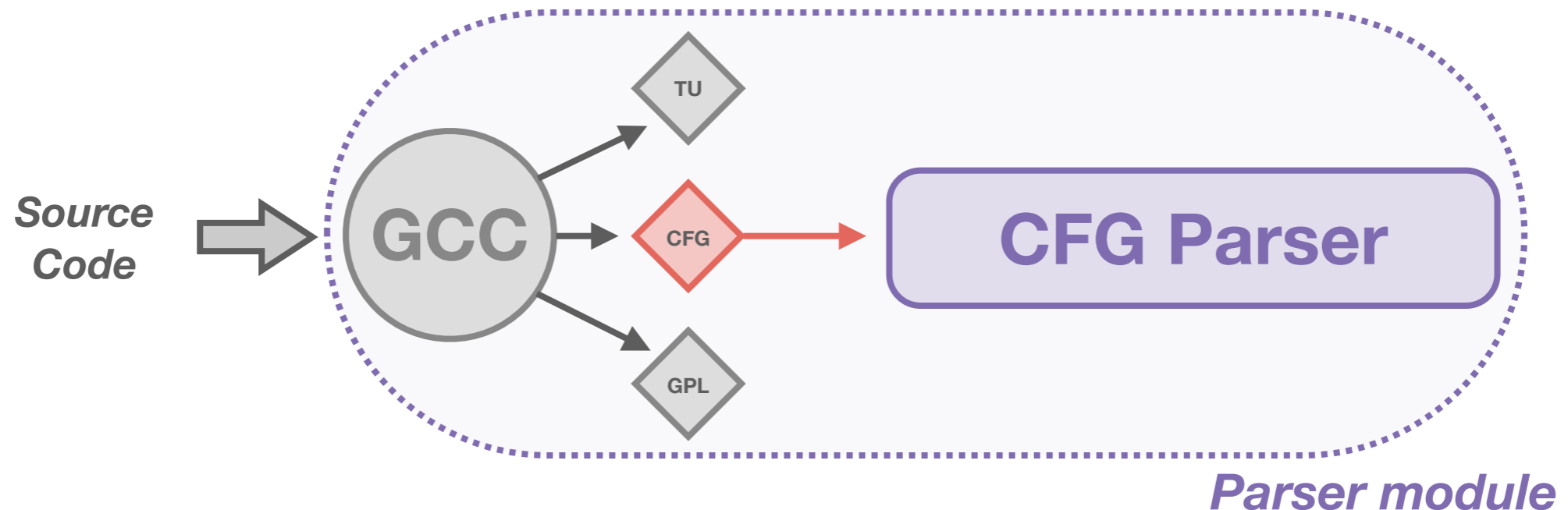
◎ Several steps :

- **Extracting relevant information** from source code
 - *Structural information*
 - *Other dedicated information*
- **Building a complete Petri net model**
- **Reducing** and optimizing the produced net
- **Analysing** the final net



Parser : Extracting Information from Source Code

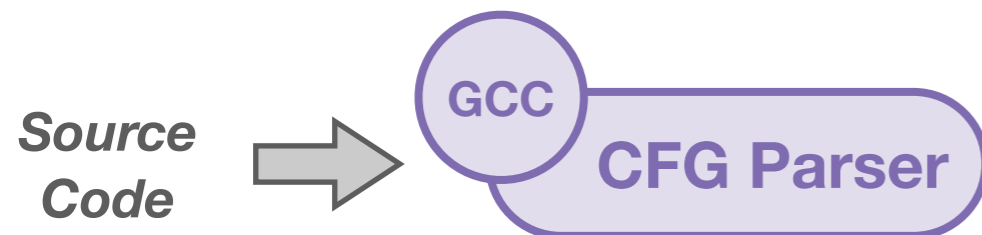
- Use of **GCC** to extract information
 - Independence from the programming language (front-ends)
 - **Control Flow Graph (CFG)** extraction



- CFG : First abstraction of the program
 - **All paths** that might be traversed during execution
 - Description in terms of **blocks**
 - Each block contains detailed instructions

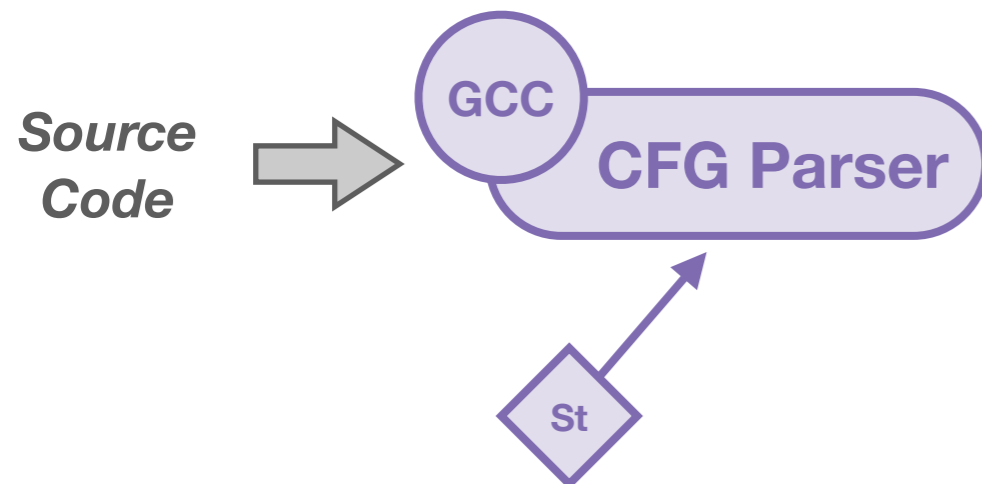
Parser : Dealing with Perspectives

- All extracted information is not relevant for analysis
 - **Structural information** is systematically extracted
- Introduce a flexible way to analyse source code : **Perspectives**
 - Each extracted information is stored into its dedicated perspective
 - Each perspective is defined using an **XML file** given to the parser
 - *Define keywords and special constructions*
 - Builder module **merges** all desired perspectives to build a unique model



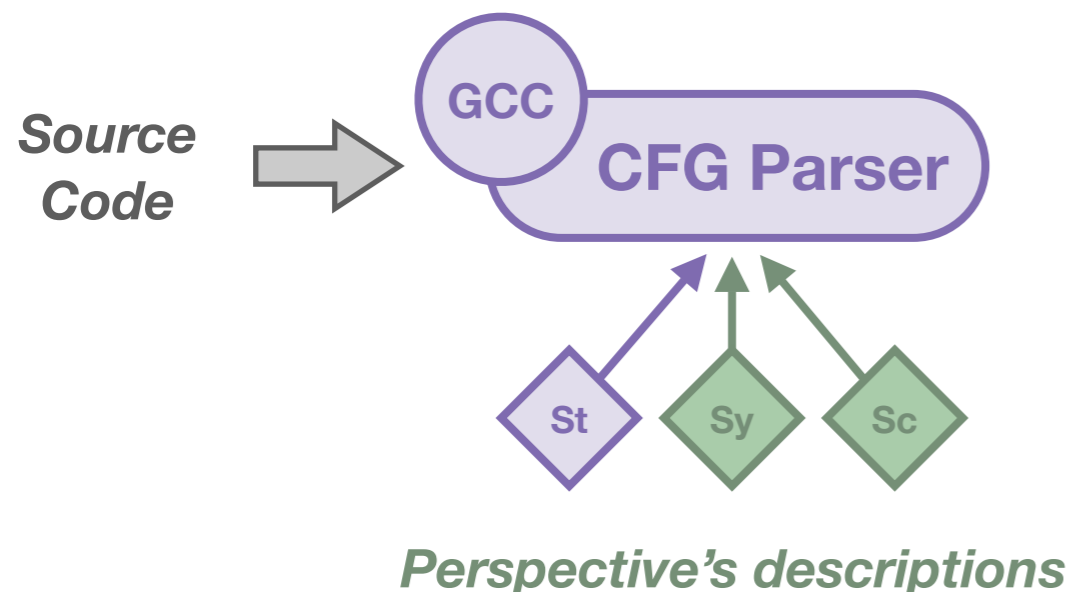
Parser : Dealing with Perspectives

- All extracted information is not relevant for analysis
 - **Structural information** is systematically extracted
- Introduce a flexible way to analyse source code : **Perspectives**
 - Each extracted information is stored into its dedicated perspective
 - Each perspective is defined using an **XML file** given to the parser
 - *Define keywords and special constructions*
 - Builder module **merges** all desired perspectives to build a unique model



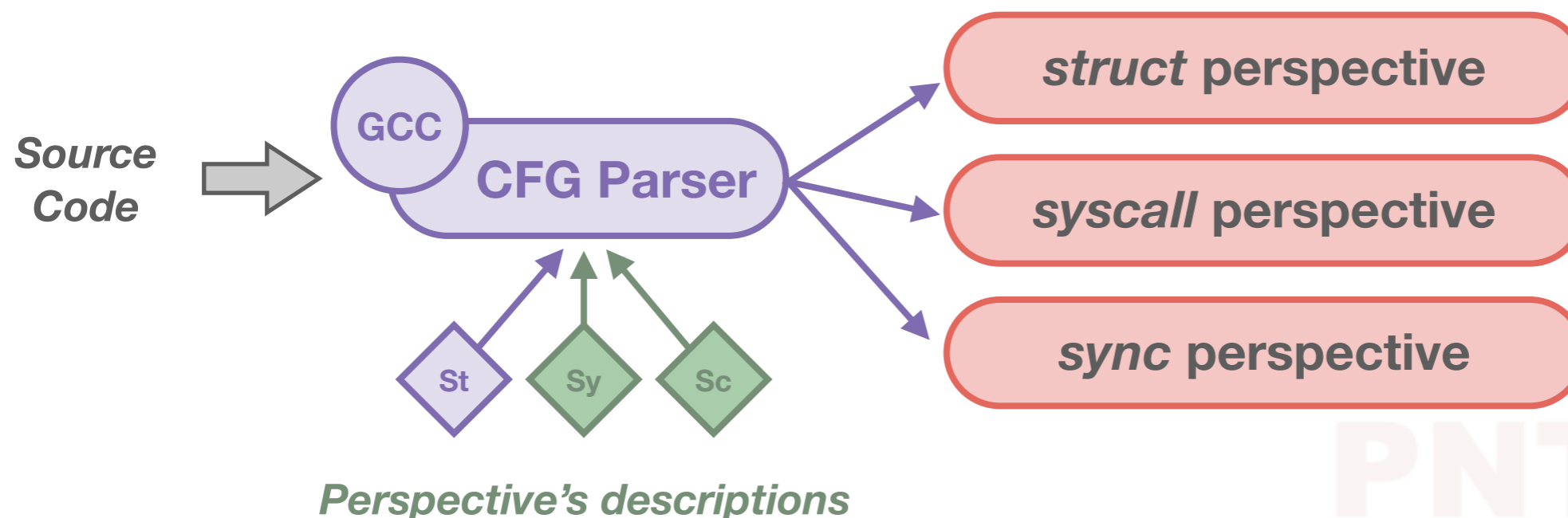
Parser : Dealing with Perspectives

- All extracted information is not relevant for analysis
 - **Structural information** is systematically extracted
- Introduce a flexible way to analyse source code : **Perspectives**
 - Each extracted information is stored into its dedicated perspective
 - Each perspective is defined using an **XML file** given to the parser
 - *Define keywords and special constructions*
 - Builder module **merges** all desired perspectives to build a unique model



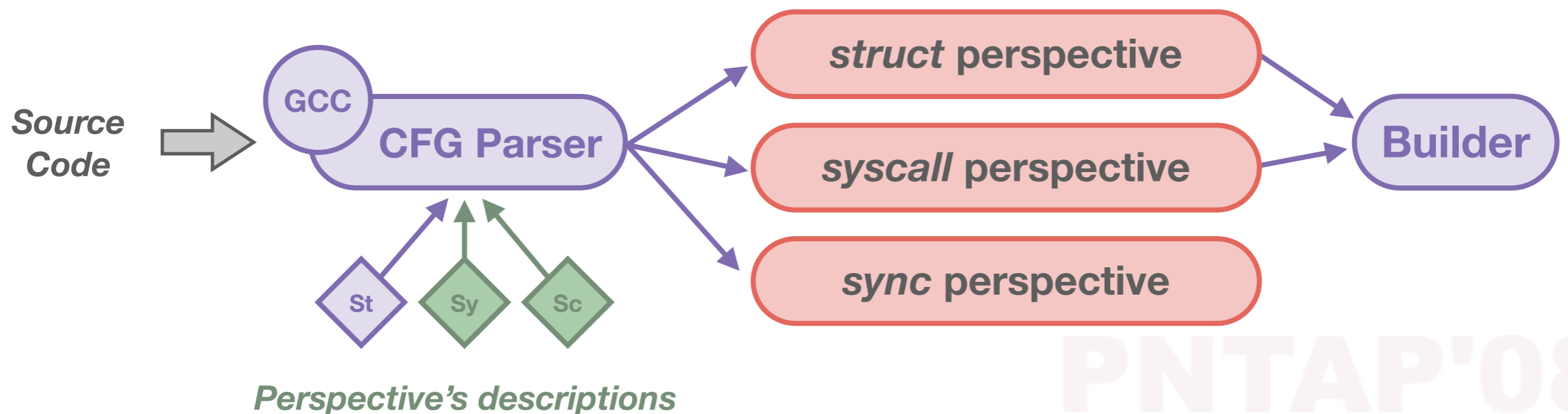
Parser : Dealing with Perspectives

- All extracted information is not relevant for analysis
 - **Structural information** is systematically extracted
- Introduce a flexible way to analyse source code : **Perspectives**
 - Each extracted information is stored into its dedicated perspective
 - Each perspective is defined using an **XML file** given to the parser
 - *Define keywords and special constructions*
 - Builder module **merges** all desired perspectives to build a unique model



Parser : Dealing with Perspectives

- All extracted information is not relevant for analysis
 - **Structural information** is systematically extracted
- Introduce a flexible way to analyse source code : **Perspectives**
 - Each extracted information is stored into its dedicated perspective
 - Each perspective is defined using an **XML file** given to the parser
 - *Define keywords and special constructions*
 - Builder module **merges** all desired perspectives to build a unique model



Readers / Writers Example

● Classical problem

- **X** readers & **Y** writers (processes) for one critical resource (size = **N**)

