MeFoSyLoMa Meeting

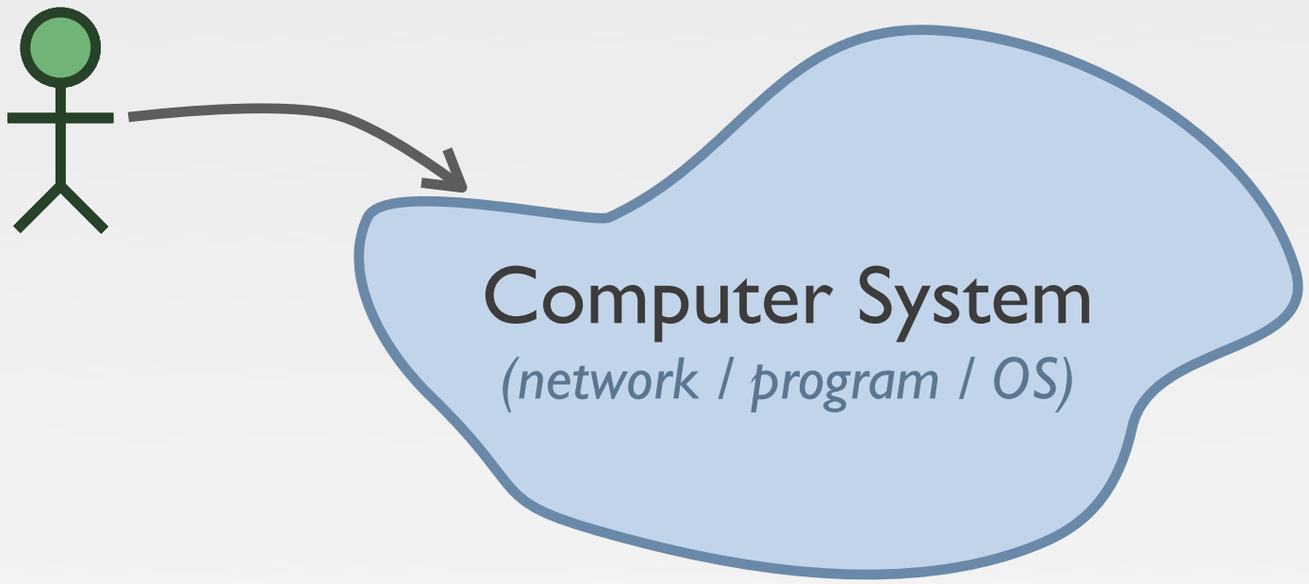
Jean-Baptiste Voron - *LIP6 (UPMC)*
Fabrice Kordon - *LIP6 (UPMC)*
Liviu Iftode - *DiscoLab (Rutgers University)*

"Evinrude"

How to build a **Petri Net-Based IDS** from **Program Sources**

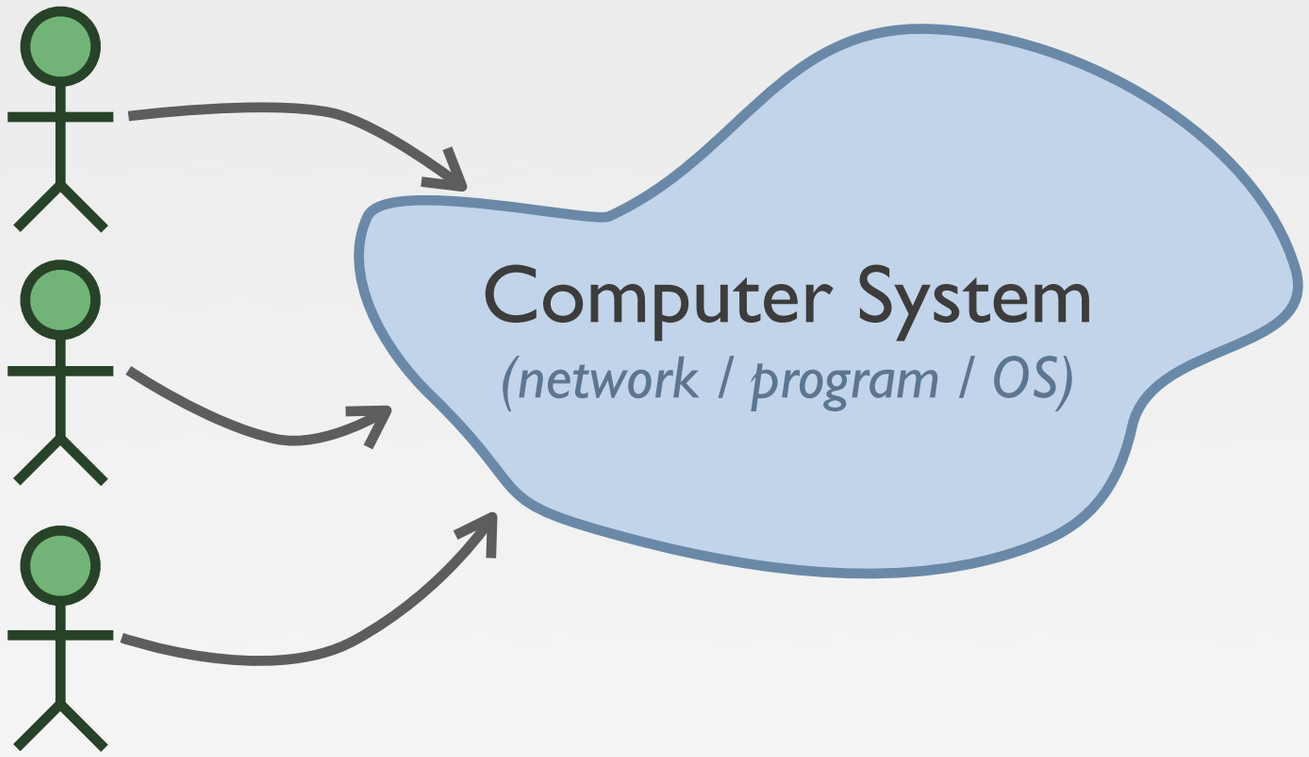
December 5th, 2008 - Version 1.3b

Context: Intrusion Detection



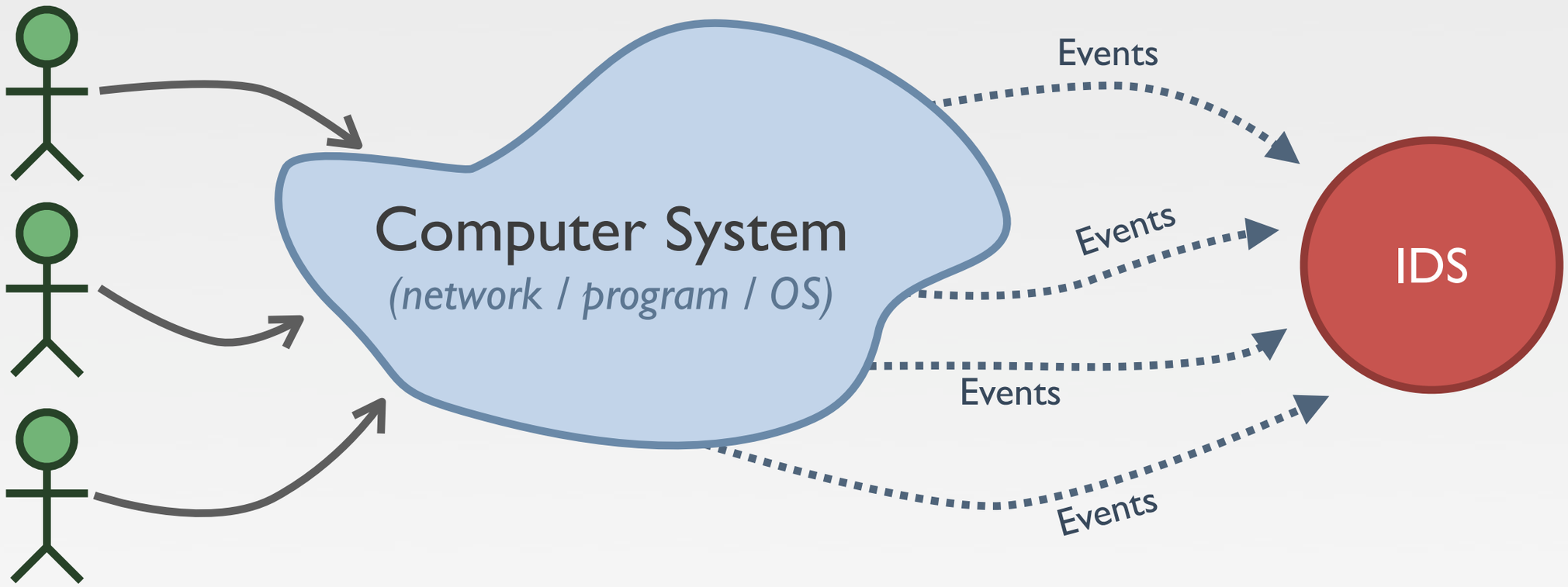
- Signature-based detection (misuse detection)
- Behavior-based detection (anomaly detection)
  - Looking for deviations from an "expected behavior"
  - Limitation: parallel and distributed programs

Context: Intrusion Detection



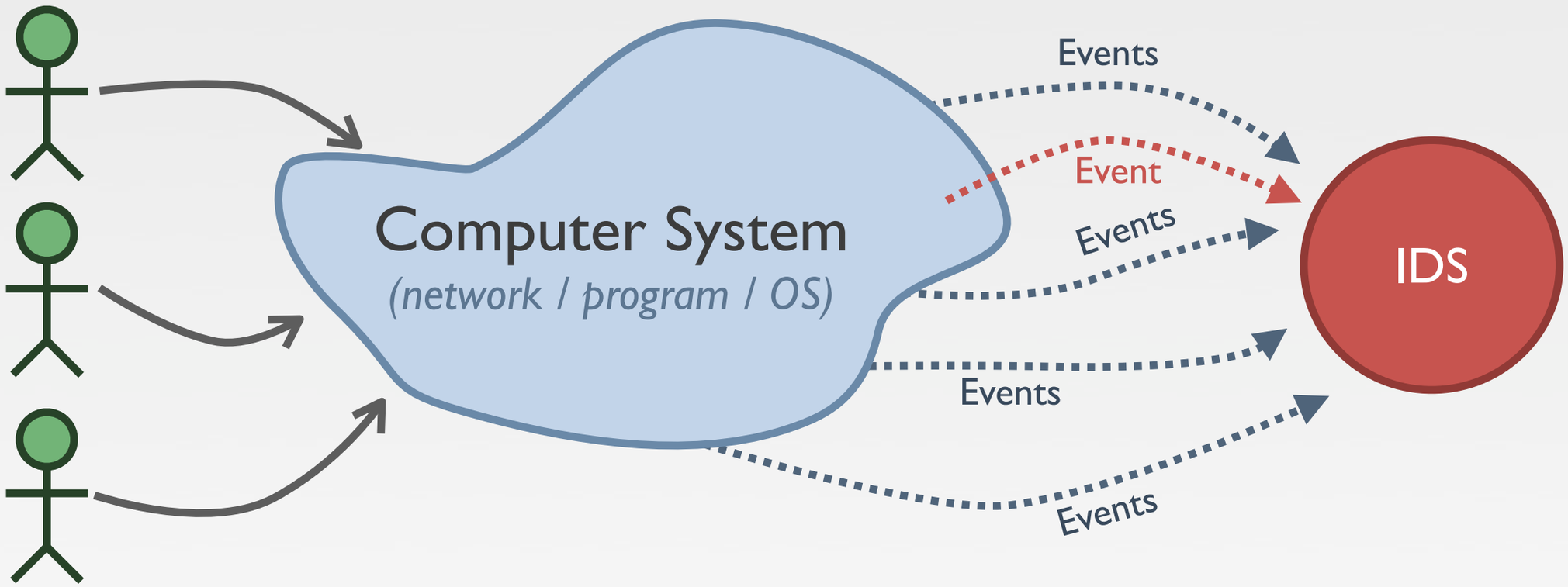
- Signature-based detection (misuse detection)
- Behavior-based detection (anomaly detection)
  - Looking for deviations from an "expected behavior"
  - Limitation: parallel and distributed programs