● 2 constraints to verify

- Exclusive Write
- Write First

```
void writer (int fd, sem_t* mutex, sem_t* rfree, sem_t* rfull) {
    int cpt = 0;
    while (1) {
        cpt++;
        sem_wait(rfree); sem_wait(mutex);
        write(fd,&cpt,sizeof(int));
        sem_post(mutex); sem_post(rfull);
    }
}
```

```
int reader (int fd, sem_t* mutex, sem_t* rfree, sem_t* rfull) {
    while (1) {
        sem_wait(mutex); sem_wait(rfull);
        read(fd,&a,sizeof(int));
        sem_post(rfree); sem_post(mutex);
    }
}
```

Parser : Choosing information to extract

● *Struct* Perspective

- Uses **CFG's embedded information**
- Locate **function's calls**

● *Syscall & Sync* Perspective

- Looks for **keywords**

```
void writer (...) {  
    int cpt = 0;  
    while (1) {  
        cpt++;  
        sem_wait(rfree); sem_wait(mutex);  
        write(fd, &cpt, sizeof(int));  
        sem_post(mutex); sem_post(rfull);  
    }  
}
```

Parser : Choosing information to extract

● *Struct* Perspective

- Uses **CFG's embedded information**
- Locate **function's calls**

● *Syscall & Sync* Perspective

- Looks for **keywords**

```
void writer (...) {
    int cpt = 0;
    while (1) {
        cpt++;
        sem_wait(rfree); sem_wait(mutex);
        write(fd,&cpt,sizeof(int));
        sem_post(mutex); sem_post(rfull);
    }
}
```

```
;; function writer
# BLOCK 2 // # PRED: ENTRY (fallthru)
  cpt = 0;
# SUCC: 3 (fallthru)

# BLOCK 3 // # PRED: 2 (fallthru) 3 (fallthru)
  D.3243 = cpt + 1;
  cpt = D.3243;
  sem_wait (rfree);
  sem_wait (mutex);
  write (fd,&cpt,4);
  sem_post (mutex);
  sem_post (rfull);
  goto <bb 3> (<L0>);
# SUCC: 3 (fallthru)
```

Parser : Choosing information to extract

● *Struct* Perspective

- Uses **CFG's embedded information**
- Locate **function's calls**

● *Syscall & Sync* Perspective

- Looks for **keywords**



Stru.dat

```
void writer (...) {
    int cpt = 0;
    while (1) {
        cpt++;
        sem_wait(rfree); sem_wait(mutex);
        write(fd,&cpt,sizeof(int));
        sem_post(mutex); sem_post(rfull);
    }
}
```

```
;; function writer
# BLOCK 2 // # PRED: ENTRY (fallthru)
  cpt = 0;
# SUCC: 3 (fallthru)

# BLOCK 3 // # PRED: 2 (fallthru) 3 (fallthru)
  D.3243 = cpt + 1;
  cpt = D.3243;
  sem_wait (rfree);
  sem_wait (mutex);
  write (fd,&cpt,4);
  sem_post (mutex);
  sem_post (rfull);
  goto <bb 3> (<L0>);
# SUCC: 3 (fallthru)
```

Parser : Choosing information to extract

● *Struct* Perspective

- Uses **CFG's embedded information**
- Locate **function's calls**

● *Syscall & Sync* Perspective

- Looks for **keywords**



```
void writer (...) {
    int cpt = 0;
    while (1) {
        cpt++;
        sem_wait(rfree); sem_wait(mutex);
        write(fd,&cpt,sizeof(int));
        sem_post(mutex); sem_post(rfull);
    }
}
```

```
;; function writer
# BLOCK 2 // # PRED: ENTRY (fallthru)
  cpt = 0;
# SUCC: 3 (fallthru)

# BLOCK 3 // # PRED: 2 (fallthru) 3 (fallthru)
  D.3243 = cpt + 1;
  cpt = D.3243;
  sem_wait (rfree);
  sem_wait (mutex);
  write (fd,&cpt,4);
  sem_post (mutex);
  sem_post (rfull);
  goto <bb 3> (<L0>);
# SUCC: 3 (fallthru)
```

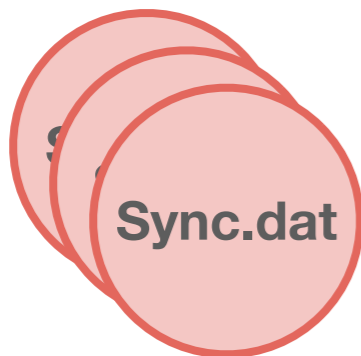
Parser : Choosing information to extract

● *Struct* Perspective

- Uses **CFG's embedded information**
- Locate **function's calls**

● *Syscall & Sync* Perspective

- Looks for **keywords**



```
void writer (...) {
    int cpt = 0;
    while (1) {
        cpt++;
        sem_wait(rfree); sem_wait(mutex);
        write(fd,&cpt,sizeof(int));
        sem_post(mutex); sem_post(rfull);
    }
}
```

```
;; function writer
# BLOCK 2 // # PRED: ENTRY (fallthru)
  cpt = 0;
# SUCC: 3 (fallthru)

# BLOCK 3 // # PRED: 2 (fallthru) 3 (fallthru)
  D.3243 = cpt + 1;
  cpt = D.3243;
  sem_wait (rfree);
  sem_wait (mutex);
  write (fd,&cpt,4);
  sem_post (mutex);
  sem_post (rfull);
  goto <bb 3> (<L0>);
# SUCC: 3 (fallthru)
```

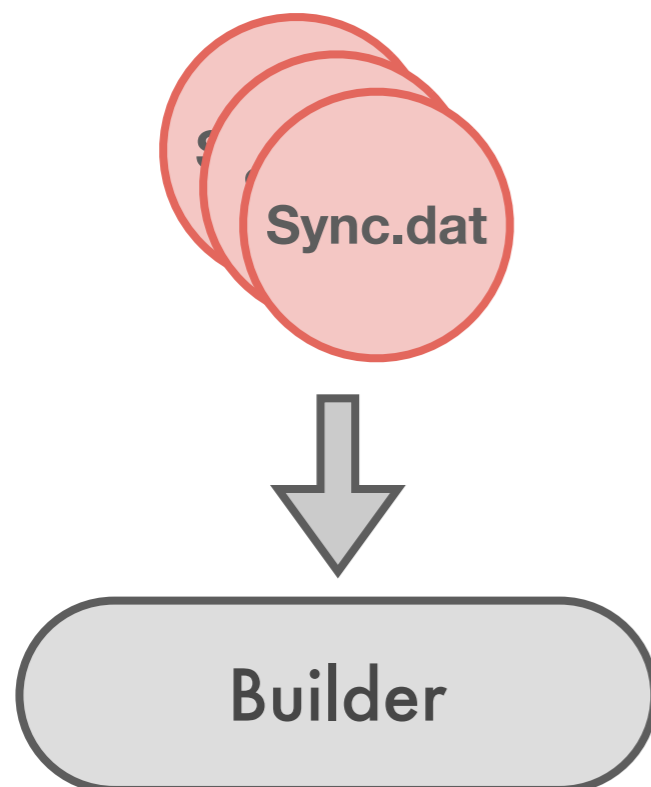
Parser : Choosing information to extract

● *Struct* Perspective

- Uses **CFG's embedded information**
- Locate **function's calls**

● *Syscall & Sync* Perspective

- Looks for **keywords**



```
void writer (...) {
    int cpt = 0;
    while (1) {
        cpt++;
        sem_wait(rfree); sem_wait(mutex);
        write(fd,&cpt,sizeof(int));
        sem_post(mutex); sem_post(rfull);
    }
}
```

```
;; function writer
# BLOCK 2 // # PRED: ENTRY (fallthru)
  cpt = 0;
# SUCC: 3 (fallthru)

# BLOCK 3 // # PRED: 2 (fallthru) 3 (fallthru)
  D.3243 = cpt + 1;
  cpt = D.3243;
  sem_wait (rfree);
  sem_wait (mutex);
  write (fd,&cpt,4);
  sem_post (mutex);
  sem_post (rfull);
  goto <bb 3> (<L0>);
# SUCC: 3 (fallthru)
```

Builder : Establishing the Structural Model

- Building the structural model
 - Use as **skeleton** for other perspectives

Stru.dat

```
;; function writer
# BLOCK 2
# PRED: ENTRY (fallthru)
# SUCC: 3 (fallthru)
# BLOCK 3
# PRED: 2 (fallthru) 3 (fallthru)
# SUCC: 3 (fallthru)
```

- Set of **6 dedicated rules**
 - Function / Blocks transformation
 - Links between blocks (x2)
 - Function calls (x2)

Builder : Establishing the Structural Model

● Building the structural model

- Use as **skeleton** for other perspectives

```
;; function writer
# BLOCK 2
# PRED: ENTRY (fallthru)
# SUCC: 3 (fallthru)
# BLOCK 3
# PRED: 2 (fallthru) 3 (fallthru)
# SUCC: 3 (fallthru)
```

Stru.dat

○ 1_Entry

● Set of **6 dedicated rules**

- Function / Blocks transformation
- Links between blocks (x2)
- Function calls (x2)

○ 1_Exit

Builder : Establishing the Structural Model

● Building the structural model

- Use as **skeleton** for other perspectives

Stru.dat

```
;; function writer
# BLOCK 2
# PRED: ENTRY (fallthru)
# SUCC: 3 (fallthru)
# BLOCK 3
# PRED: 2 (fallthru) 3 (fallthru)
# SUCC: 3 (fallthru)
```

○ 1_Entry

○ 1_2

○ 1_3

○ 1_Exit

● Set of **6 dedicated rules**

- Function / Blocks transformation
- Links between blocks (x2)
- Function calls (x2)

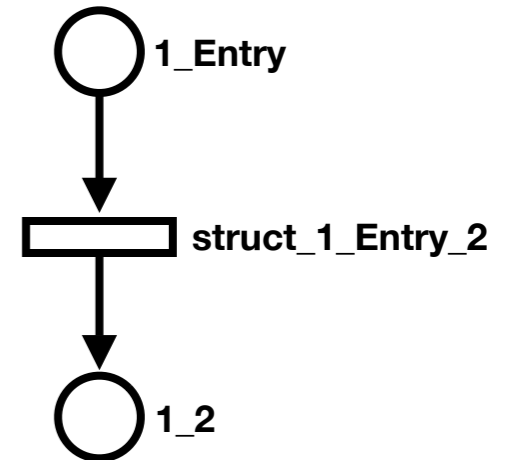
Builder : Establishing the Structural Model

● Building the structural model

- Use as **skeleton** for other perspectives

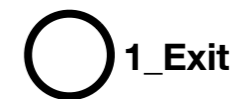
Stru.dat

```
;; function writer
# BLOCK 2
# PRED: ENTRY (fallthru)
# SUCC: 3 (fallthru)
# BLOCK 3
# PRED: 2 (fallthru) 3 (fallthru)
# SUCC: 3 (fallthru)
```



● Set of **6 dedicated rules**

- Function / Blocks transformation
- Links between blocks (x2)
- Function calls (x2)



Builder : Establishing the Structural Model

- Building the structural model
 - Use as **skeleton** for other perspectives

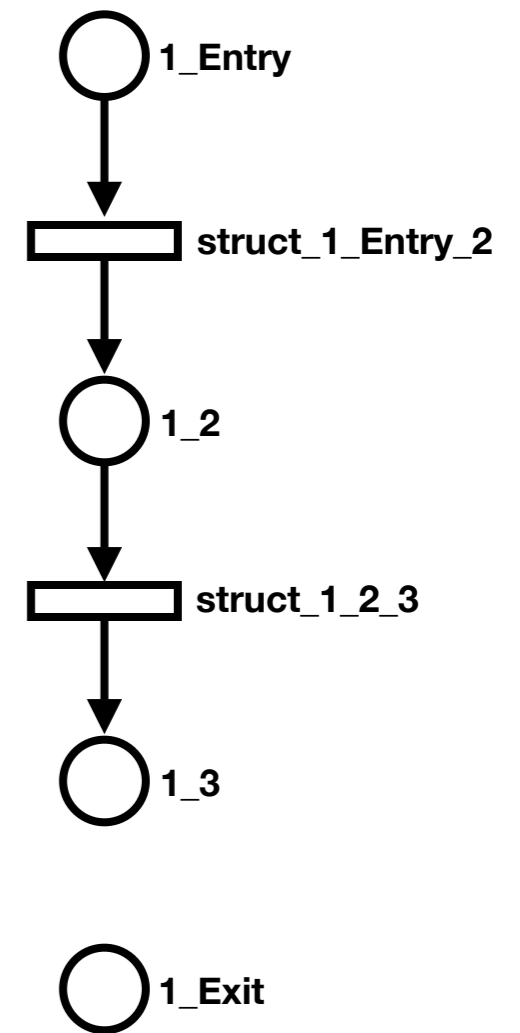
```

;; function writer
# BLOCK 2
# PRED: ENTRY (fallthru)
# SUCC: 3 (fallthru)
# BLOCK 3
# PRED: 2 (fallthru) 3 (fallthru)
# SUCC: 3 (fallthru)

```

Stru.dat

- Set of **6 dedicated rules**
 - Function / Blocks transformation
 - Links between blocks (x2)
 - Function calls (x2)



Builder : Establishing the Structural Model

- Building the structural model
 - Use as **skeleton** for other perspectives

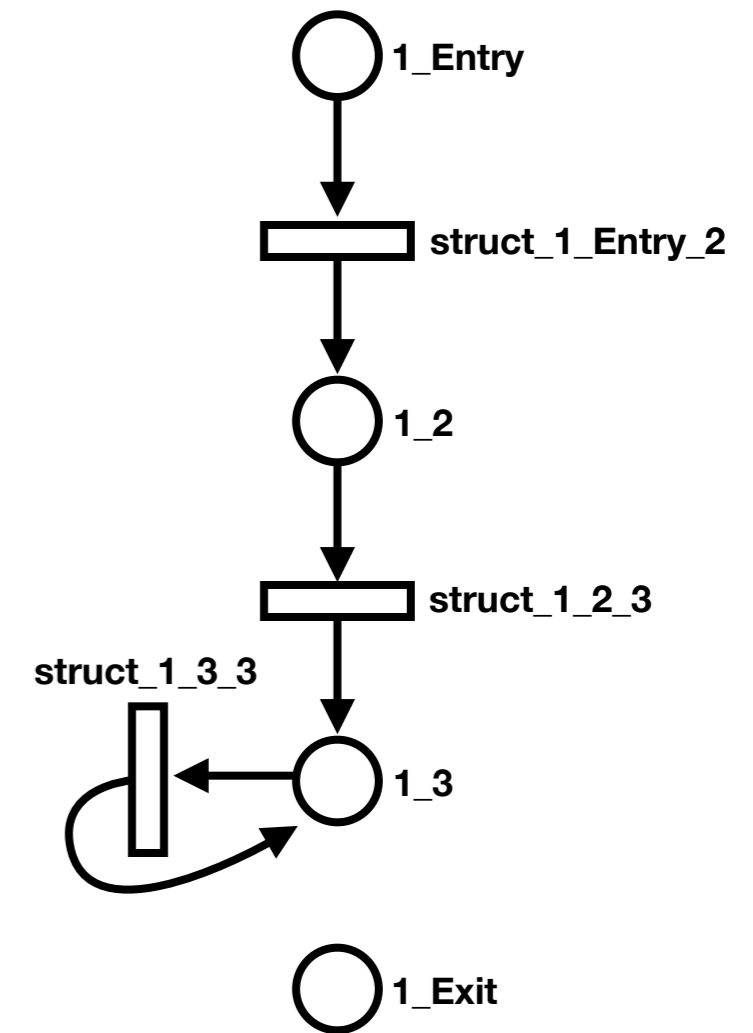
```

;; function writer
# BLOCK 2
# PRED: ENTRY (fallthru)
# SUCC: 3 (fallthru)
# BLOCK 3
# PRED: 2 (fallthru) 3 (fallthru)
# SUCC: 3 (fallthru)

```

Stru.dat

- Set of **6 dedicated rules**
 - Function / Blocks transformation
 - Links between blocks (x2)
 - Function calls (x2)



Builder : Establishing the Structural Model

- Building the structural model
 - Use as **skeleton** for other perspectives

```

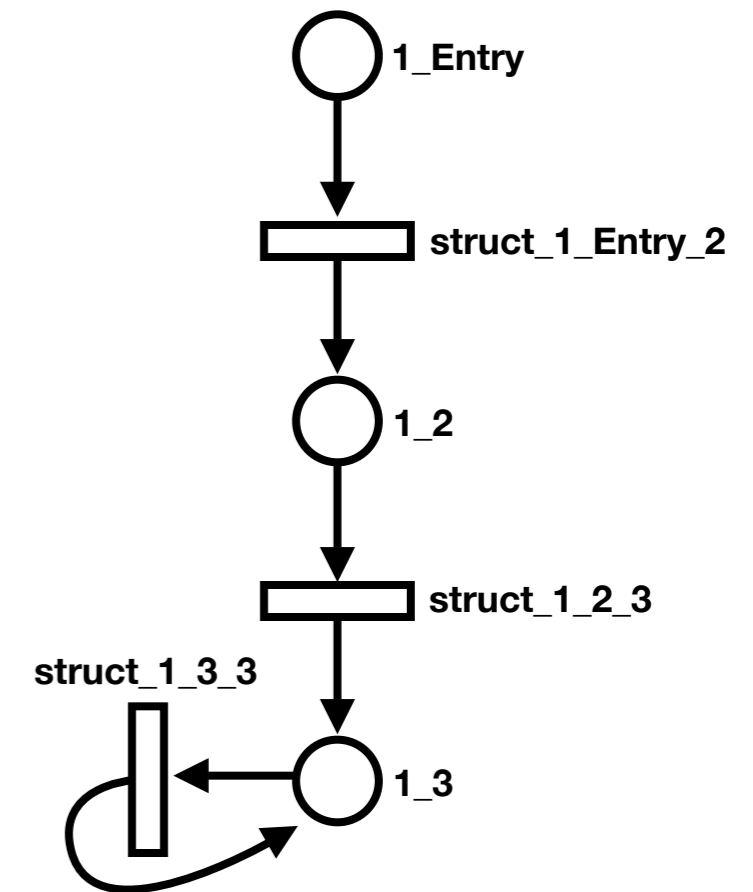
;; function writer
# BLOCK 2
# PRED: ENTRY (fallthru)
# SUCC: 3 (fallthru)
# BLOCK 3
# PRED: 2 (fallthru) 3 (fallthru)
# SUCC: 3 (fallthru)

```

Stru.dat

- Set of **6 dedicated rules**

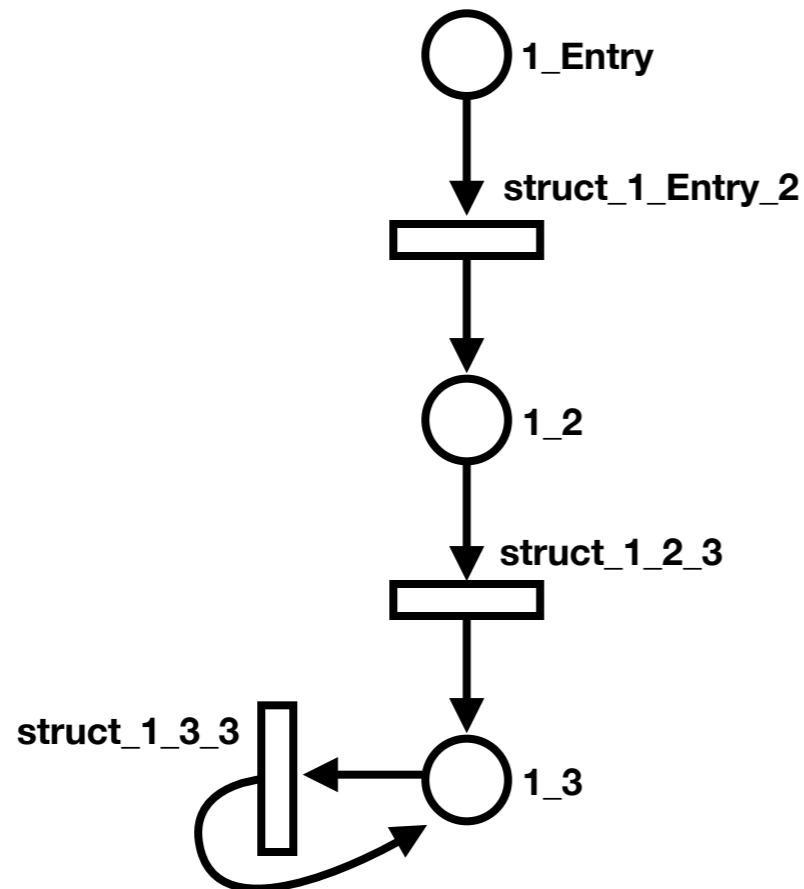
- Function / Blocks transformation
- Links between blocks (x2)
- Function calls (x2)



Builder : Adding Information from Perspectives

● Associate Petri subnets to reference places

- All subnets must have an Entry and Exit place
- “Keep an eye” on instruction’s position inside block



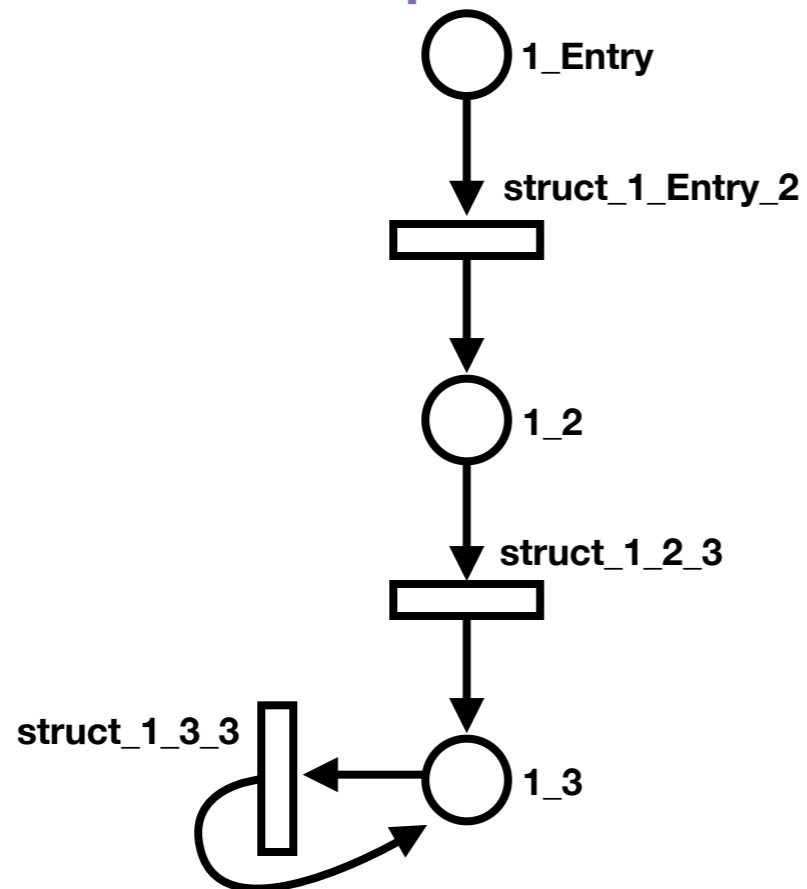
Builder : Adding Information from Perspectives

◎ Associate Petri subnets to reference places

- All subnets must have an Entry and Exit place
- “Keep an eye” on instruction’s position inside block

Sys.dat

```
# BLOCK 3
4. write(fd, &cpt, 4);
```

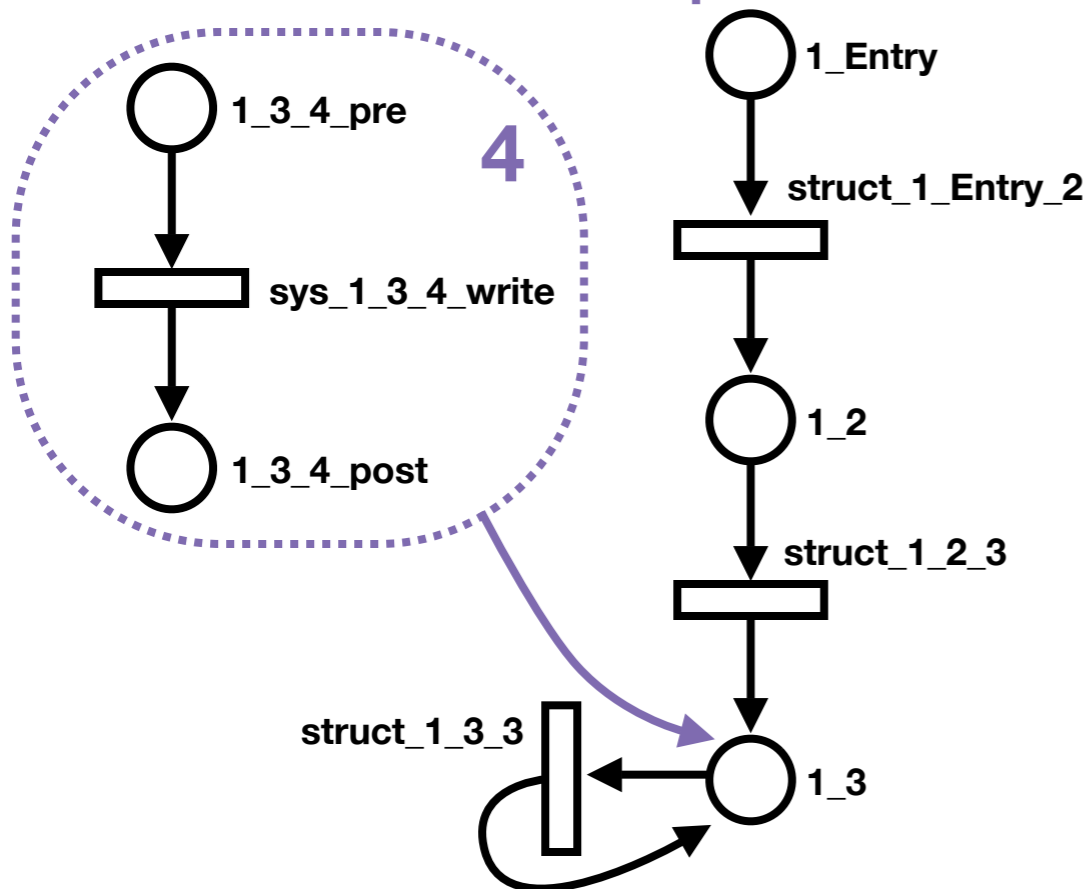


Builder : Adding Information from Perspectives

- Associate Petri subnets to reference places
 - All subnets must have an Entry and Exit place
 - “Keep an eye” on instruction’s position inside block