Context: Intrusion Detection



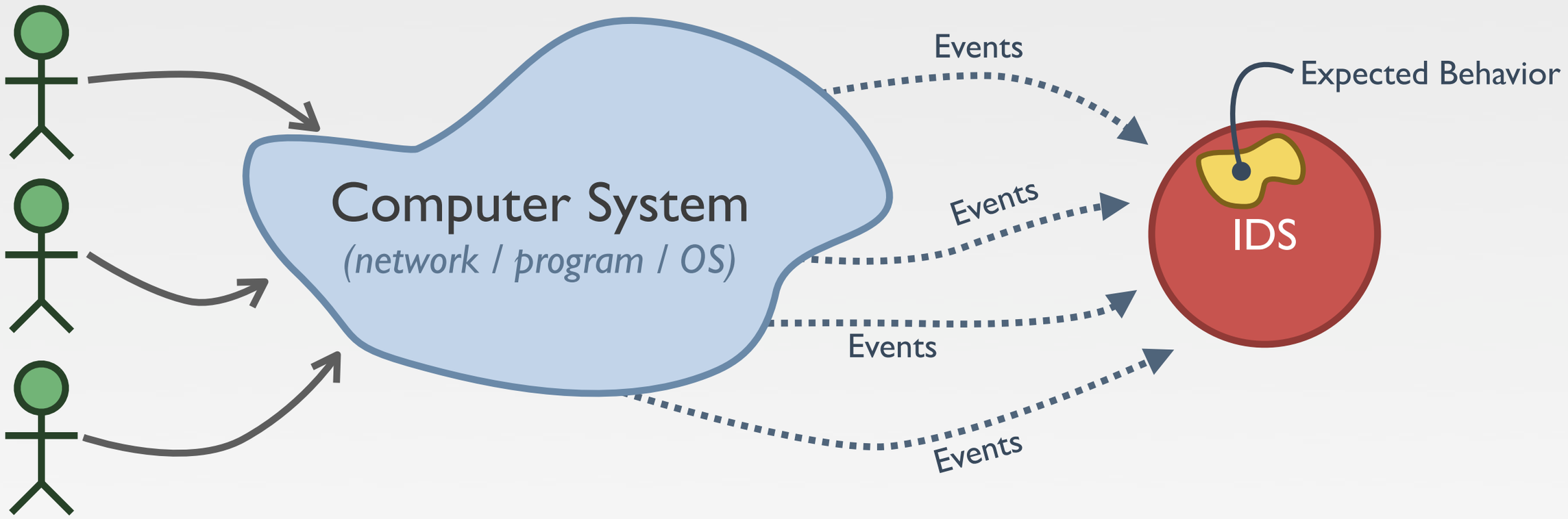
- Signature-based detection (misuse detection)
- Behavior-based detection (anomaly detection)
  - Looking for deviations from an "expected behavior"
  - Limitation: parallel and distributed programs

Context: Intrusion Detection



- Signature-based detection (misuse detection)
- Behavior-based detection (anomaly detection)
  - Looking for deviations from an "expected behavior"
  - Limitation: parallel and distributed programs

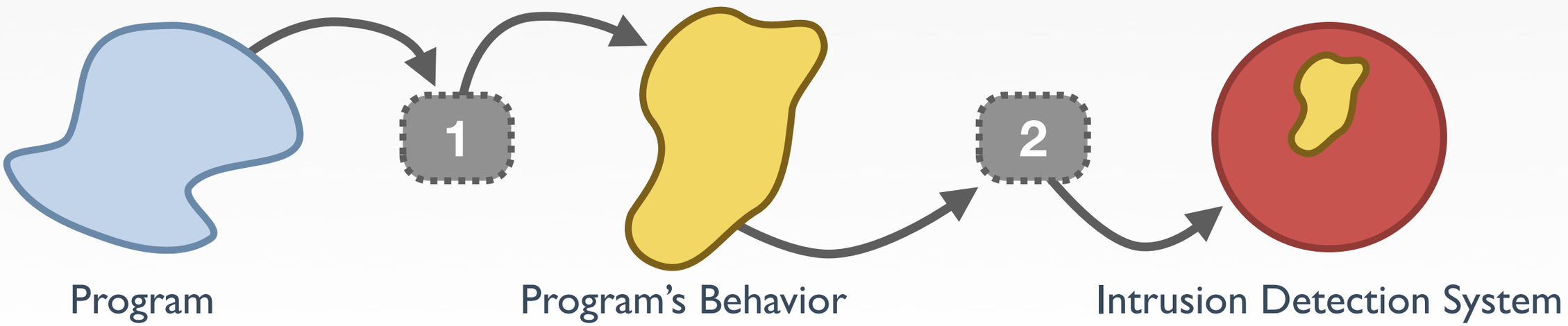
Context: Intrusion Detection



- Signature-based detection (misuse detection)
- Behavior-based detection (anomaly detection)
  - Looking for deviations from an "expected behavior"
  - Limitation: parallel and distributed programs

Approach & Choices

- Build an Intrusion Detection System (IDS)
  - Dedicated to a program & Behavior based
- Handle large and complex programs
  - C programs (real-life programs)
  - Multi Processes / Multi Threaded programs

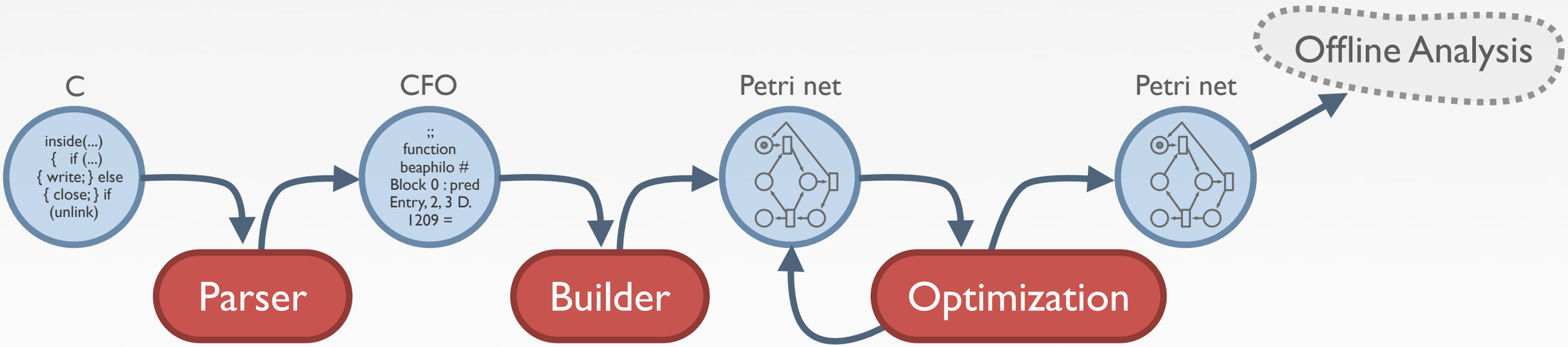


Approach & Choices

- Behavior modeling relies on Petri nets
  - Used as state-space generators
- Construction process is fully automatic
  - IDS is produced from program sources
  - No formal background required for developers
  - No code instrumentation (neither source nor binary)
- Hypothesis
  - Operating System is considered healthy

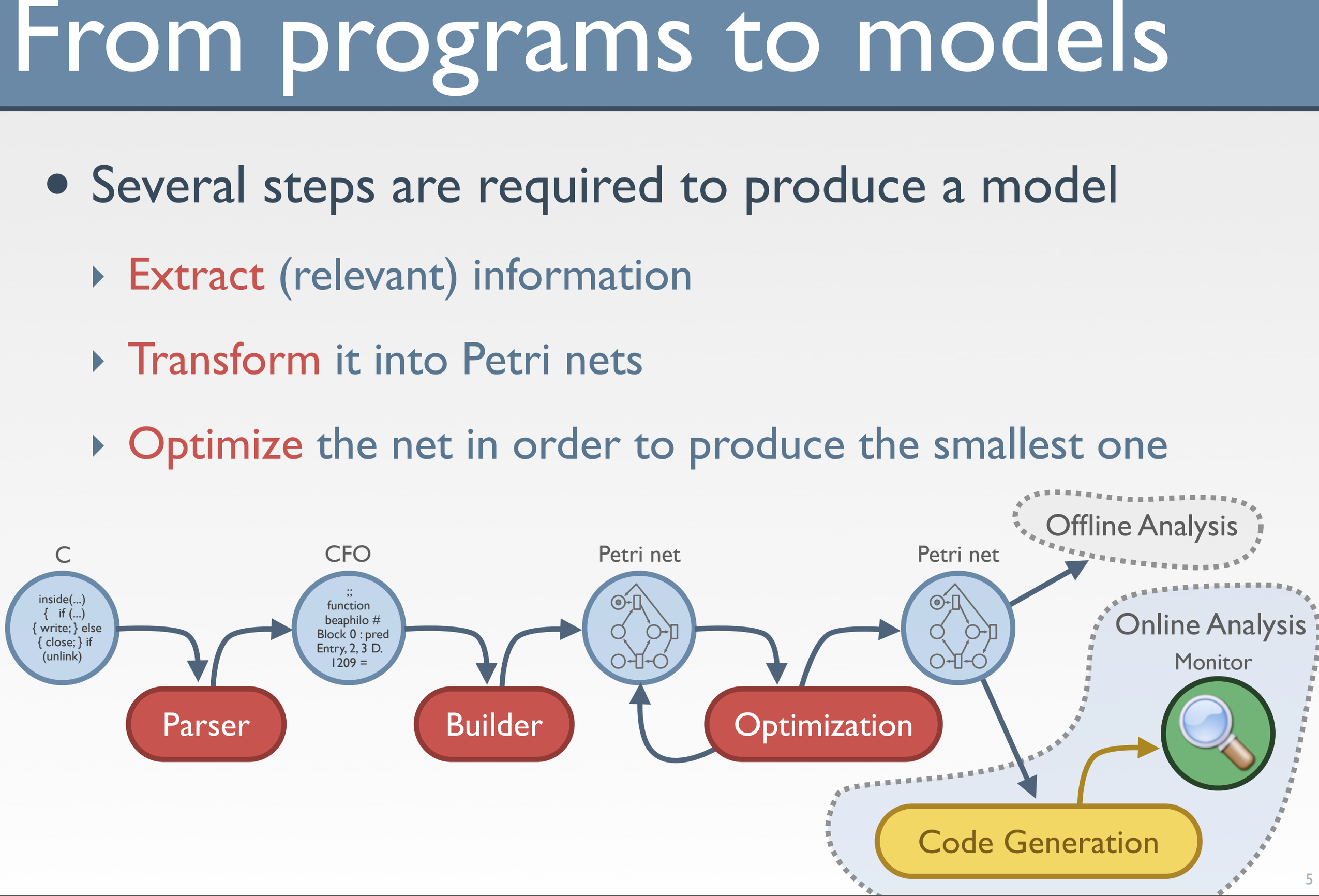
From programs to models

- Several steps are required to produce a model
  - Extract (relevant) information
  - Transform it into Petri nets
  - Optimize the net in order to produce the smallest one



From programs to models

- Several steps are required to produce a model
  - Extract (relevant) information
  - Transform it into Petri nets
  - Optimize the net in order to produce the smallest one



Monitoring Philosophers...