Sys.dat

```
# BLOCK 3
4. write(fd, &cpt, 4);
```



Builder : Adding Information from Perspectives

● Associate Petri subnets to reference places

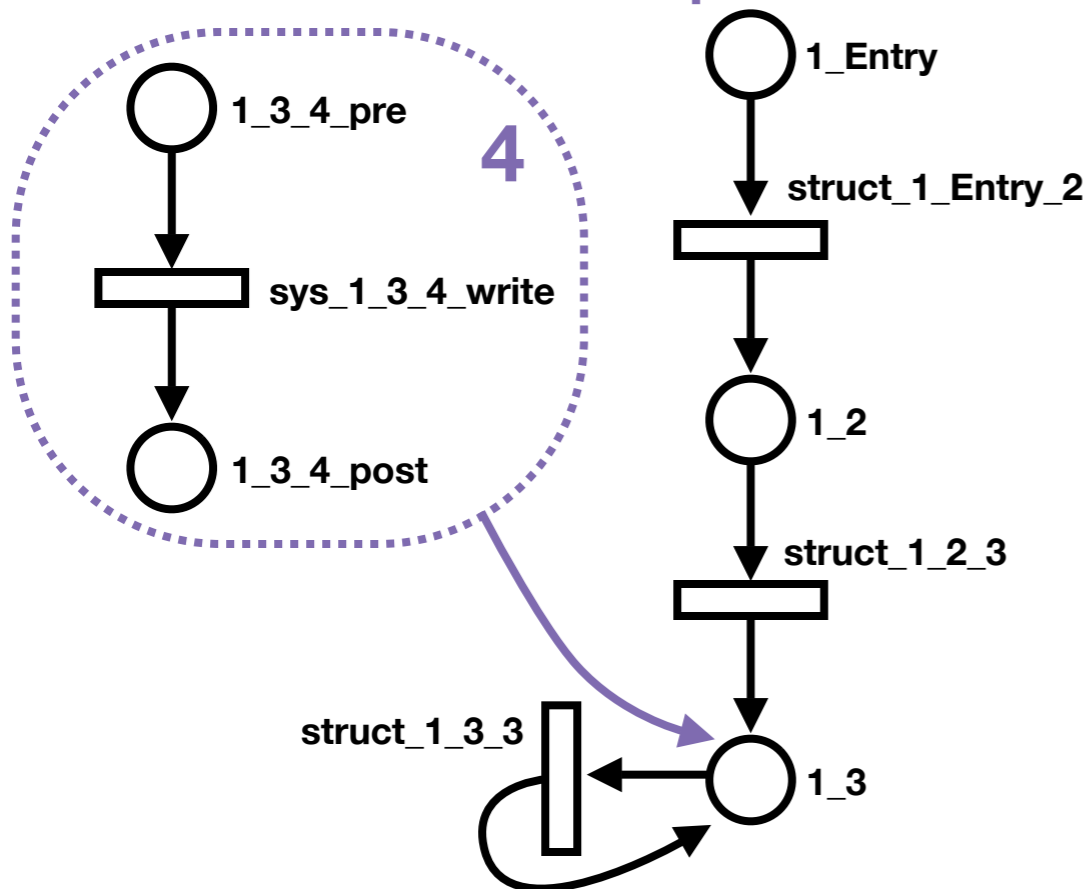
- All subnets must have an Entry and Exit place
- “Keep an eye” on instruction’s position inside block

Syn.dat

```
# BLOCK 3
2. sem_wait (rfree);
3. sem_wait (mutex);
5. sem_post (mutex);
6. sem_post (rfull);
```

Sys.dat

```
# BLOCK 3
4. write(fd, &cpt, 4);
```



Builder : Adding Information from Perspectives

● Associate Petri subnets to reference places

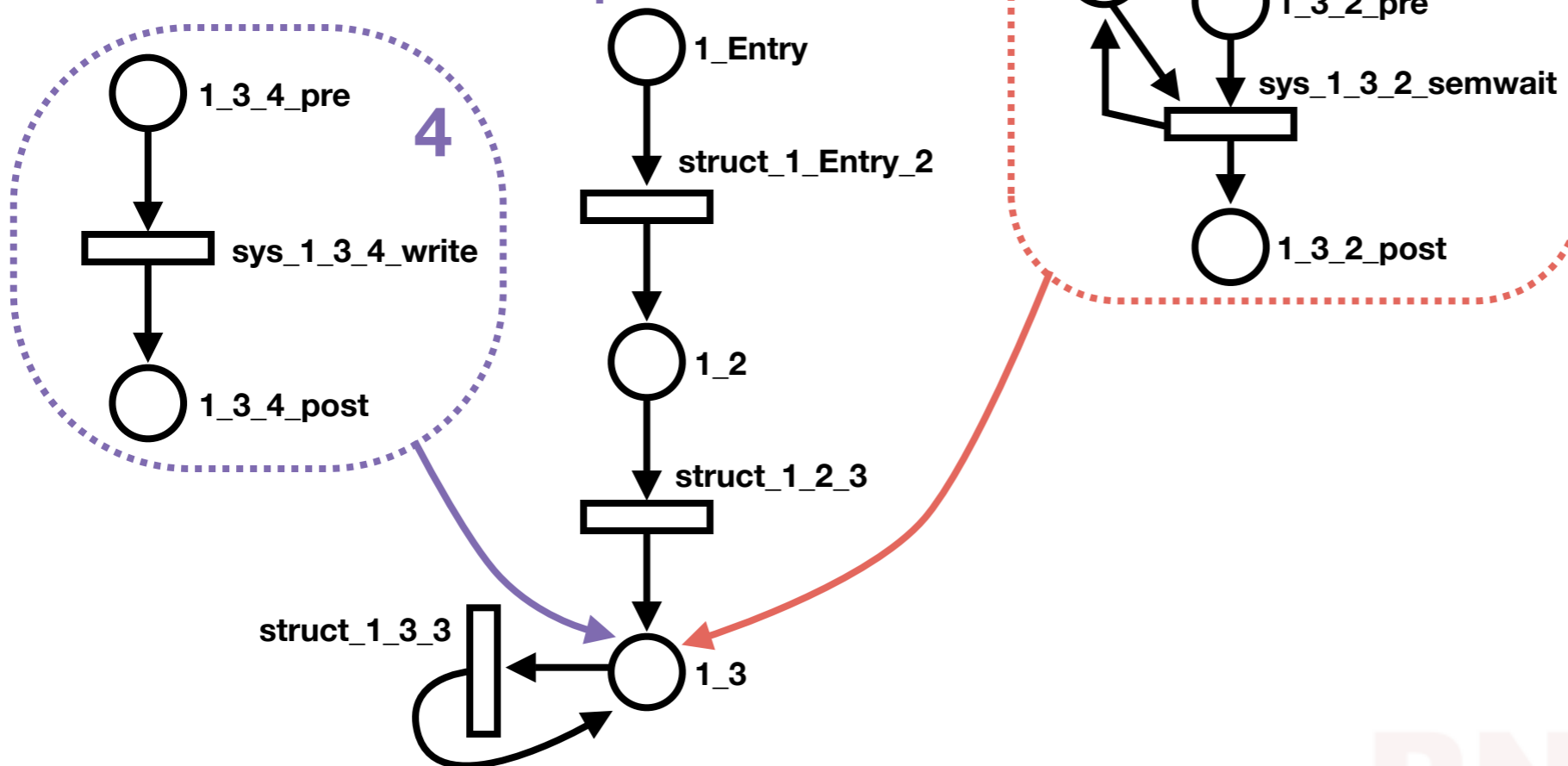
- All subnets must have an Entry and Exit place
- “Keep an eye” on instruction’s position inside block

Syn.dat

```
# BLOCK 3
2. sem_wait (rfree);
3. sem_wait (mutex);
5. sem_post (mutex);
6. sem_post (rfull);
```

Sys.dat

```
# BLOCK 3
4. write(fd, &cpt, 4);
```



Builder : Adding Information from Perspectives

Associate Petri subnets to reference places

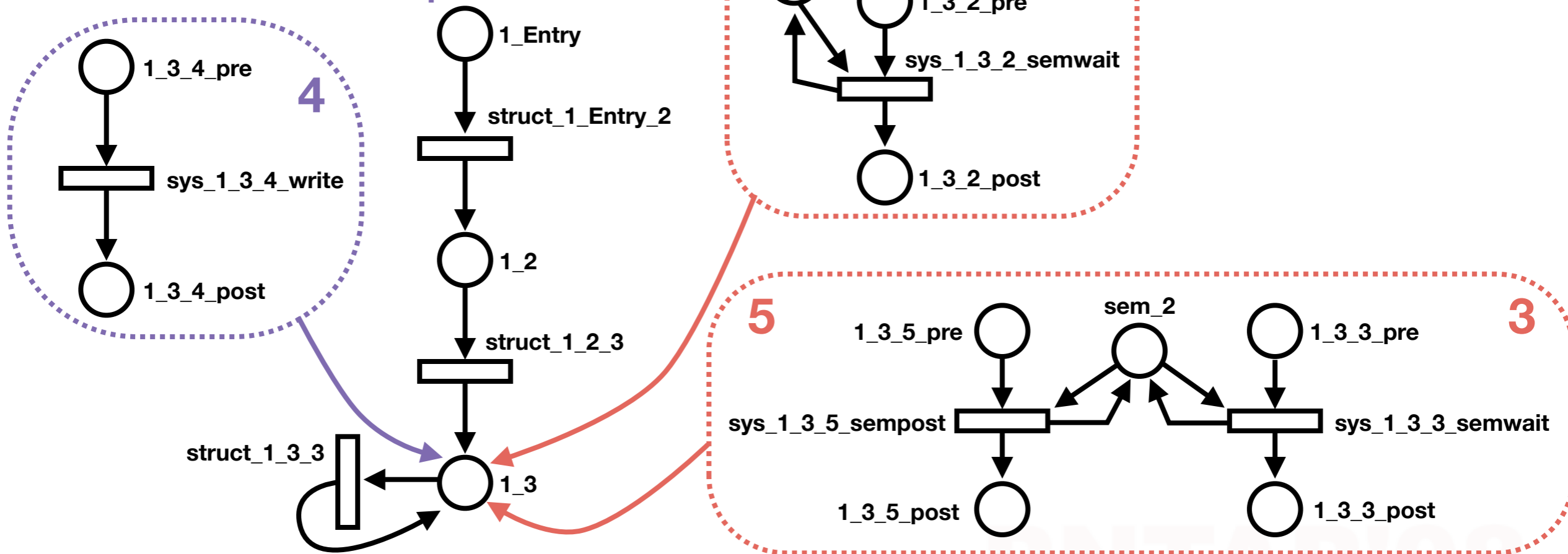
- All subnets must have an Entry and Exit place
- “Keep an eye” on instruction’s position inside block

Syn.dat

```
# BLOCK 3
2. sem_wait (rfree);
3. sem_wait (mutex);
5. sem_post (mutex);
6. sem_post (rfull);
```

Sys.dat

```
# BLOCK 3
4. write(fd, &cpt, 4);
```



Builder : Adding Information from Perspectives

● Associate Petri subnets to reference places

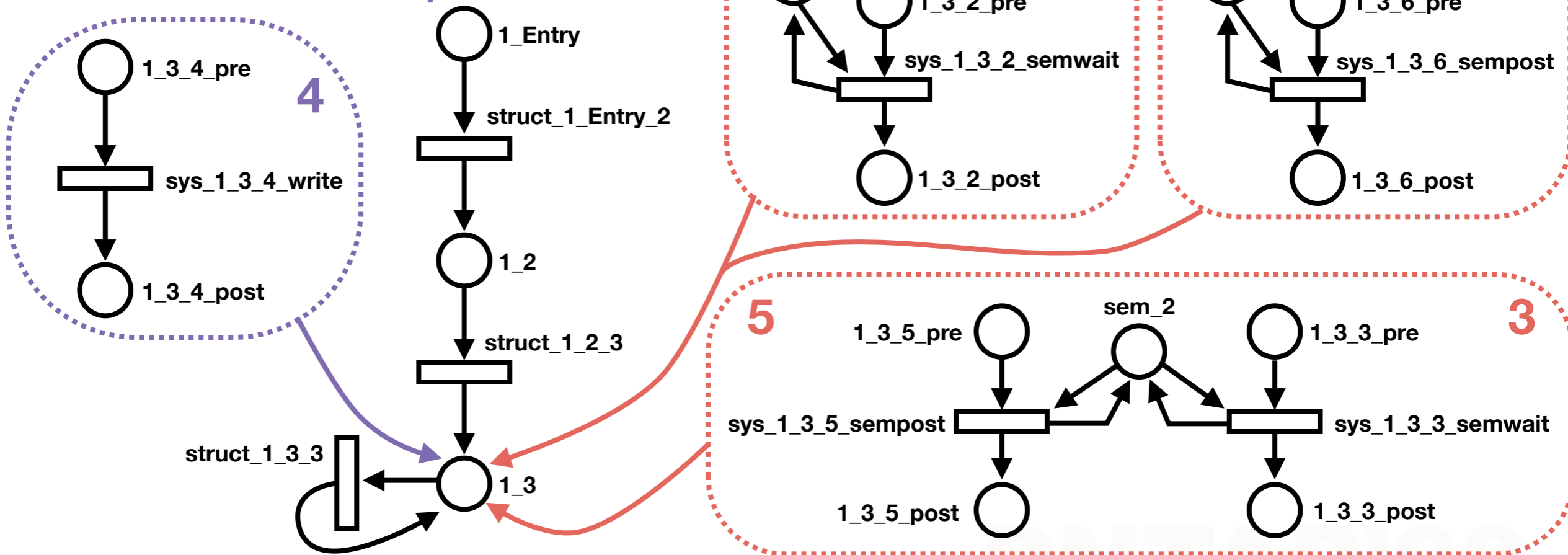
- All subnets must have an Entry and Exit place
- “Keep an eye” on instruction’s position inside block