- ... is pretty hard with standard IDS !
- Multi-threaded version
  - Pthread library calls
    - Thread Management
    - Thread Synchronization

Monitoring Philosophers...

- ... is pretty hard with standard IDEs !

```
int main (int argn, char **argv) {
  pthread_mutex_init(&foodlock,NULL);

  // Forks...
  for (i = 0; i < PHILO; i++) {
    pthread_mutex_init(&fork[i],NULL);
    pthread_create(&p[i],NULL,philosopher,NULL);
  }

  for (i = 0; i < PHILO; i++) {
    pthread_join(p[i],NULL);
  }
```

Monitoring Philosophers...

- ... is pretty hard
-

```
int main (int argn, char **
 pthread_mutex_init(&food

 // Forks...
 for (i = 0; i < PHILO
  pthread_mutex_init
  pthread_create(&p
 }

 for (i = 0; i < PHILO
  pthread_join(p[i],NULL)
 }
```

```
void * philosopher () {
 int f;
 printf("Philo is sitting down to dinner.\n");
 while((f = food_on_table())) {
  pthread_mutex_lock(&fork_left);
  pthread_mutex_lock(&fork_right);
  printf("Philo is eating.\n");
  pthread_mutex_unlock(&fork_right);
  pthread_mutex_unlock(&fork_left);
 }
 printf("Philo is done eating.\n");
 pthread_exit(NULL);
}
```

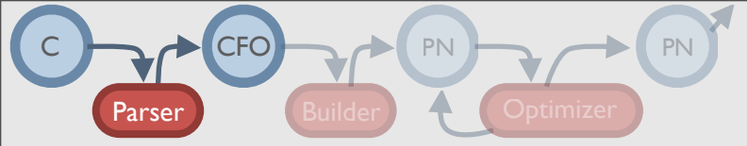
Monitoring Philosophers...

- ... is pretty hard
-

```
int main (int argn, char **
  pthread_mutex_init(&food

  // Forks...
  for (i = 0; i < PHILO
    pthread_mutex_init
    pthr  create(&p
  }

  for
    pt
  }
```

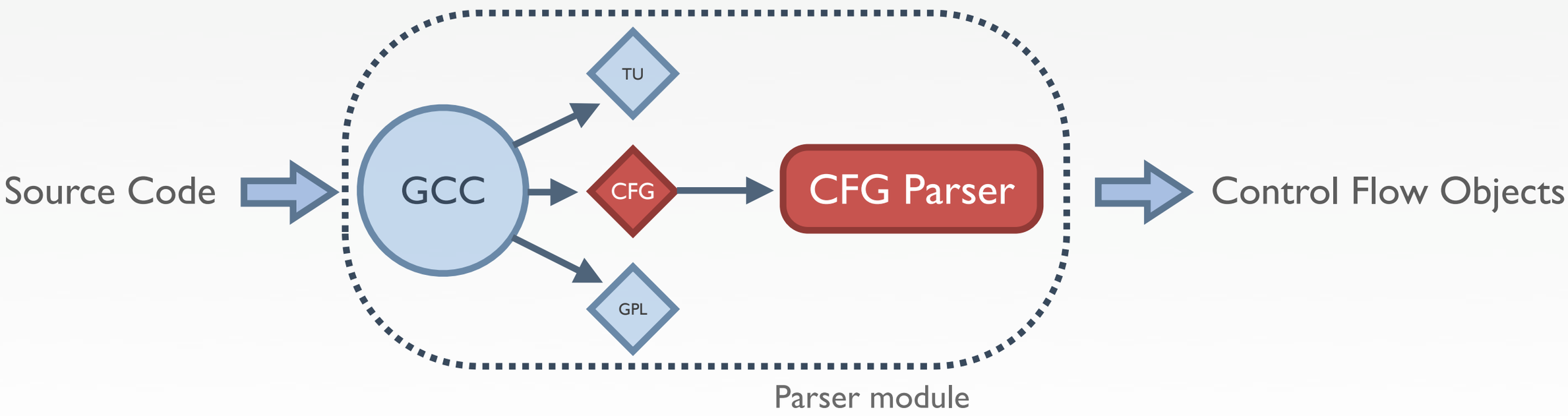
```
void * philosopher () {
  int f;
  printf("Philo is sitting down to dinner.\n");
  while((f = food_on_table())) {
    pthread_mutex_lock(&fork_left);
    pthread_mutex_lock(&fork_right);
    printf("Philo is eating.\n");
    pthread_mutex_unlock(&fork_right);
    pthread_mutex_unlock(&fork_left);
  }
  printf("Philo is
}
```

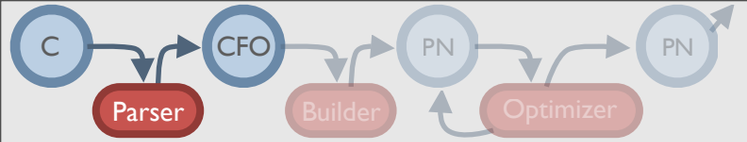
```
int food_on_table () {
  static int food = FOOD;
  int myfood;

  pthread_mutex_lock(&foodlock);
  if (food > 0) { food--; }
  myfood = food;
  pthread_mutex_unlock (&foodlock);
  return myfood;
}
```



Extracting Information

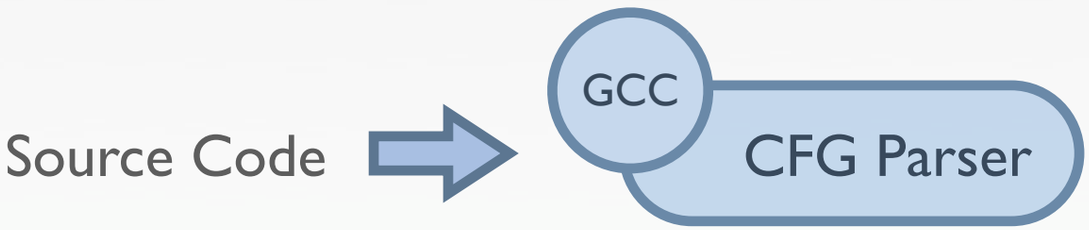
- Use of GCC to extract information
  - During the compilation
  - No need to modify the MakeFile (just set ENV variable)
  - Use of Extended Control Flow Graph (ECFG)

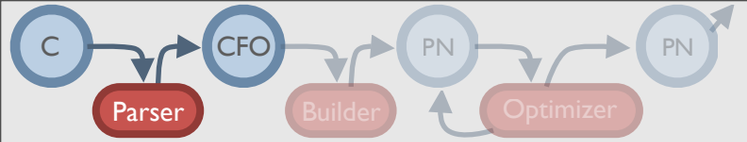




Dealing with Perspectives

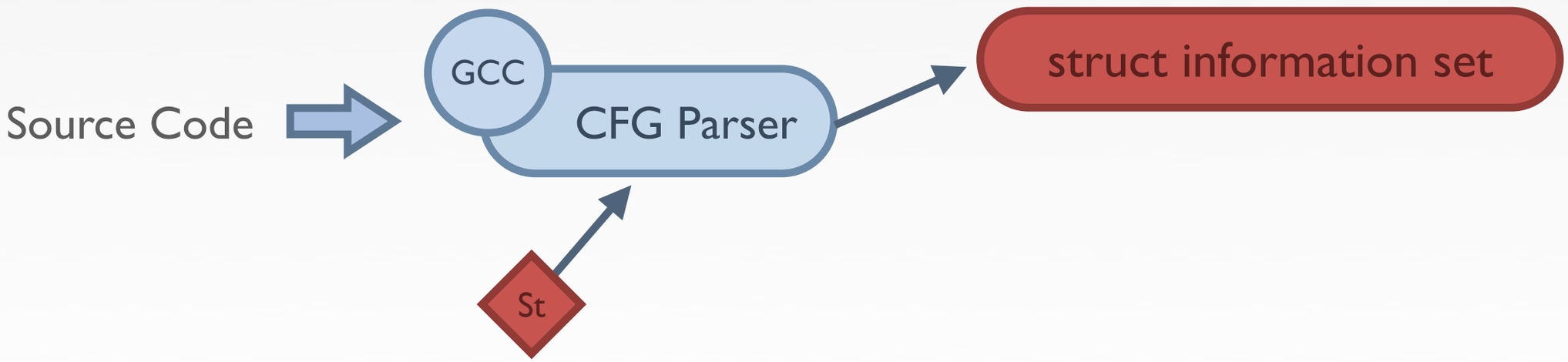
- All extracted information is not relevant for analysis
  - However, structural information is systematically extracted
- Need a flexible way to analyse source: Perspectives
  - Based on a dictionary of remarkable elements
  - Set of transformations

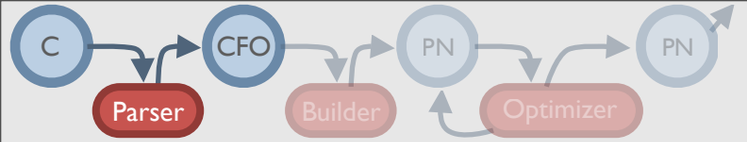




Dealing with Perspectives

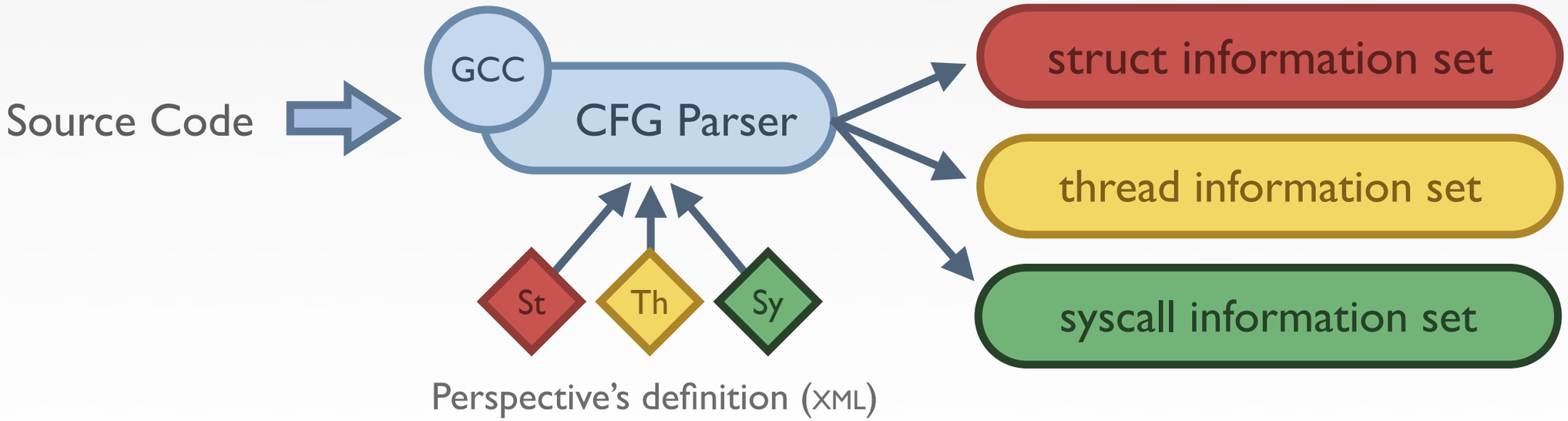
- All extracted information is not relevant for analysis
  - However, structural information is systematically extracted
- Need a flexible way to analyse source: Perspectives
  - Based on a dictionary of remarkable elements
  - Set of transformations

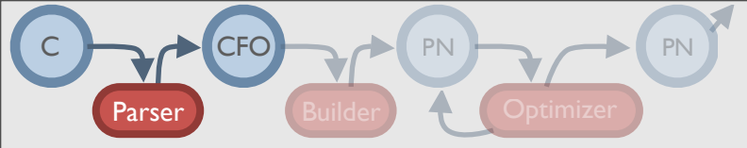




Dealing with Perspectives

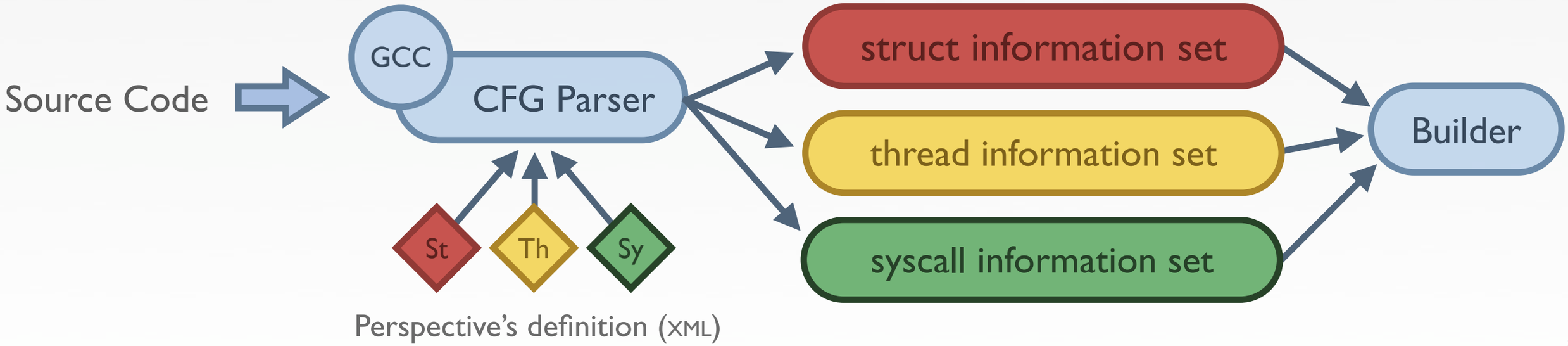
- All extracted information is not relevant for analysis
  - However, structural information is systematically extracted
- Need a flexible way to analyse source: Perspectives
  - Based on a dictionary of remarkable elements
  - Set of transformations

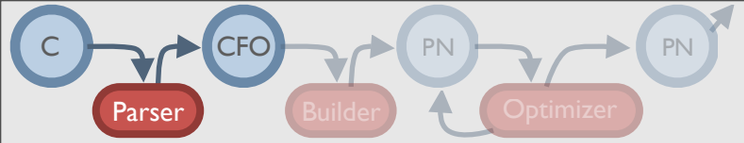




Dealing with Perspectives

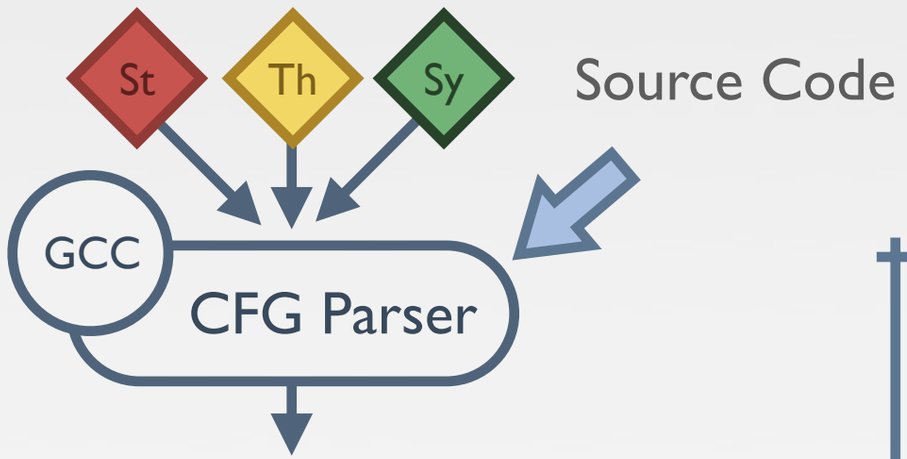
- All extracted information is not relevant for analysis
  - However, structural information is systematically extracted
- Need a flexible way to analyse source: Perspectives
  - Based on a dictionary of remarkable elements
  - Set of transformations



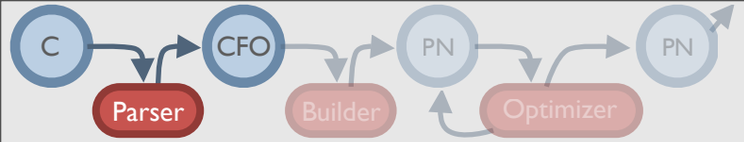


Perspectives & Philosophers

Perspective's descriptions (XML)

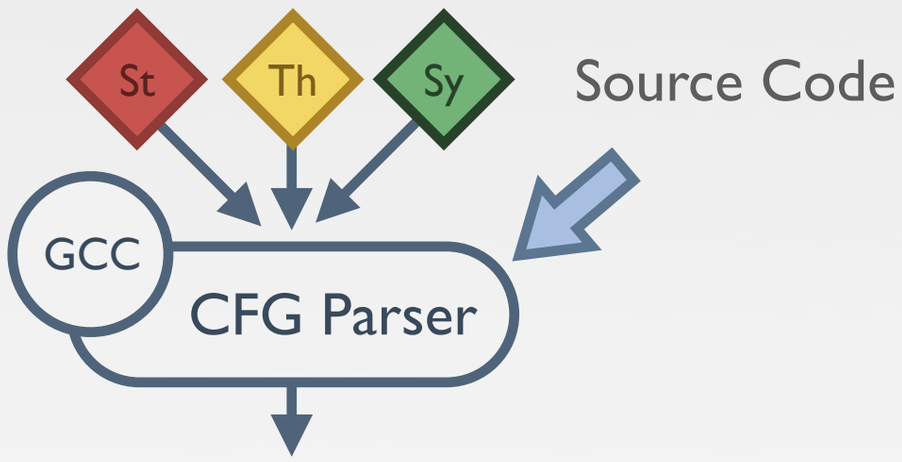



```
void * philosopher () {
 int f;
 printf("Philo is sitting down to dinner.\n");
 while((f = food_on_table())) {
  pthread_mutex_lock(&fork_left);
  pthread_mutex_lock(&fork_right);
  printf("Philo is eating.\n");
  pthread_mutex_unlock(&fork_right);
  pthread_mutex_unlock(&fork_left);
 }
 printf("Philo is done eating.\n");
 pthread_exit(NULL);
}
```



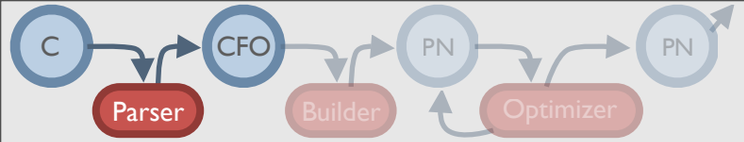
Perspectives & Philosophers

Perspective's descriptions (XML)



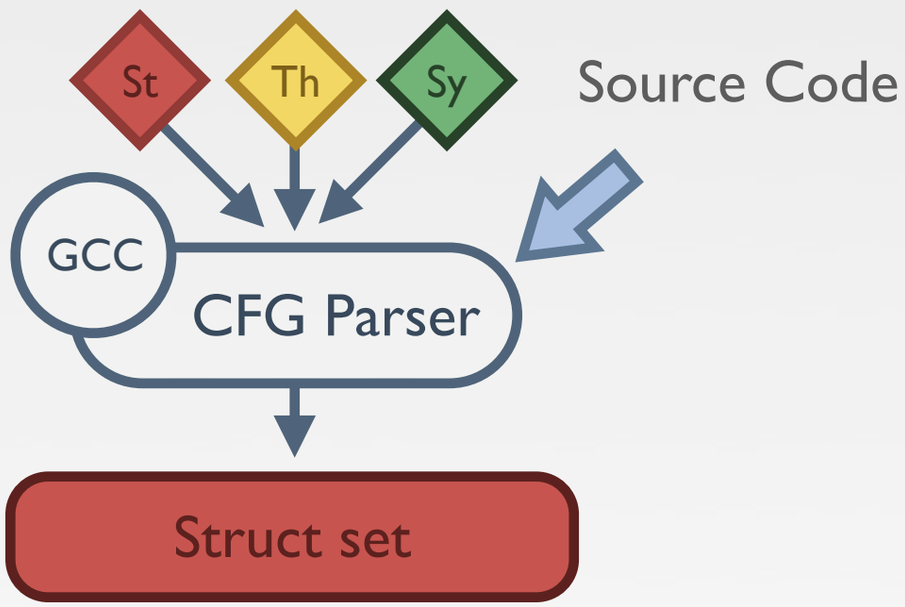
ECFG

```
1.   ;; Function philosopher
2.   # BLOCK 2
3.     # PRED:ENTRY(fallthru)
4.      printf(&"Philosopher ...
5.      goto <bb 4> (<L1>);
6.     # SUCC:4(fallthru)
7.   # BLOCK 3
8.     # PRED: 4 (true)
9.      pthread_mutex_lock(&fork3);
10.     pthread_mutex_lock(&fork1);
11.     printf(&"Philosopher ...
12.     pthread_mutex_unlock(&fork3);
13.     pthread_mutex_unlock(&fork1);
14.    # SUCC:4(fallthru)
15.  # BLOCK 4
16.    # PRED:2(fallthru) 3(fallthru)
17.     D.3892 = food_on_table();
18.     f = D.3892;
19.     if (f != 0) goto <L0>;
        else goto <L2>;
20.    # SUCC:3(true) 5(false)
21.  # BLOCK 5
22.    # PRED:4(false)
23.     printf(&"Philosopher ...
24.     pthread_exit (0B);
25.    # SUCC:EXIT
```

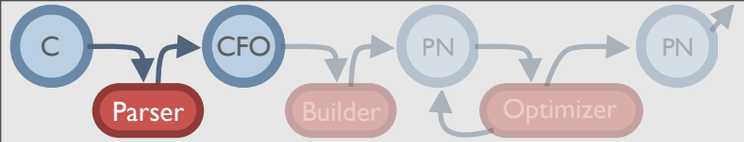


Perspectives & Philosophers

Perspective's descriptions (XML)