Syn.dat

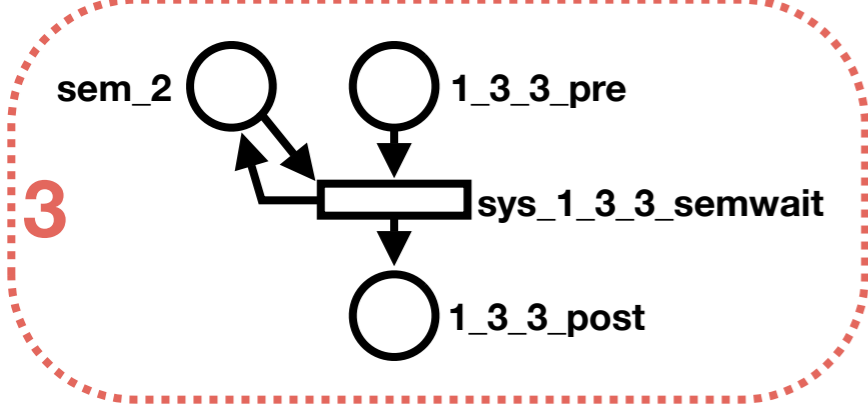
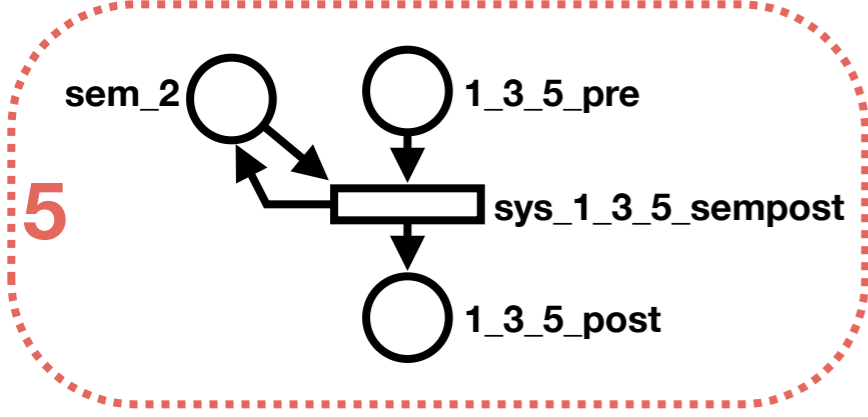
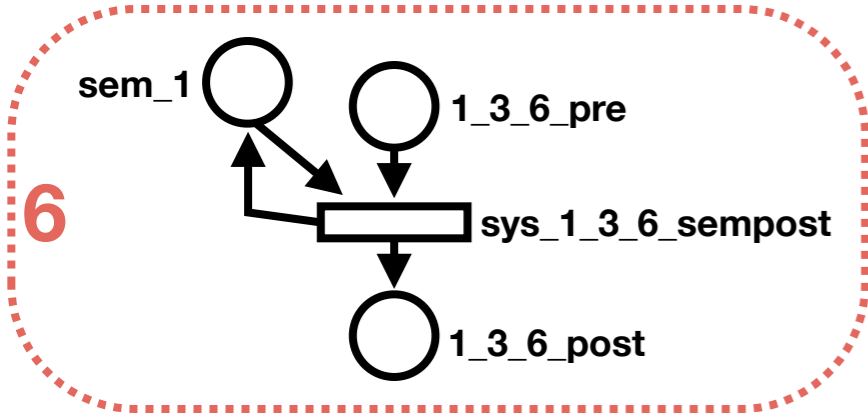
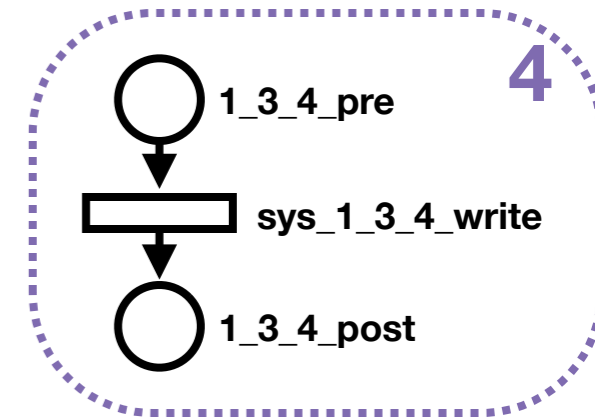
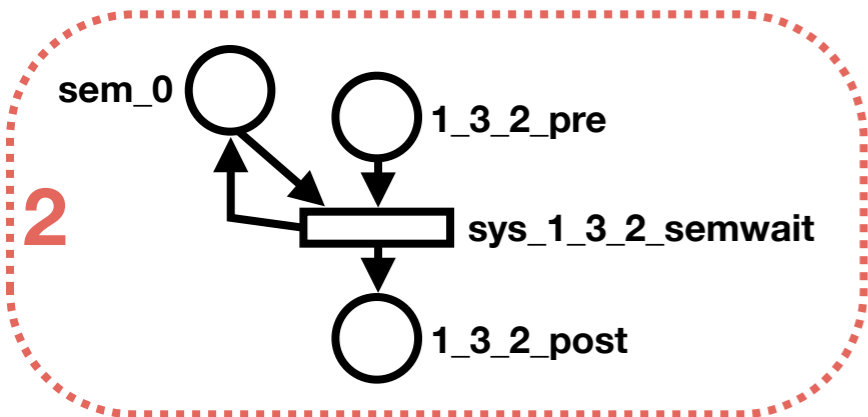
```
# BLOCK 3
2. sem_wait (rfree);
3. sem_wait (mutex);
5. sem_post (mutex);
6. sem_post (rfull);
```

Sys.dat

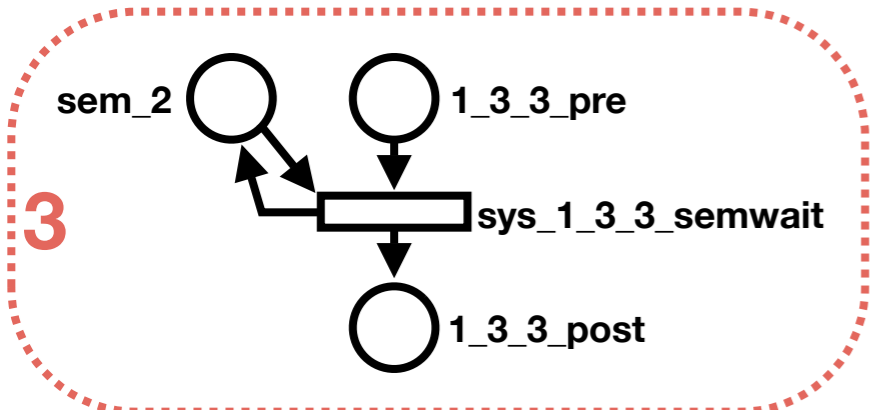
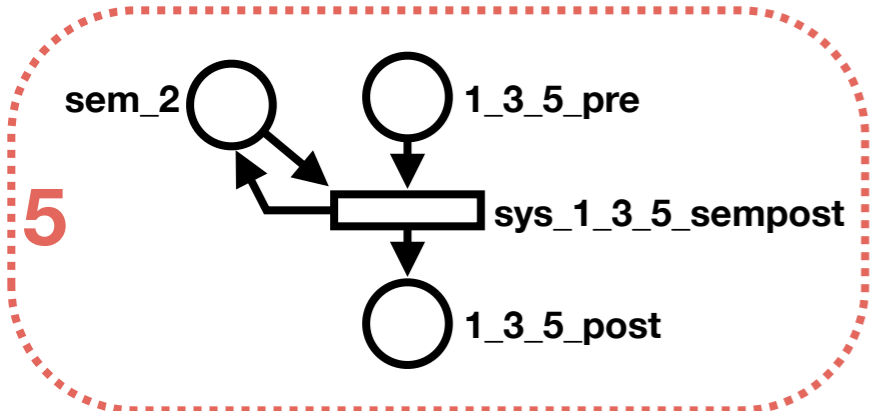
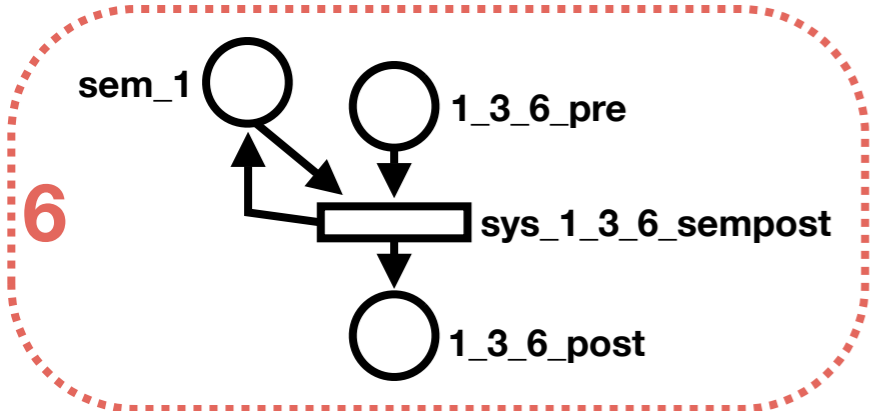
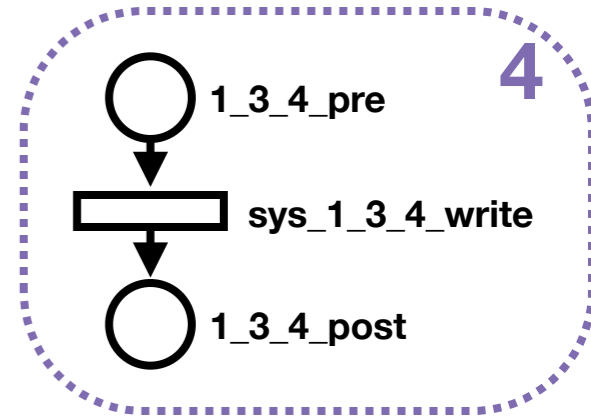
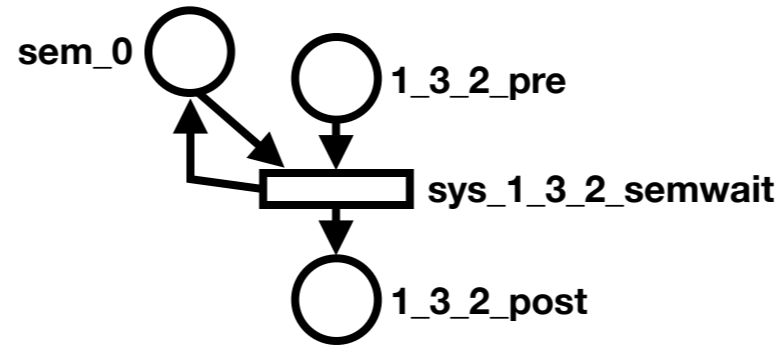
```
# BLOCK 3
4. write(fd, &cpt, 4);
```



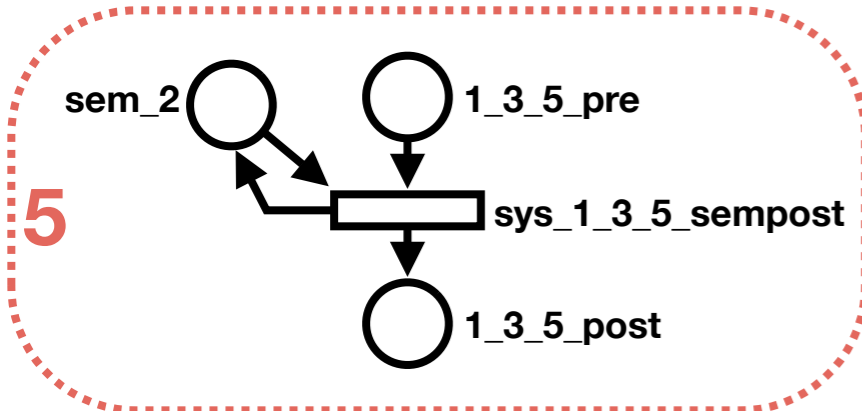
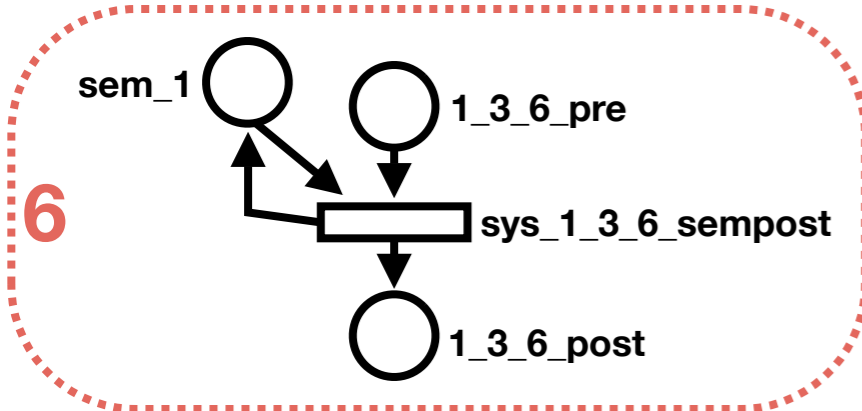
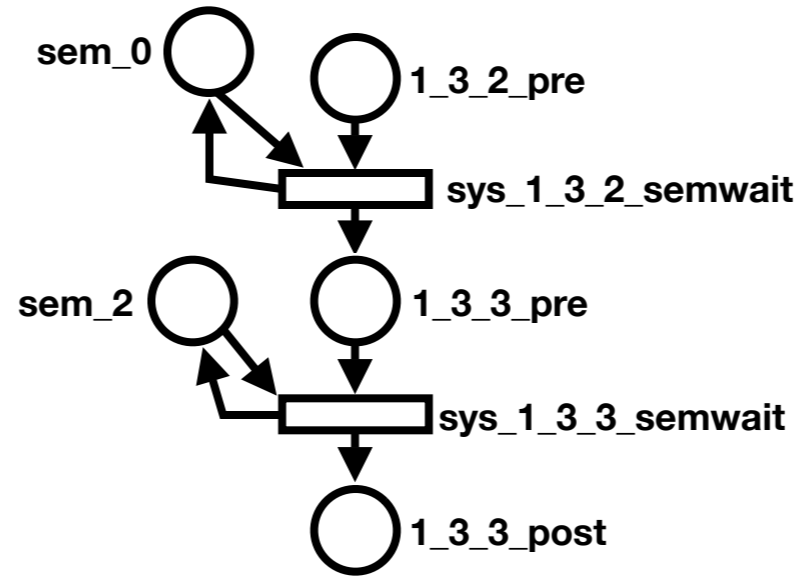
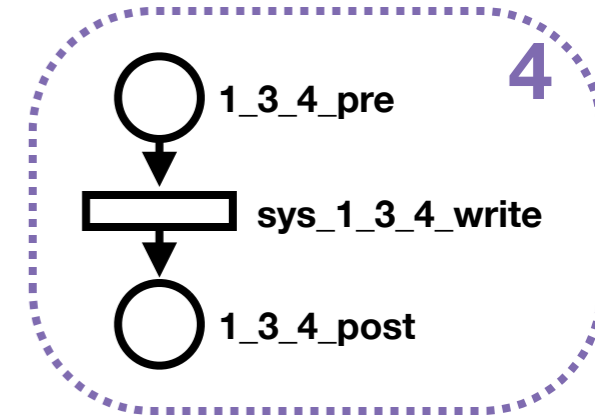
Builder : Merging selected perspectives