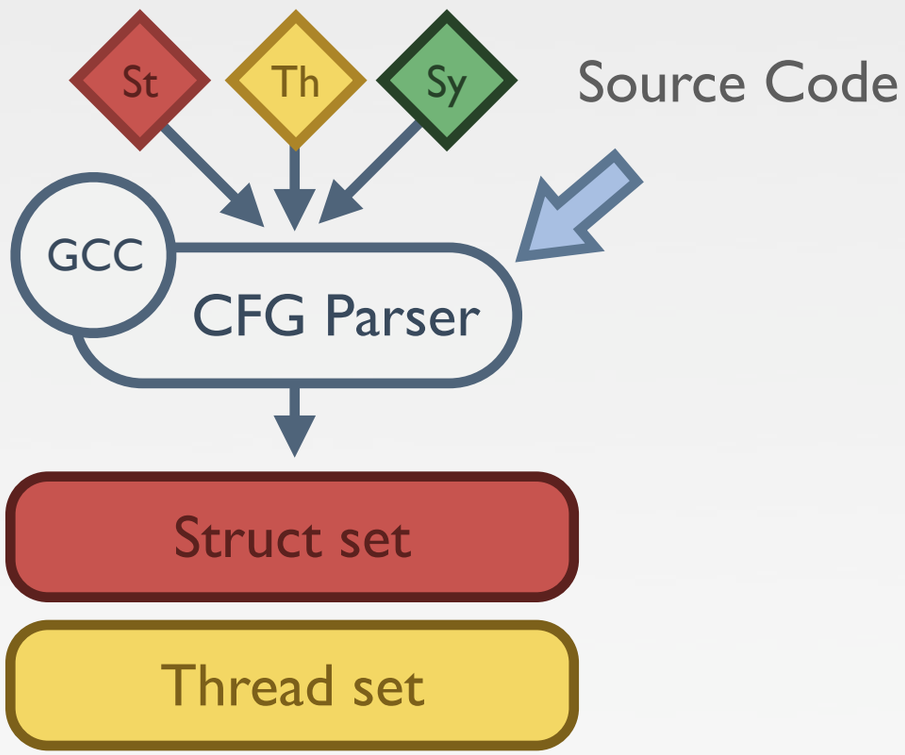



```
1.   ;; Function philosopher
2.   # BLOCK 2
3.     # PRED:ENTRY(fallthru)
4.      printf(&"Philosopher ...
5.      goto <bb 4> (<L1>);
6.     # SUCC:4(fallthru)
7.   # BLOCK 3
8.     # PRED: 4 (true)
9.      pthread_mutex_lock(&fork3);
10.     pthread_mutex_lock(&fork1);
11.     printf(&"Philosopher ...
12.     pthread_mutex_unlock(&fork3);
13.     pthread_mutex_unlock(&fork1);
14.    # SUCC:4(fallthru)
15.  # BLOCK 4
16.    # PRED:2(fallthru) 3(fallthru)
17.     D.3892 = food_on_table();
18.     f = D.3892;
19.     if (f != 0) goto <L0>;
        else goto <L2>;
20.    # SUCC:3(true) 5(false)
21.  # BLOCK 5
22.    # PRED:4(false)
23.     printf(&"Philosopher ...
24.     pthread_exit (0B);
25.    # SUCC:EXIT
```

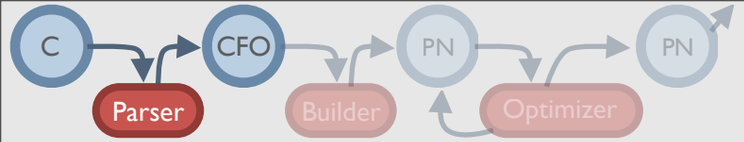


Perspectives & Philosophers

Perspective's descriptions (XML)

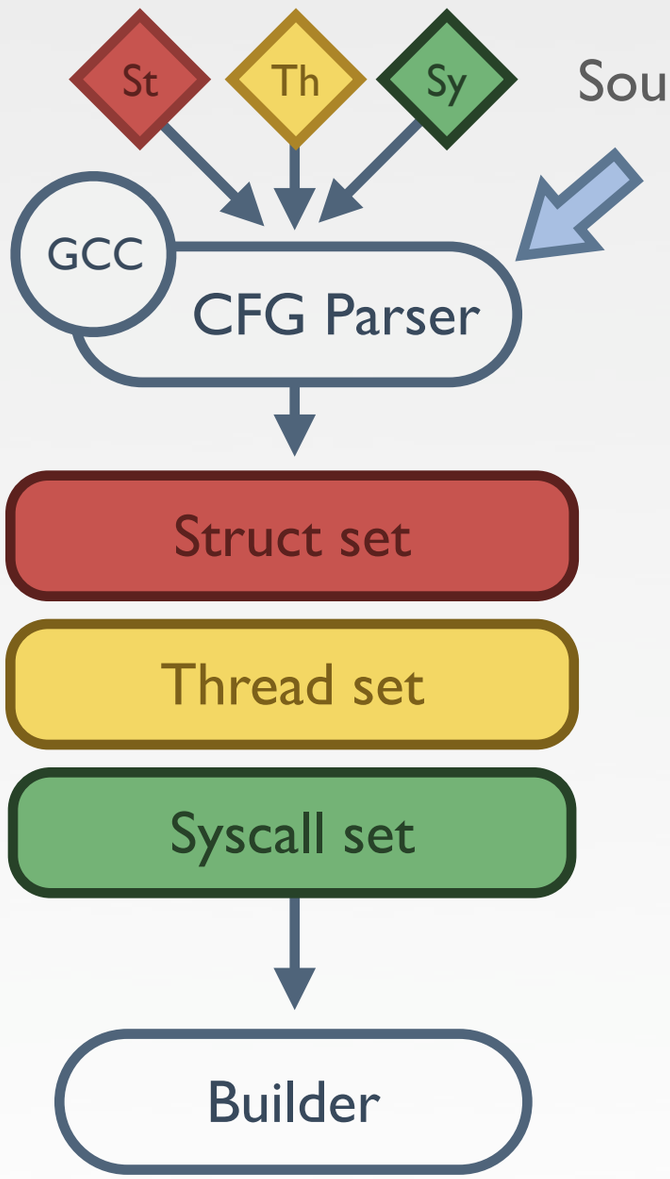



```
1.   ;; Function philosopher
2.   # BLOCK 2
3.     # PRED:ENTRY(fallthru)
4.      printf(&"Philosopher ...
5.      goto <bb 4> (<L1>);
6.     # SUCC:4(fallthru)
7.   # BLOCK 3
8.     # PRED: 4 (true)
9.      pthread_mutex_lock(&fork3);
10.     pthread_mutex_lock(&fork1);
11.     printf(&"Philosopher ...
12.     pthread_mutex_unlock(&fork3);
13.     pthread_mutex_unlock(&fork1);
14.    # SUCC:4(fallthru)
15.  # BLOCK 4
16.    # PRED:2(fallthru) 3(fallthru)
17.     D.3892 = food_on_table();
18.     f = D.3892;
19.     if (f != 0) goto <L0>;
        else goto <L2>;
20.    # SUCC:3(true) 5(false)
21.  # BLOCK 5
22.    # PRED:4(false)
23.     printf(&"Philosopher ...
24.     pthread_exit (0B);
25.    # SUCC:EXIT
```

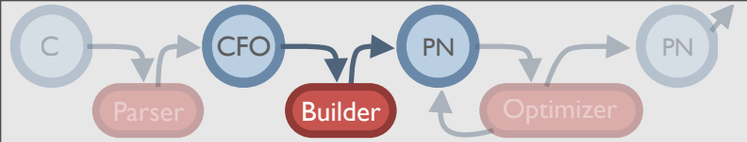


Perspectives & Philosophers

Perspective's descriptions (XML)

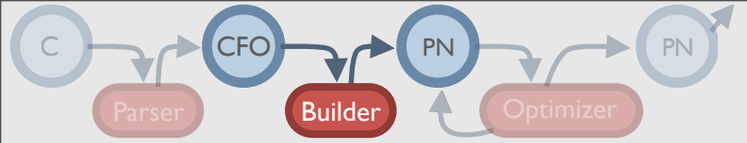



```
1.   ;; Function philosopher
2.   # BLOCK 2
3.     # PRED:ENTRY(fallthru)
4.      printf(&"Philosopher ...
5.      goto <bb 4> (<L1>);
6.     # SUCC:4(fallthru)
7.   # BLOCK 3
8.     # PRED: 4 (true)
9.      pthread_mutex_lock(&fork3);
10.     pthread_mutex_lock(&fork1);
11.     printf(&"Philosopher ...
12.     pthread_mutex_unlock(&fork3);
13.     pthread_mutex_unlock(&fork1);
14.    # SUCC:4(fallthru)
15.  # BLOCK 4
16.    # PRED:2(fallthru) 3(fallthru)
17.     D.3892 = food_on_table();
18.     f = D.3892;
19.     if (f != 0) goto <L0>;
        else goto <L2>;
20.    # SUCC:3(true) 5(false)
21.  # BLOCK 5
22.    # PRED:4(false)
23.     printf(&"Philosopher...
24.     pthread_exit (0B);
25.    # SUCC:EXIT
```

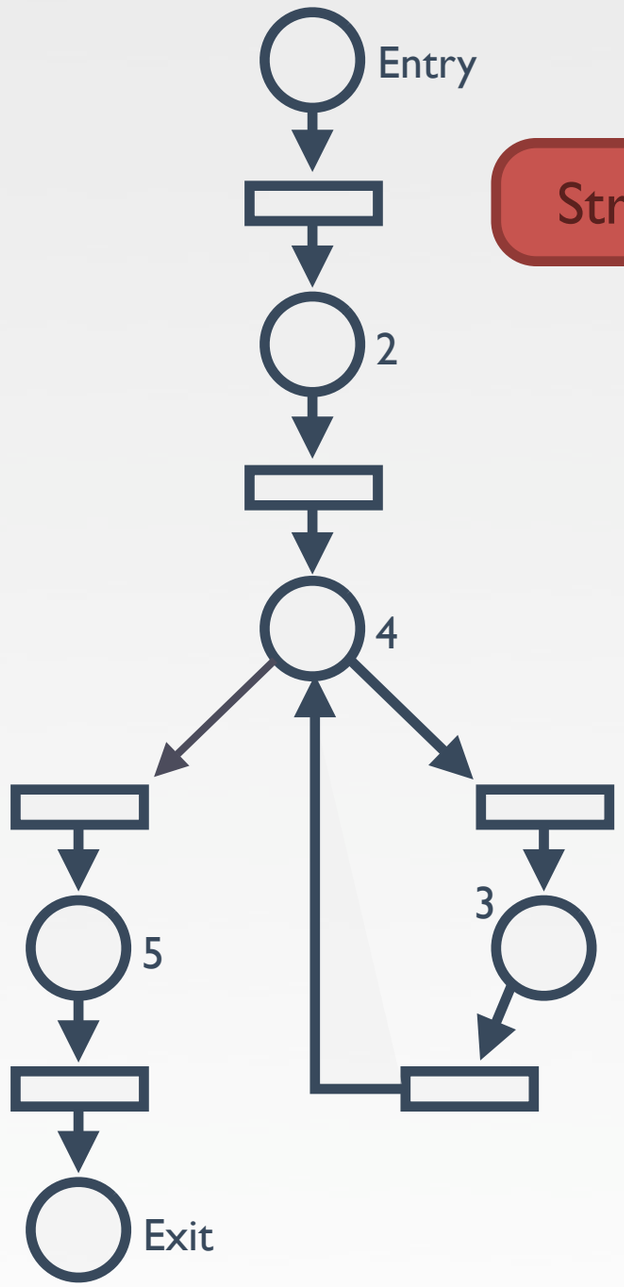


Building (small) Petri Nets

- Production rules : Source to Petri net patterns
  - Each perspective comes with its own production rules
    - 7 rules to build the structural model
    - 6 for Thread perspective / 6 for Process Management perspective
- Each information set is transformed into a Petri net
  - Struct information set gives Structural Model
  - Others give submodels to be plugged to the structural one

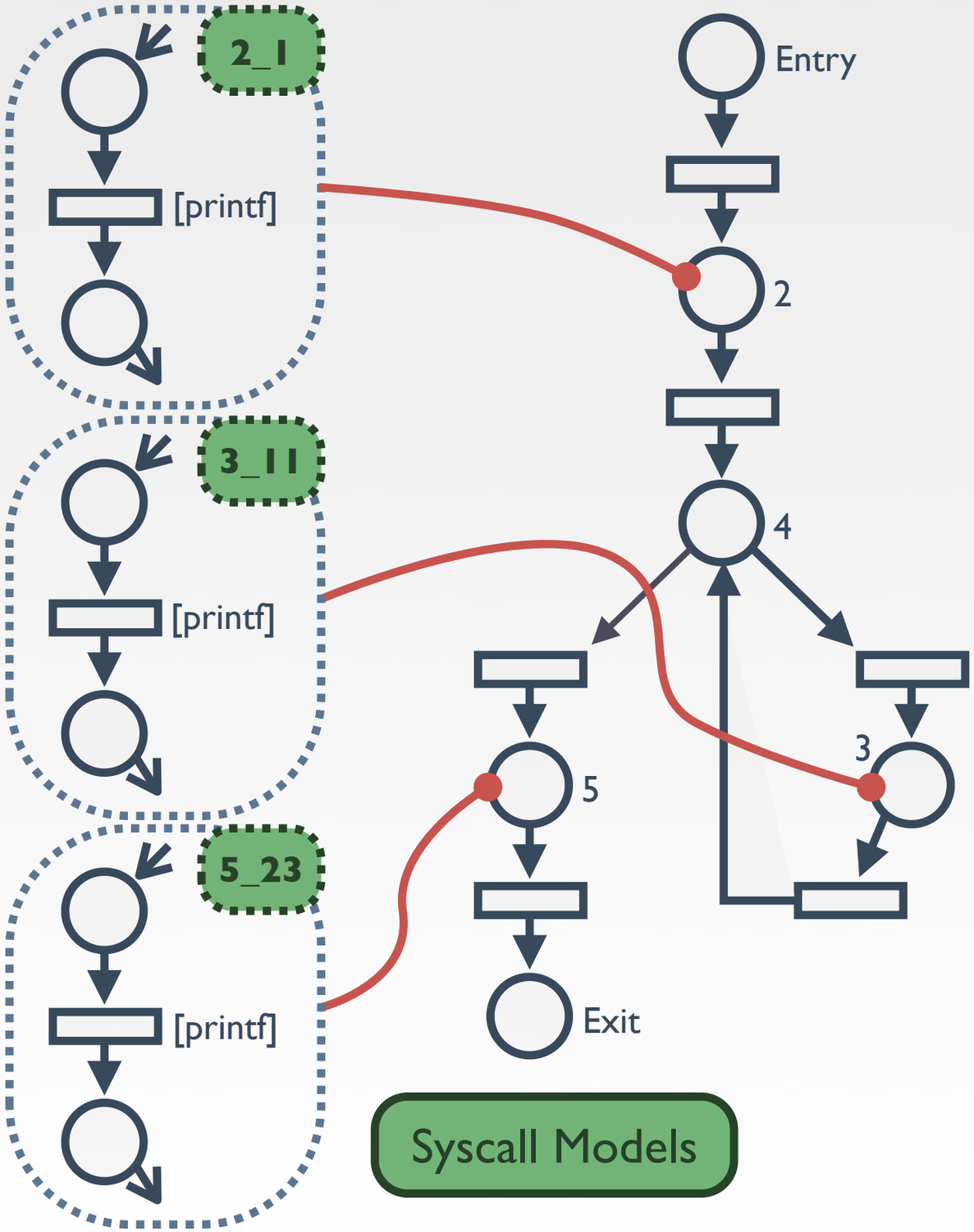


Philos'R'Nets (still small)

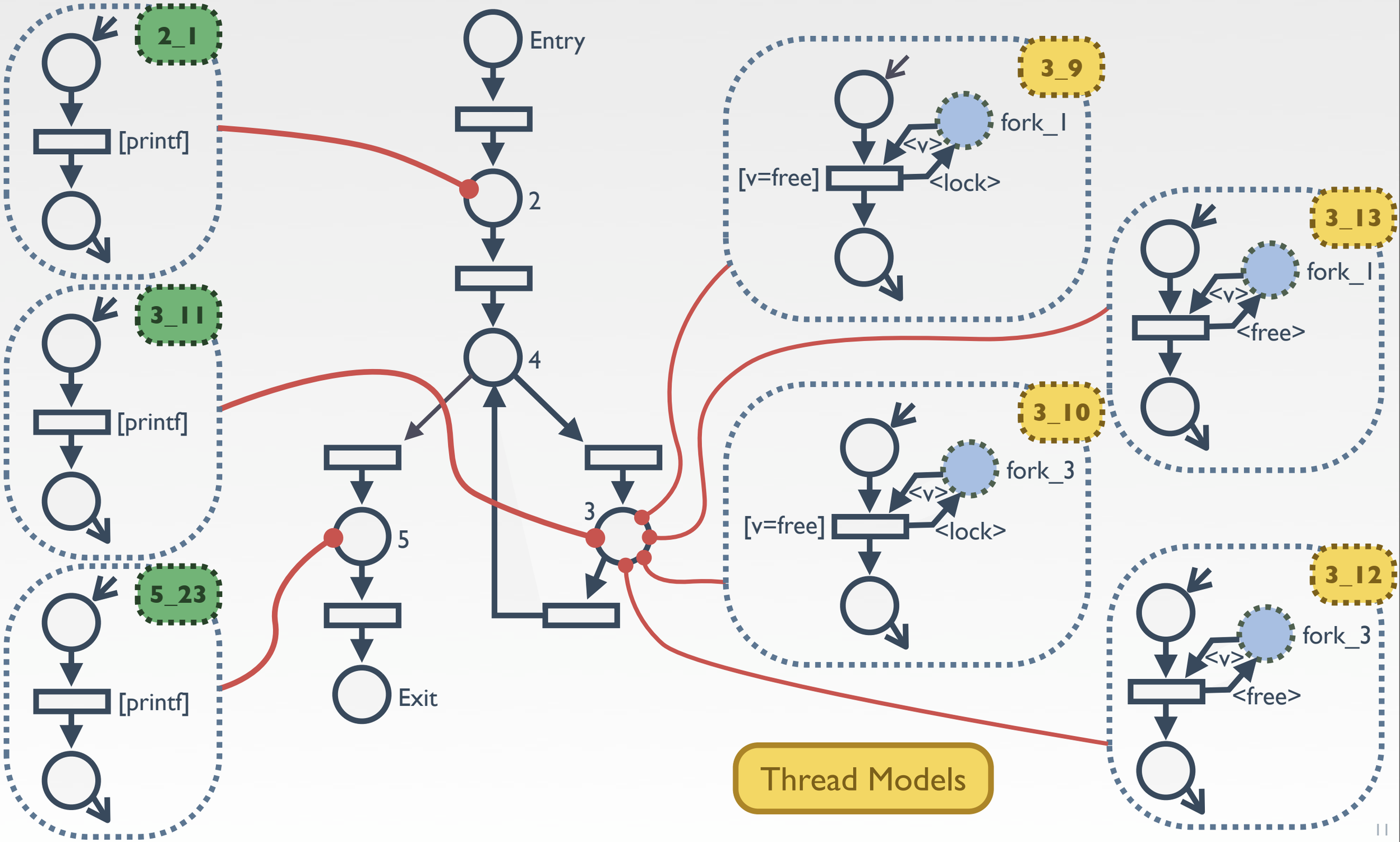


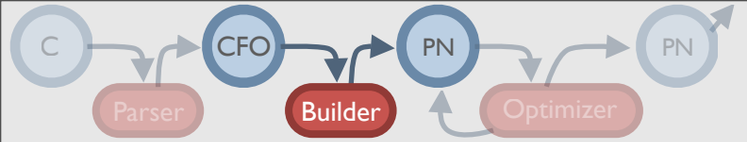
Struct Model

Philos'R'Nets (still small)



Philos'R'Nets (still small)