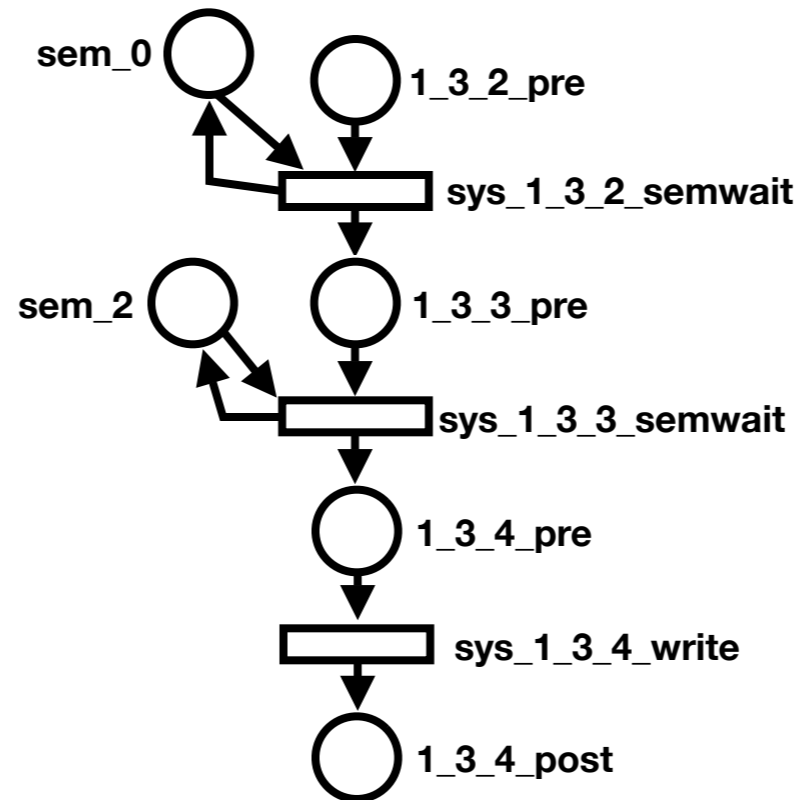
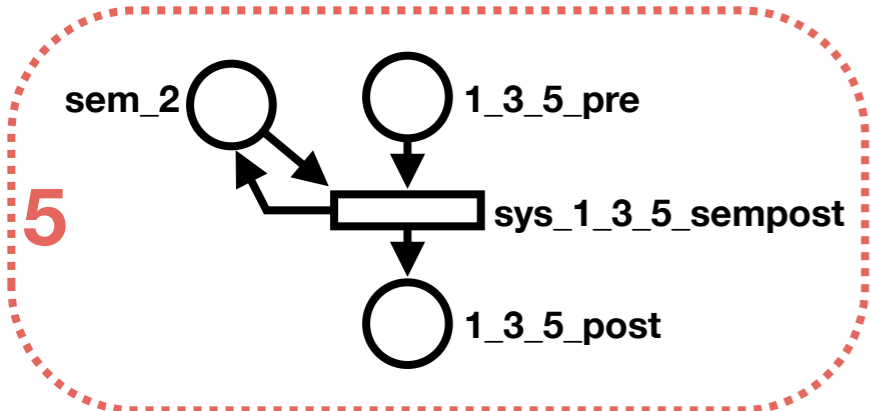
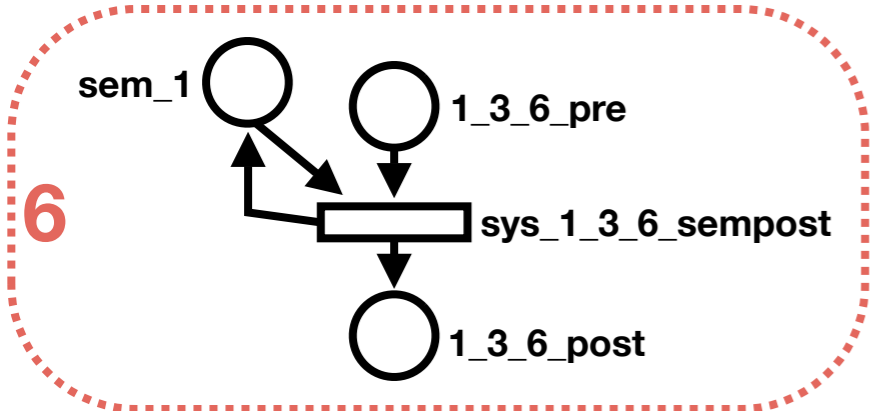
Builder : Merging selected perspectives



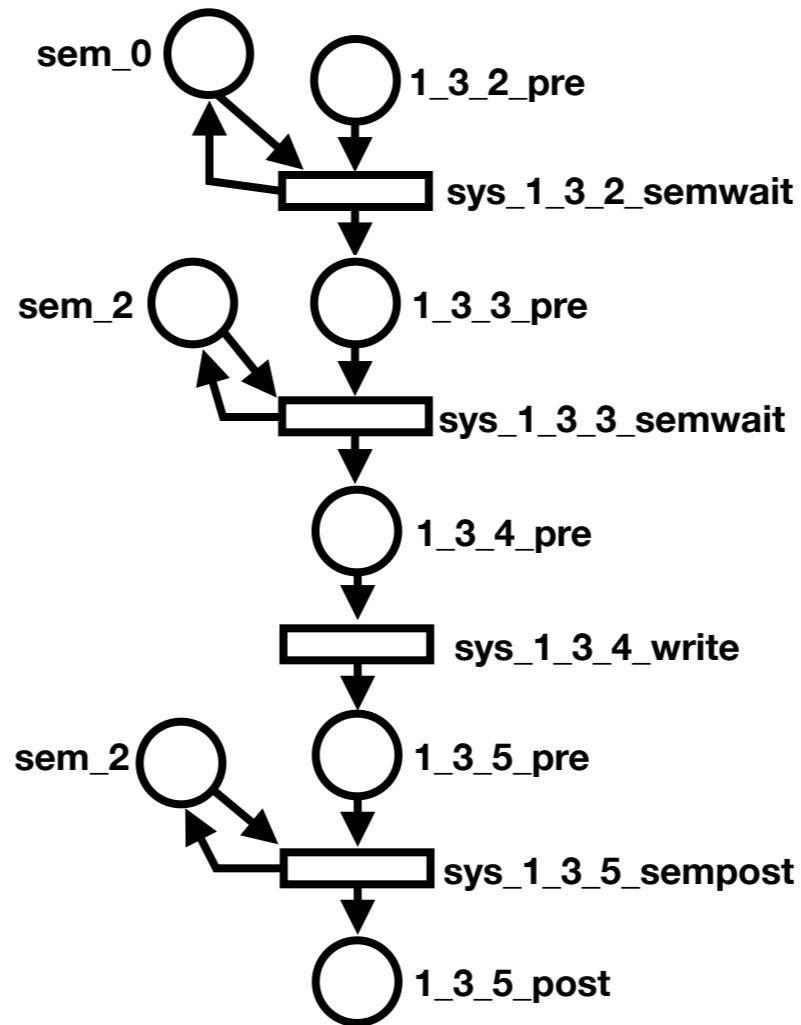
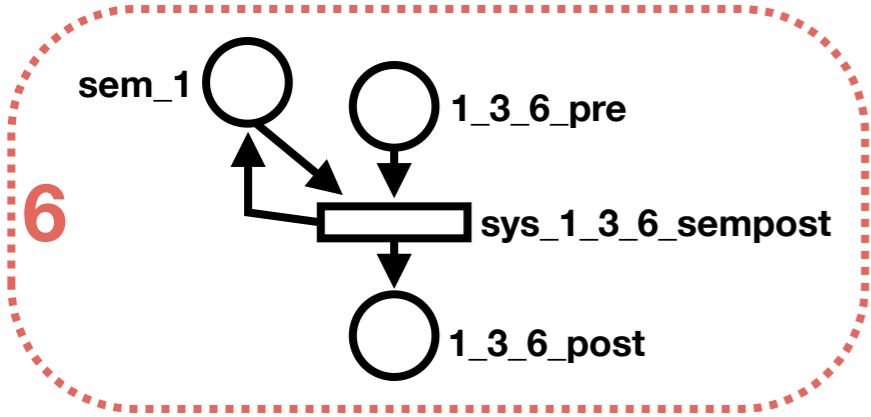
Builder : Merging selected perspectives



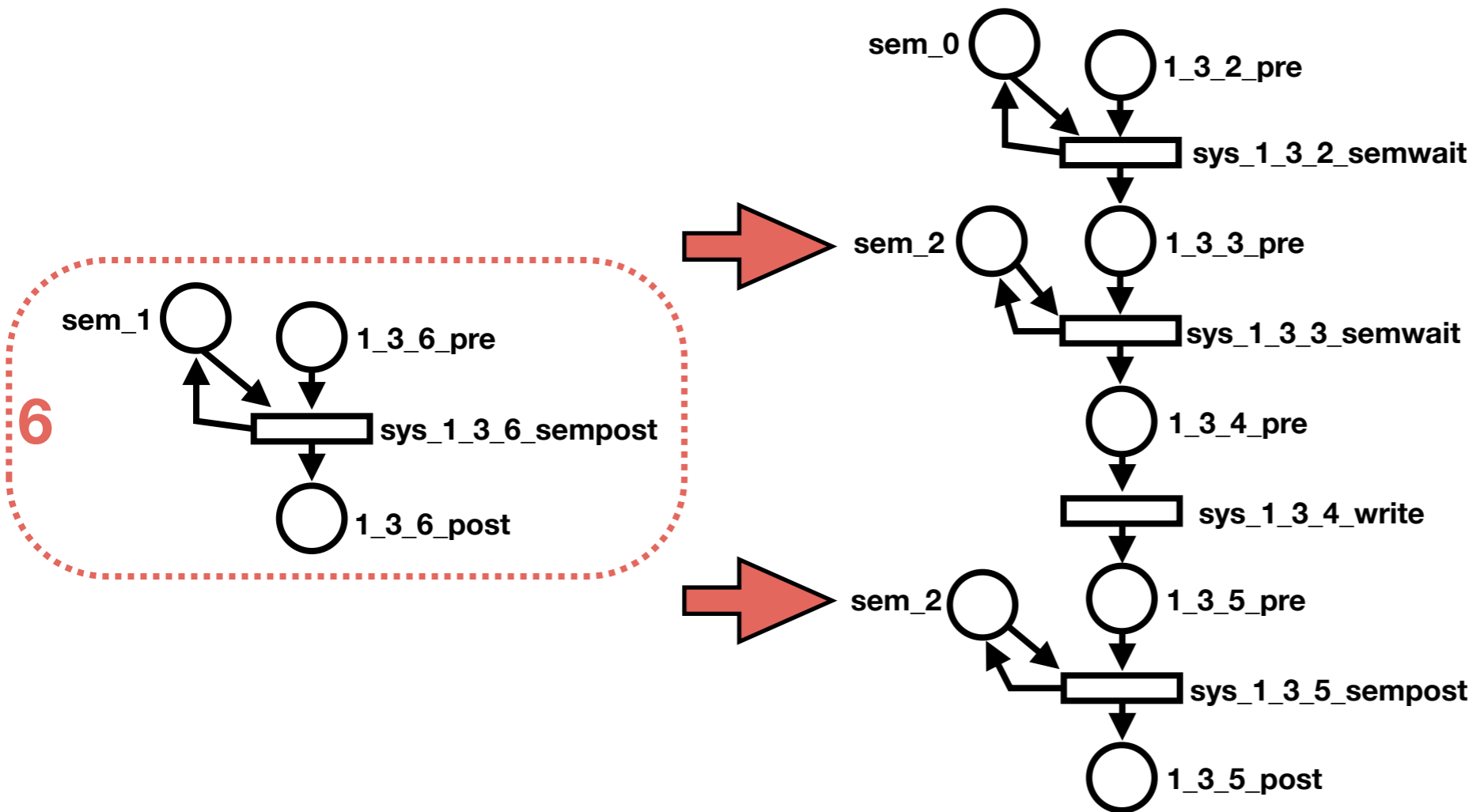
Builder : Merging selected perspectives



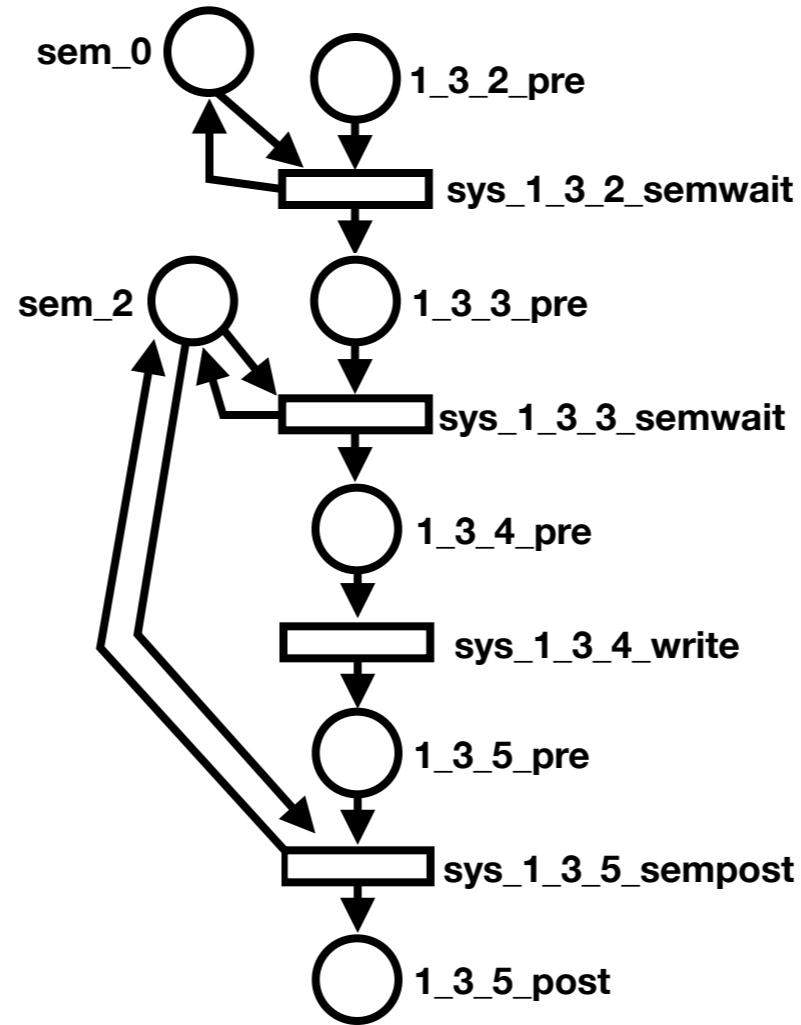
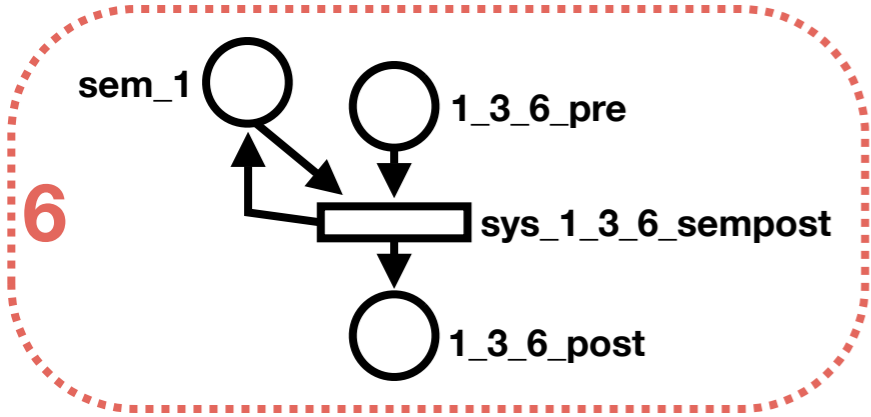
Builder : Merging selected perspectives



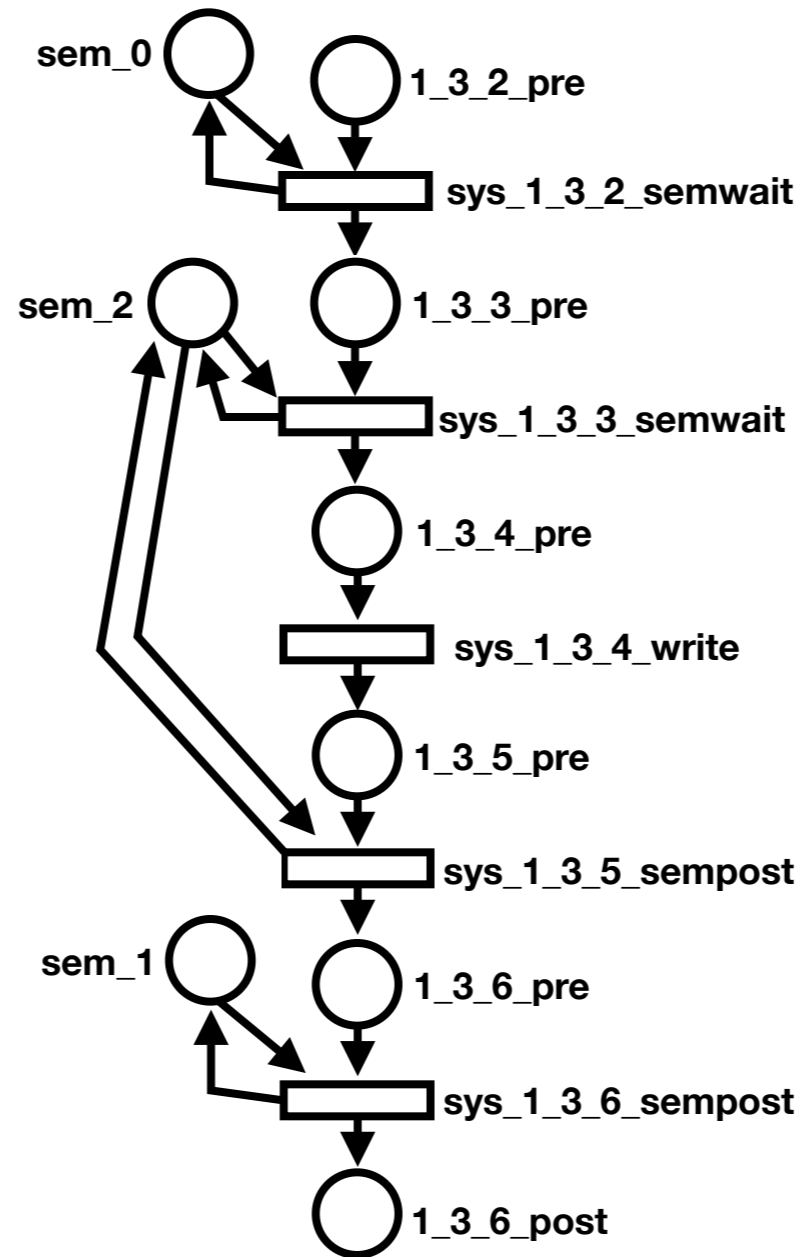
Builder : Merging selected perspectives



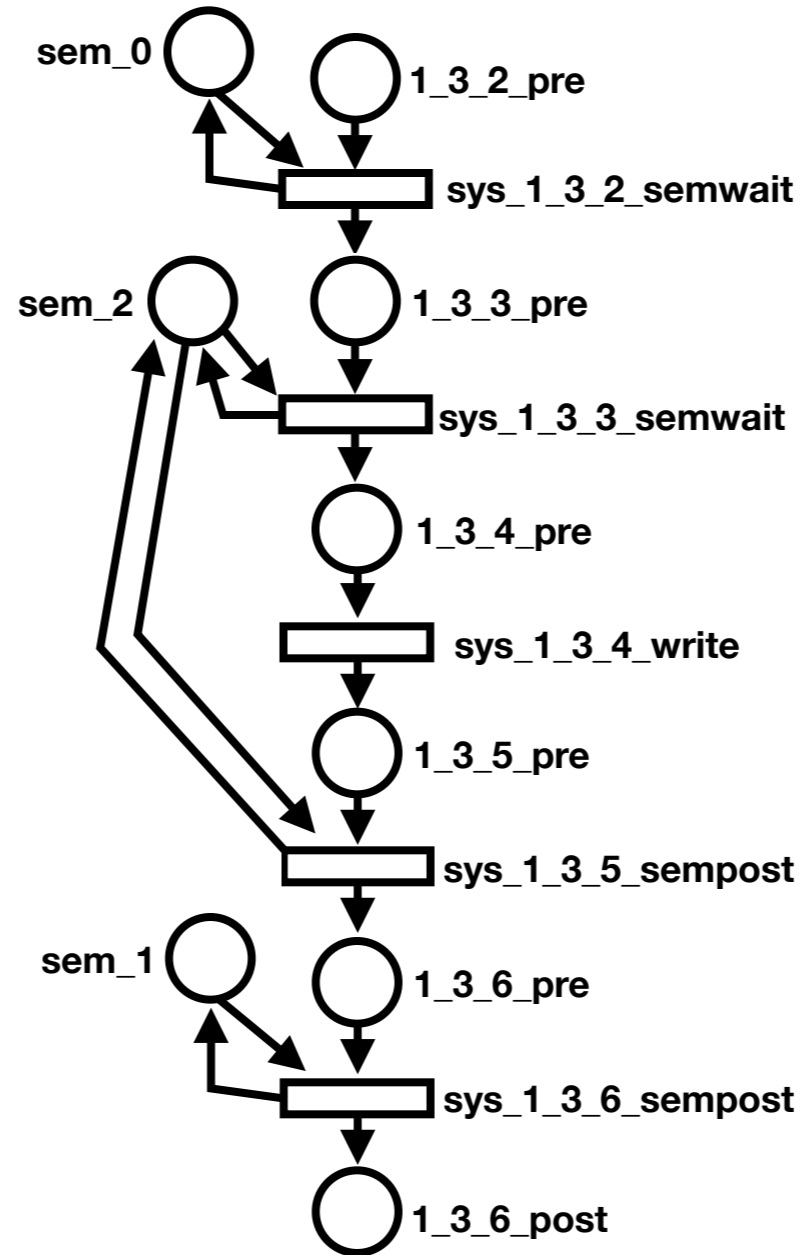
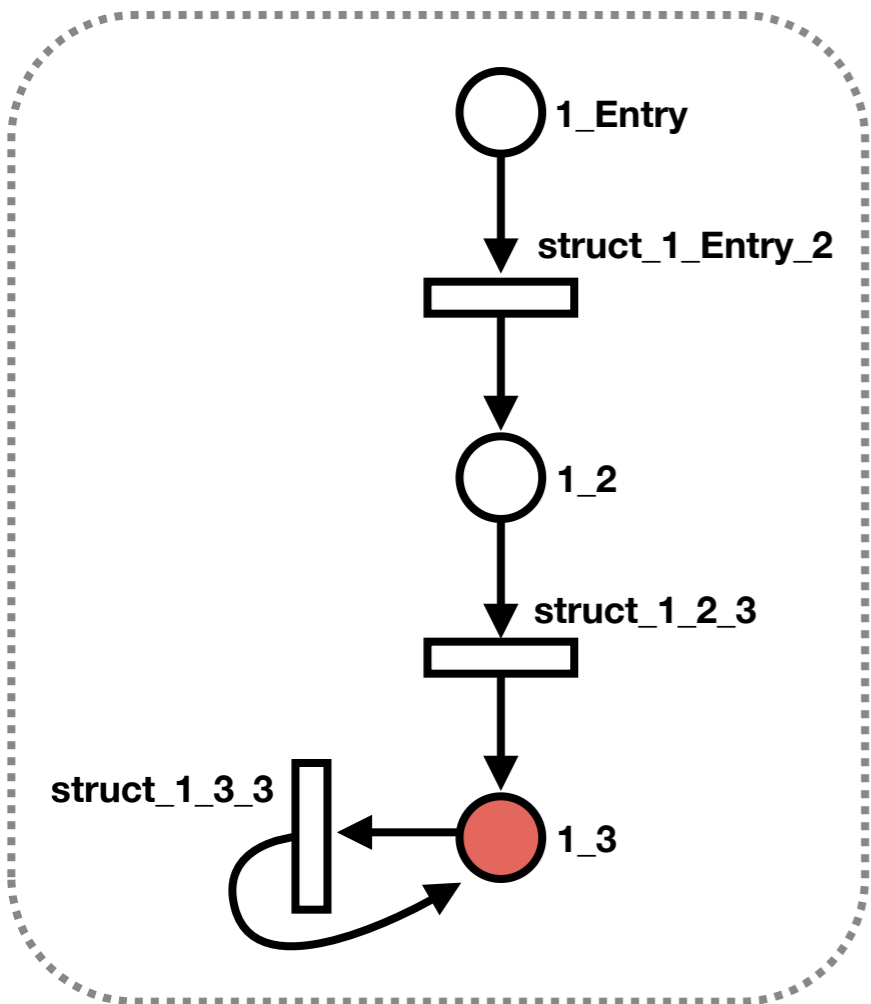
Builder : Merging selected perspectives