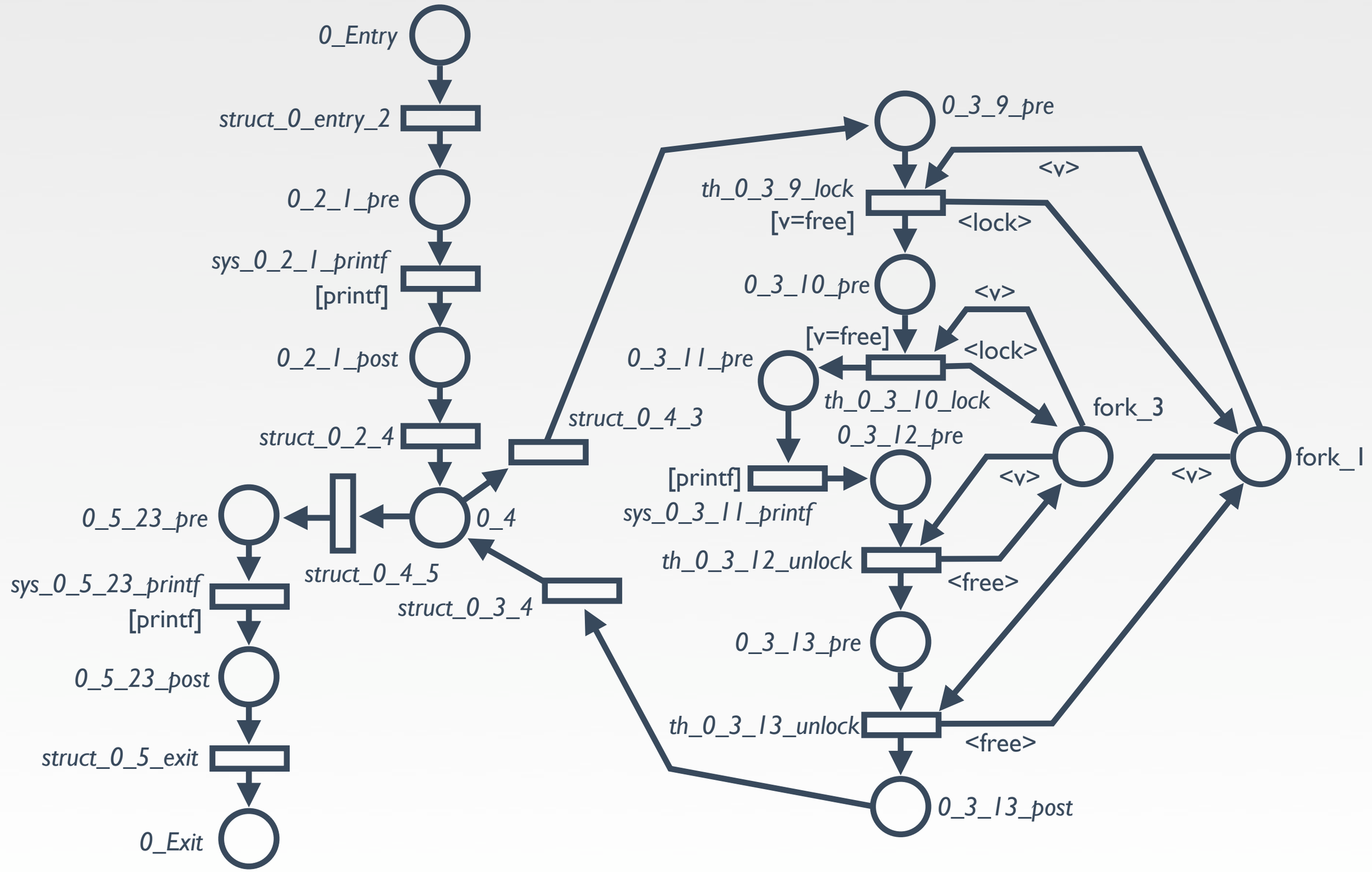




Building (big) Petri Nets

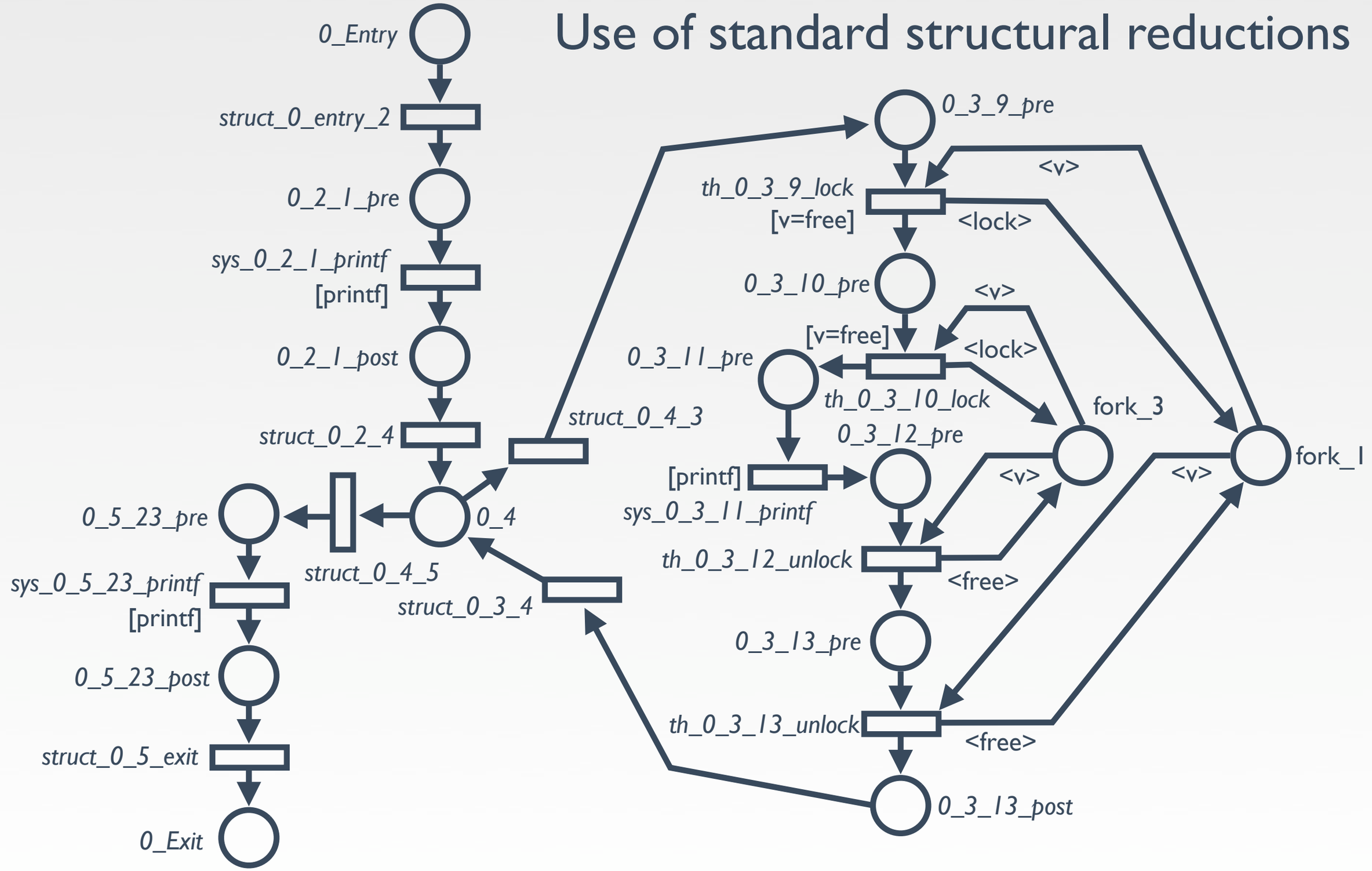
- Merge all subnets into the structural model
  - Find the right order thanks to the ECFG metadata
  - Traceability of the origin of Petri nets elements
  - Manage all specificities of Petri nets to produce correct nets
    - Color classes / Color domains
    - Initial marking
- One net for each “main” function
  - Potentially several nets for large applications
    - When composed of several executables (start / stop / status...)

One philosopher...



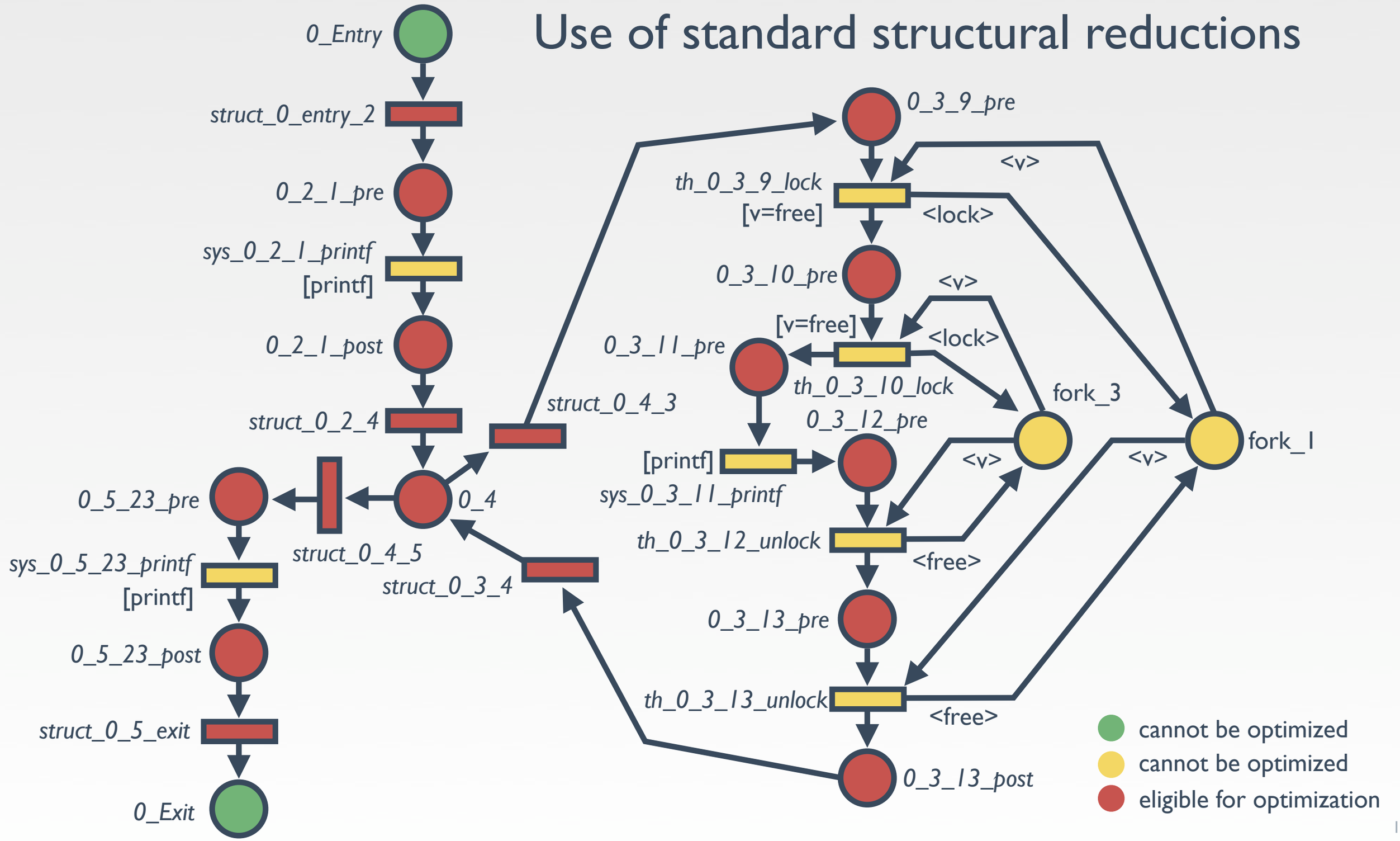
One philosopher...

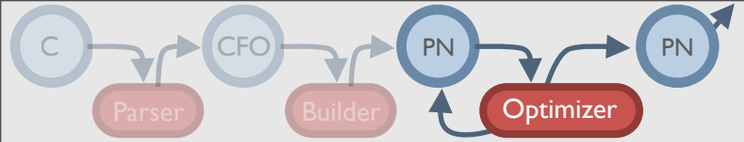
Use of standard structural reductions



One philosopher...

Use of standard structural reductions



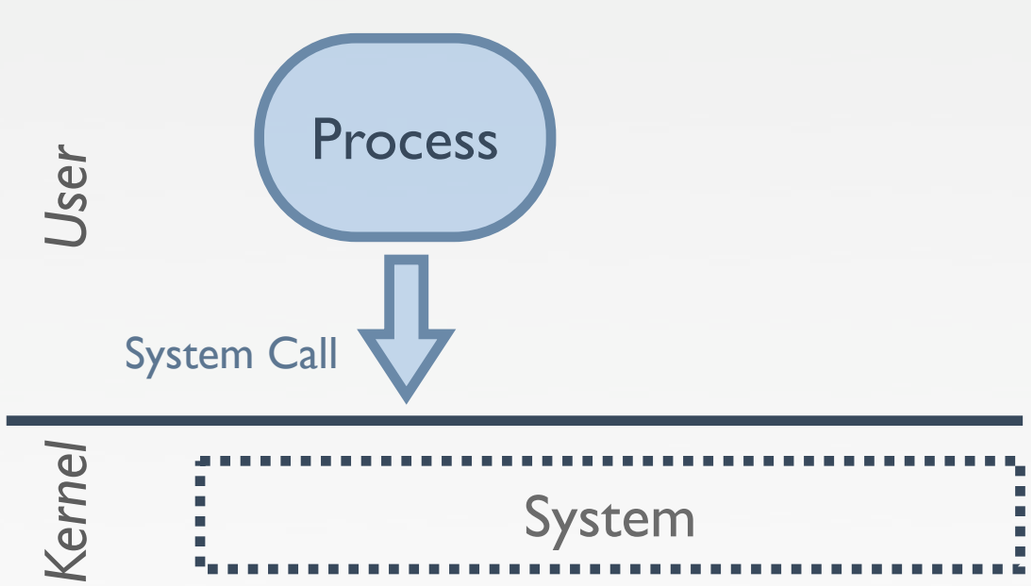


Offline Analysis

- Some basic properties can be checked before runtime
- Structural properties
  - Dead code / Infinite loops
- Reachability properties
  - Dead code / Deadlock
- Causal properties
  - Starvation / Race Condition
  - User-specified: “Can a read occurs on a file after I closed it”