Builder : Merging selected perspectives

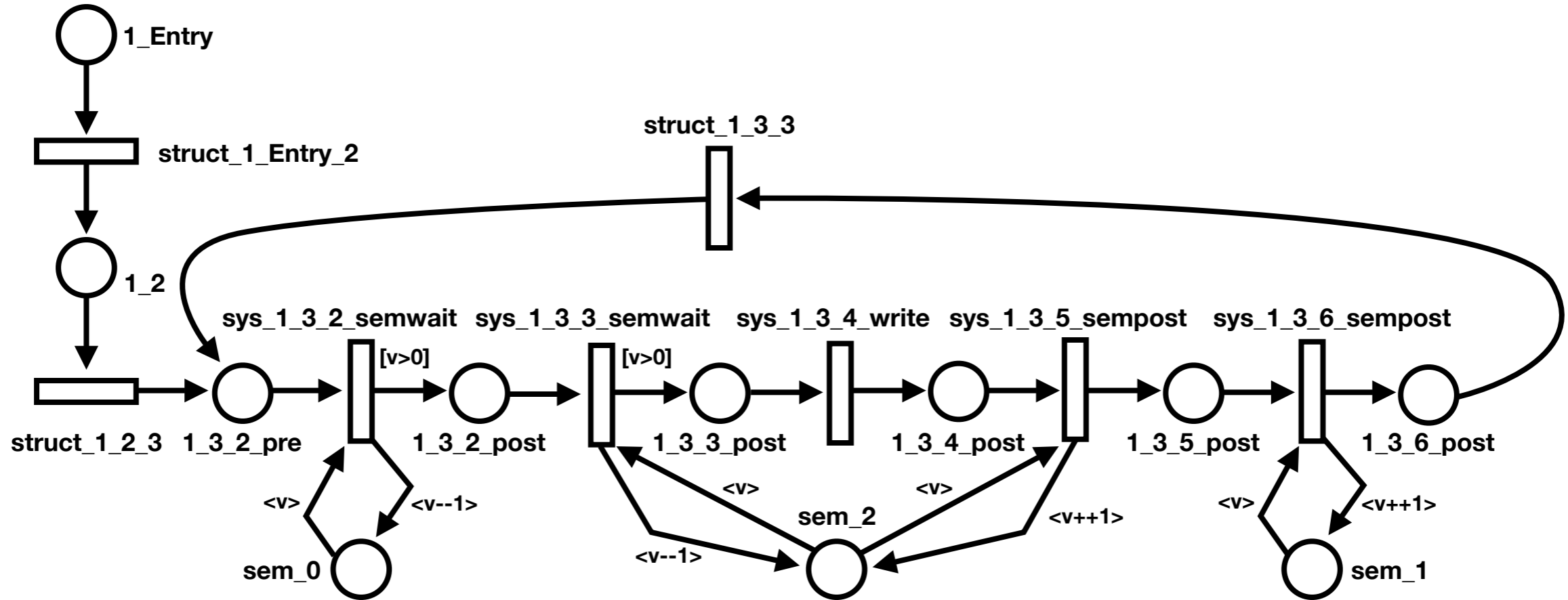


Builder : Merging selected perspectives



Builder : Merging selected perspectives

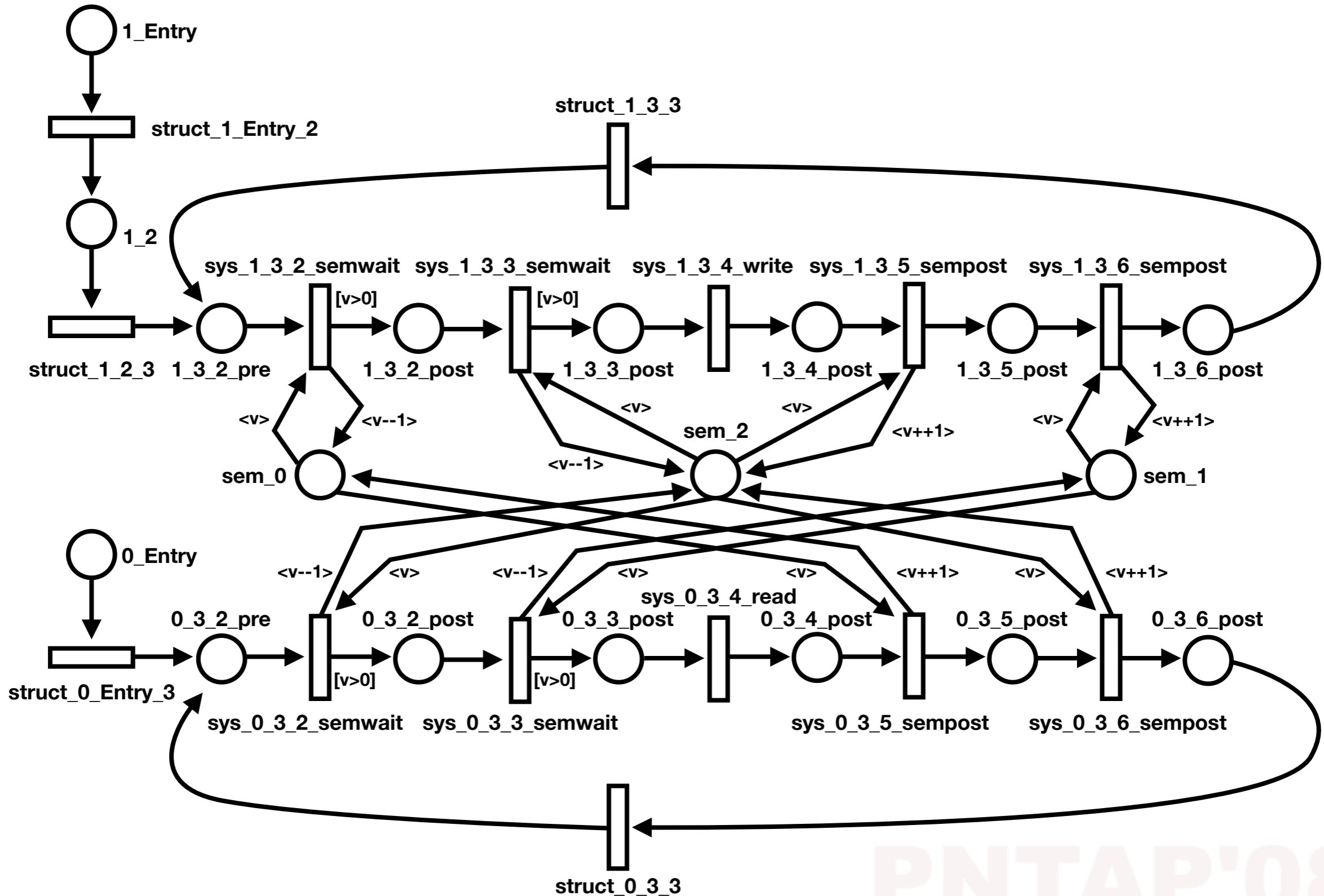
Writer (1)



Builder : Merging selected perspectives

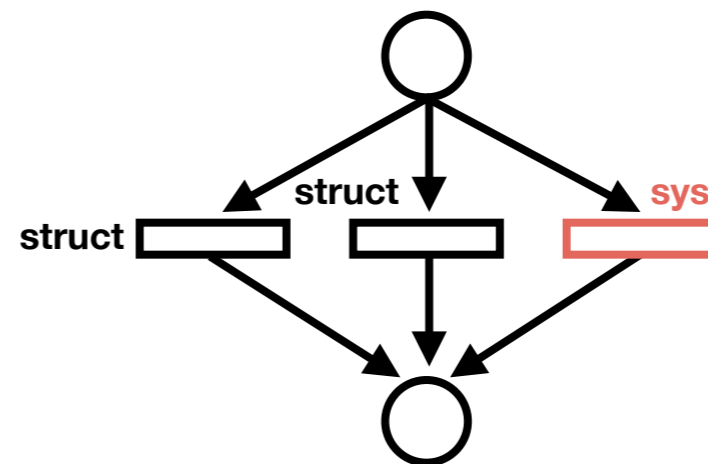
Writer (1)

Reader (0)



Optimization : Reducing Petri nets

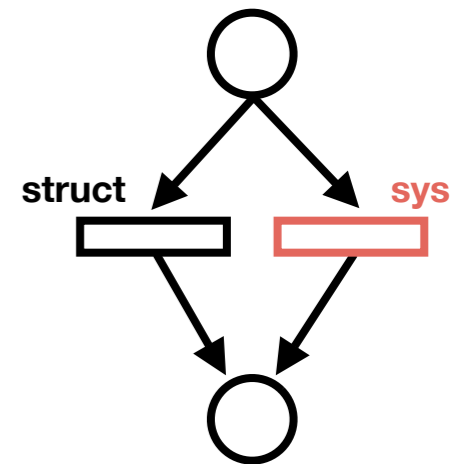
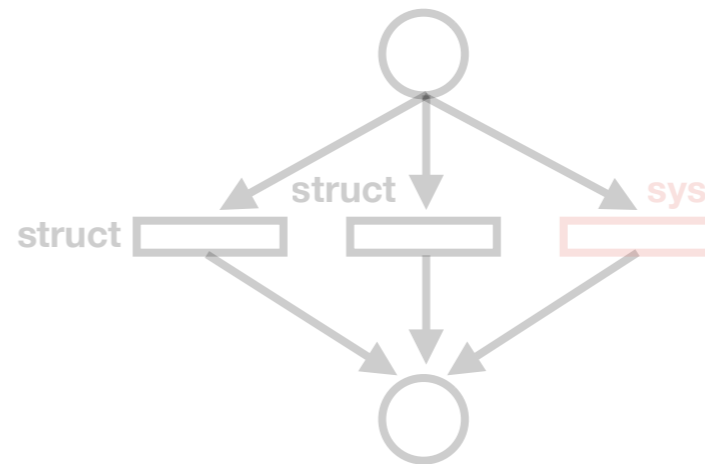
- Objective : Do not alterate the modeled behavior
 - Only consider **structural places or transitions**
- Use of **Haddad's reductions** (adapted to fit our strategy)
 - Pre-Agglomeration of transitions
 - Post-Agglomeration of transitions
 - *Diamond* reductions



- **New reductions** will be used as soon as necessary
 - New perspectives / New constructions
- On benchmarks : up to 65% gain

Optimization : Reducing Petri nets

- Objective : Do not alterate the modeled behavior
 - Only consider **structural places or transitions**
 - Use of **Haddad's reductions** (adapted to fit our strategy)
 - Pre-Agglomeration of transitions
 - Post-Agglomeration of transitions
 - *Diamond* reductions
-
- **New reductions** will be used as soon as necessary
 - New perspectives / New constructions
 - On benchmarks : up to 65% gain



Analysis : Structural Information

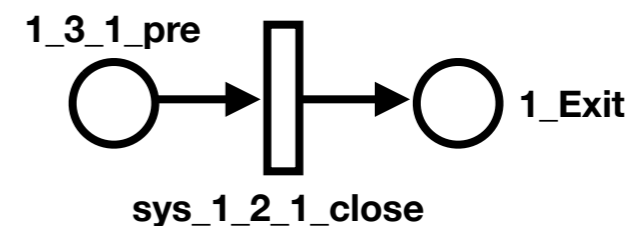
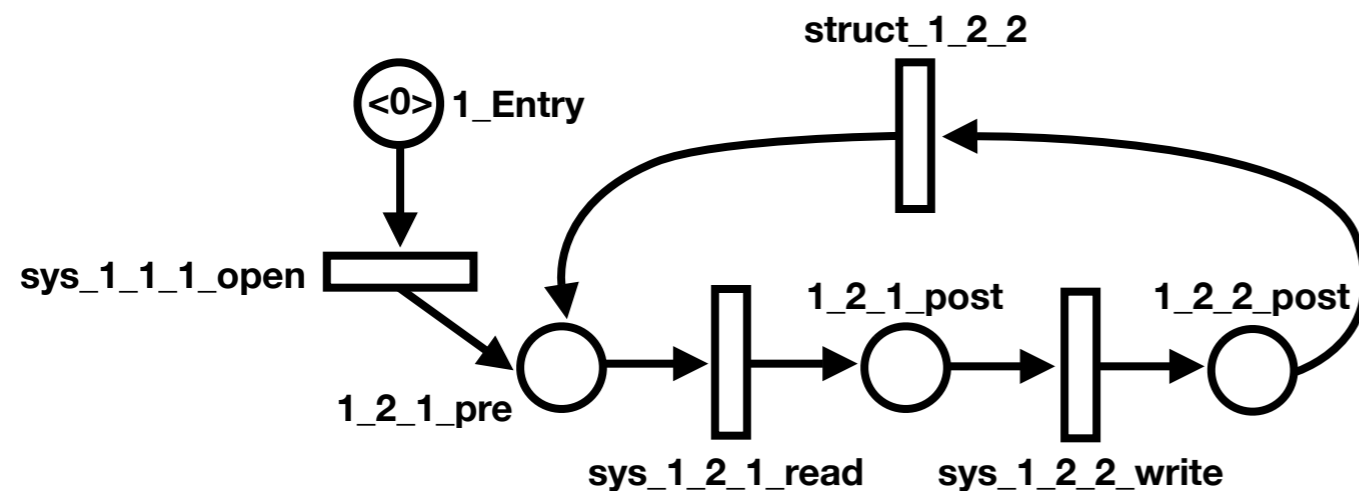
◎ Particular structures

- Structural **infinite loops**

- *Cycle without exit condition*

- Structural **dead code**

- *Subnet not connected to the main Petri net (without initial marking)*



◎ Other assertions coming from **programming good practices**

Analysis : Behavioral Information

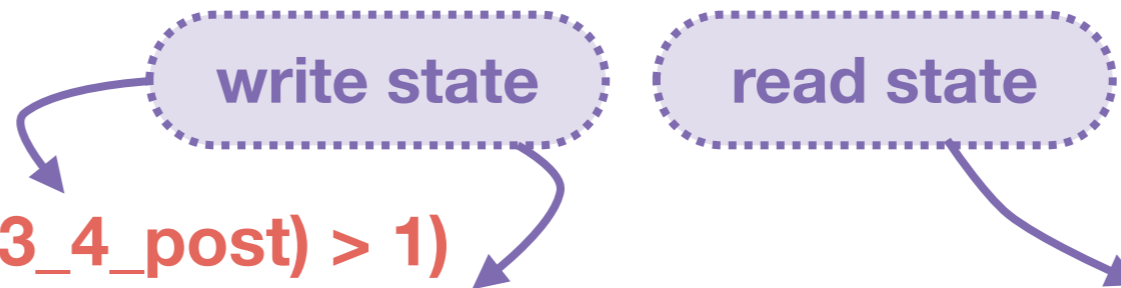
- Check some specific configurations (eg. security concerns)
- Exclusive Write
 - Reachability : $(\text{card}(1_3_4_post) > 1)$
 $\parallel ((\text{card}(1_3_4_post) > 0) \ \&\& \ (\text{card}(0_3_4_post) > 0))$
- Write First
 - CTL : $\text{AG} ((\text{card}(0_3_4_post) = 0) \ \text{U} \ (\text{card}(1_3_4_post) > 0))$
 - Failed !
- Existing deadlocks ?
 - **Yes.** When a reader first accesses to the critical ressource.
- Use of CPN-AMI to analyse the model

Analysis : Behavioral Information

- Check some specific configurations (eg. security concerns)

- Exclusive Write

- Reachability : $(\text{card}(1_3_4_post) > 1) \parallel ((\text{card}(1_3_4_post) > 0) \ \&\& \ (\text{card}(0_3_4_post) > 0))$



- Write First

- CTL : $\text{AG} ((\text{card}(0_3_4_post) = 0) \ \text{U} \ (\text{card}(1_3_4_post) > 0))$
- Failed !

- Existing deadlocks ?

- **Yes.** When a reader first accesses to the critical ressource.

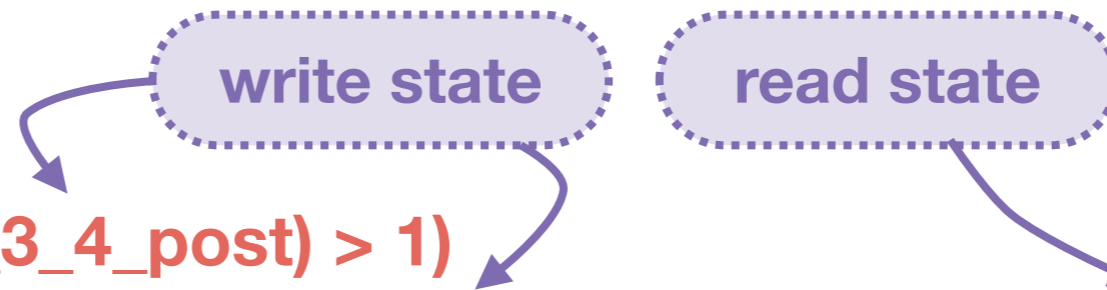
- Use of CPN-AMI to analyse the model

Analysis : Behavioral Information

- Check some specific configurations (eg. security concerns)

- Exclusive Write

- Reachability : $(\text{card}(1_3_4_post) > 1) \parallel ((\text{card}(1_3_4_post) > 0) \ \&\& \ (\text{card}(0_3_4_post) > 0))$



- Write First

- CTL : $\text{AG} ((\text{card}(0_3_4_post) = 0) \ \text{U} \ (\text{card}(1_3_4_post) > 0))$

- Failed !



- Existing deadlocks ?

- **Yes.** When a reader first accesses to the critical ressource.

- Use of CPN-AMI to analyse the model

Conclusion & Perspectives

- **Automatic** way to build Petri nets from source code

- Use of **GCC's internal representations**
- Focus on specific program's aspects describes as **perspectives**

- Our prototype scales up

	whois	ping	gzip
program's size (loc)	874	2454	7323
model size (nodes)	1499	2348	5692
optimized model size (nodes)	627	1037	3301

- Future work

- Development of **new perspectives** (signals, array bounds...)
- Add **data flow management** to our parser
- Migrate information extraction to be **independent from the input language**

Thank you for your attention

© Questions ?