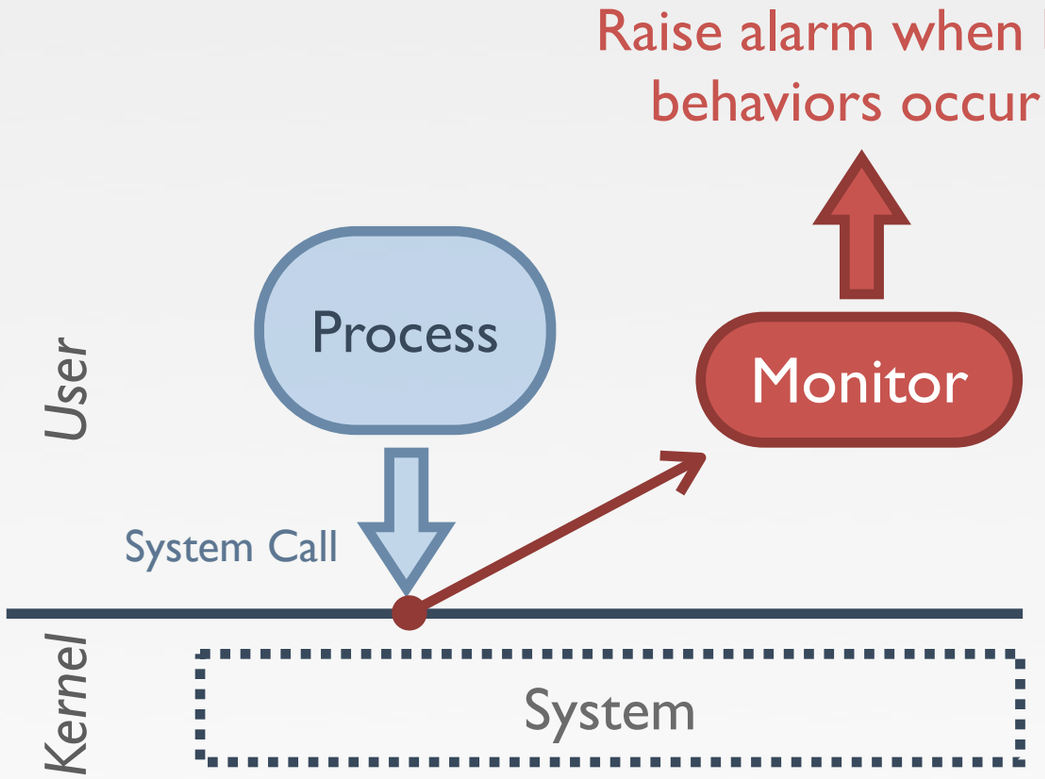
From Models to Monitor

- Objective : building a dedicated program monitor



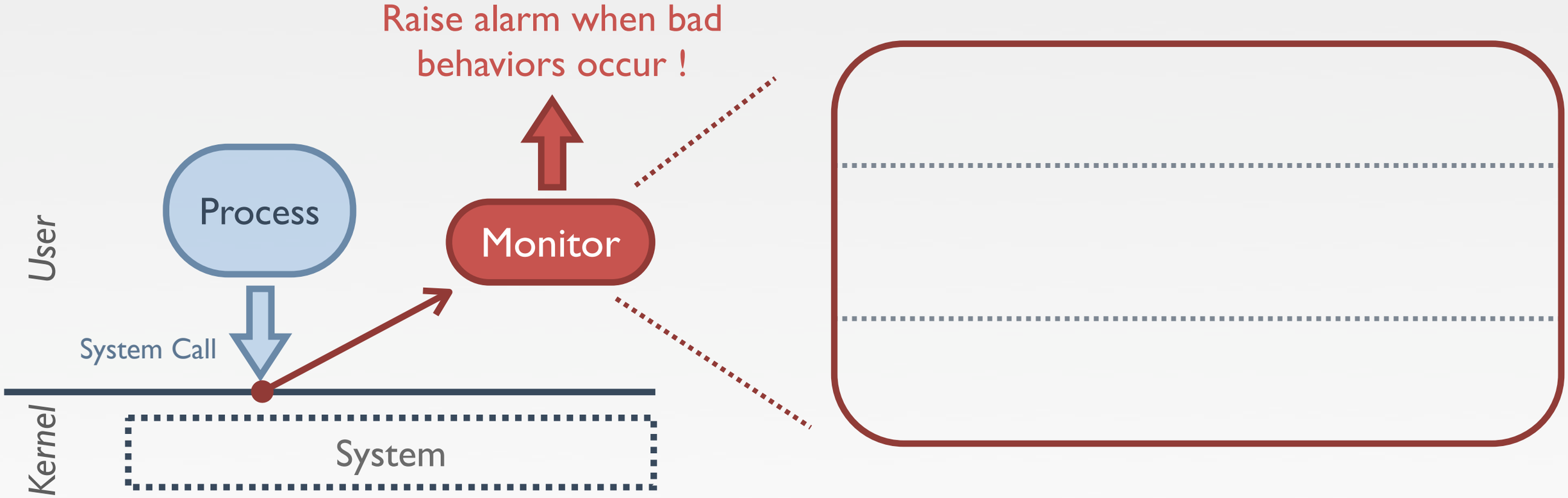
From Models to Monitor

- Objective : building a dedicated program monitor



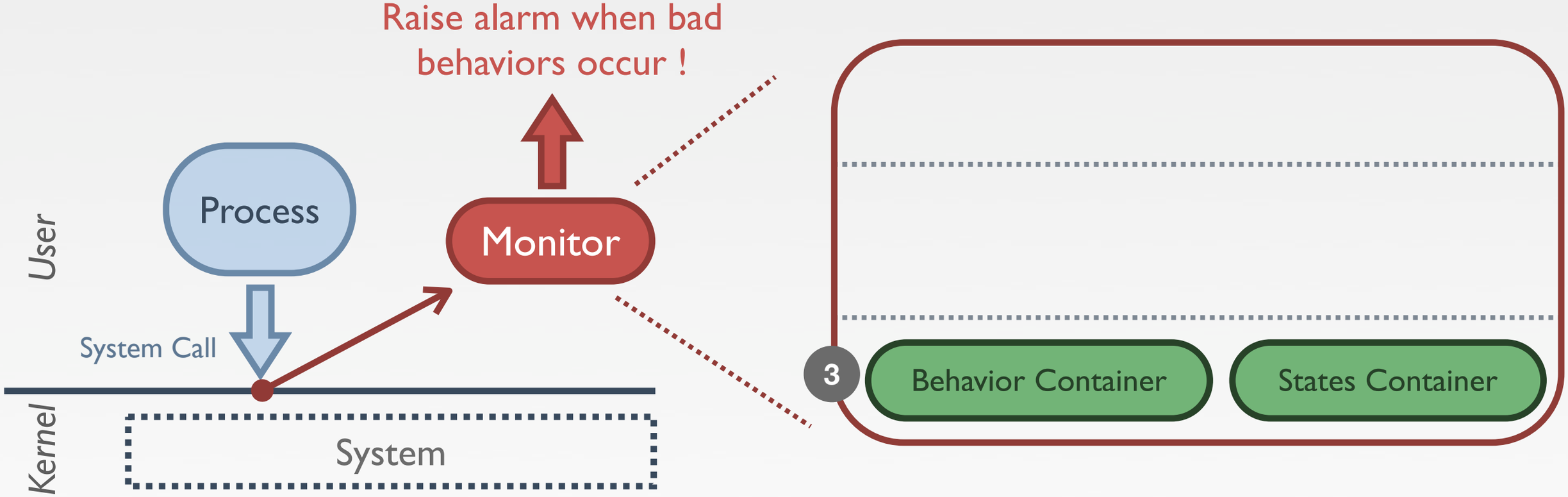
From Models to Monitor

- Objective : building a dedicated program monitor



From Models to Monitor

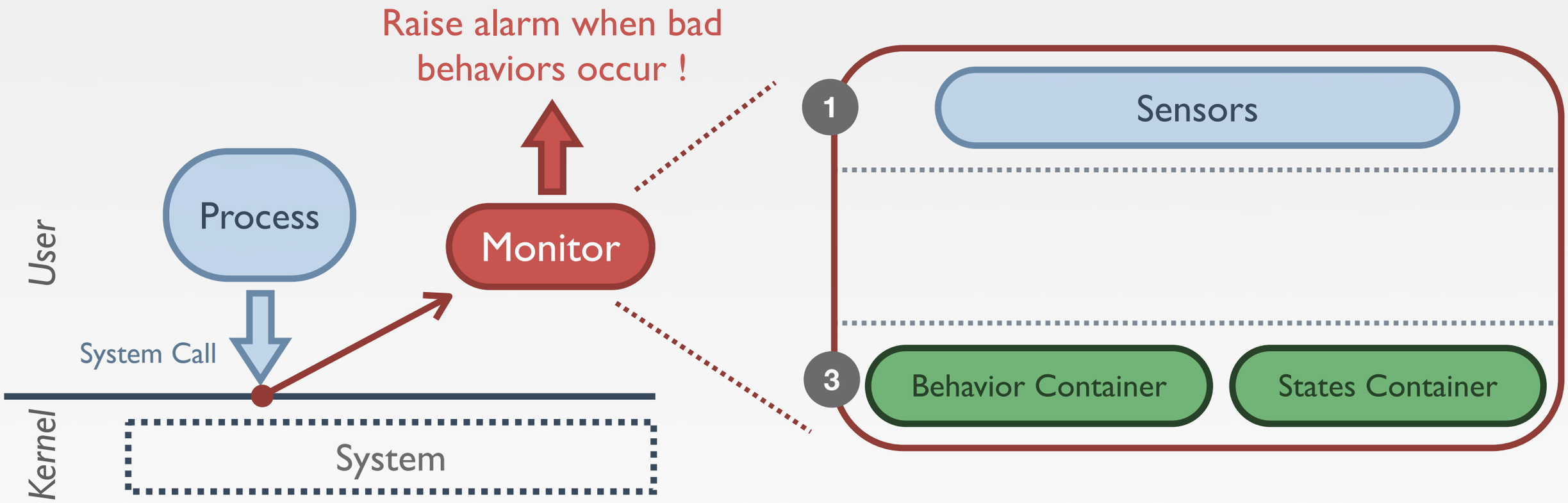
- Objective : building a dedicated program monitor



✓ Embed the Petri net built during previous phase

From Models to Monitor

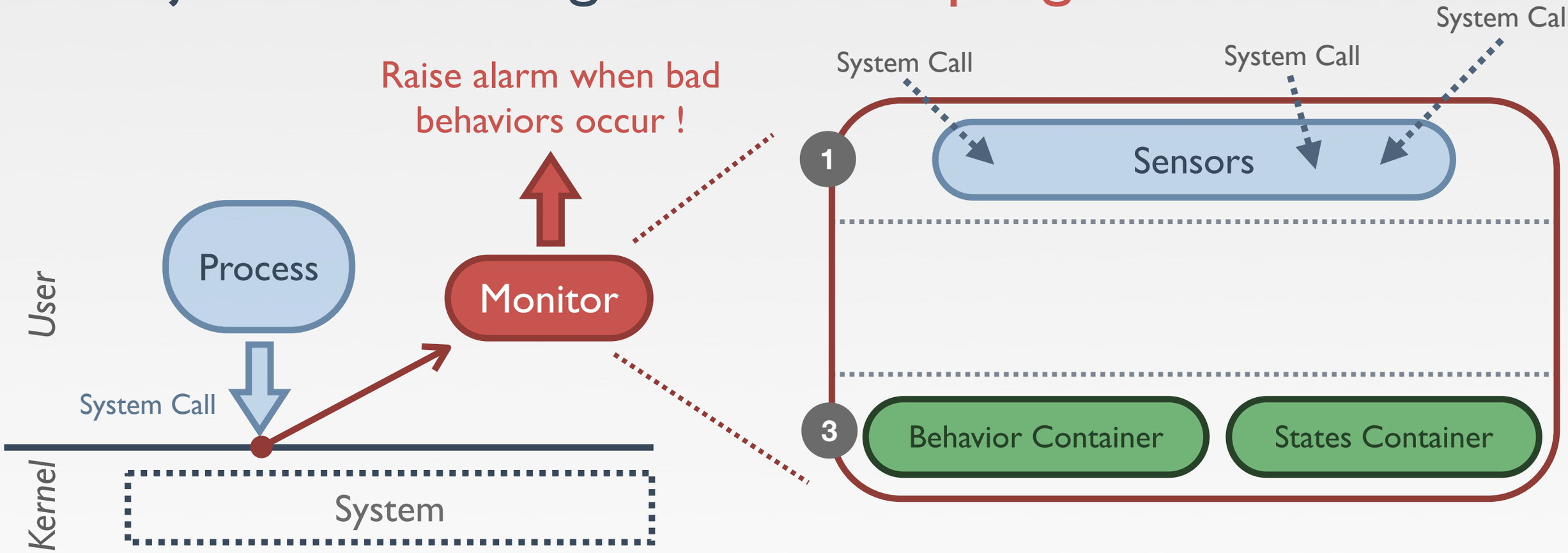
- Objective : building a dedicated program monitor



✓ Embed the Petri net built during previous phase

From Models to Monitor

- Objective : building a dedicated program monitor

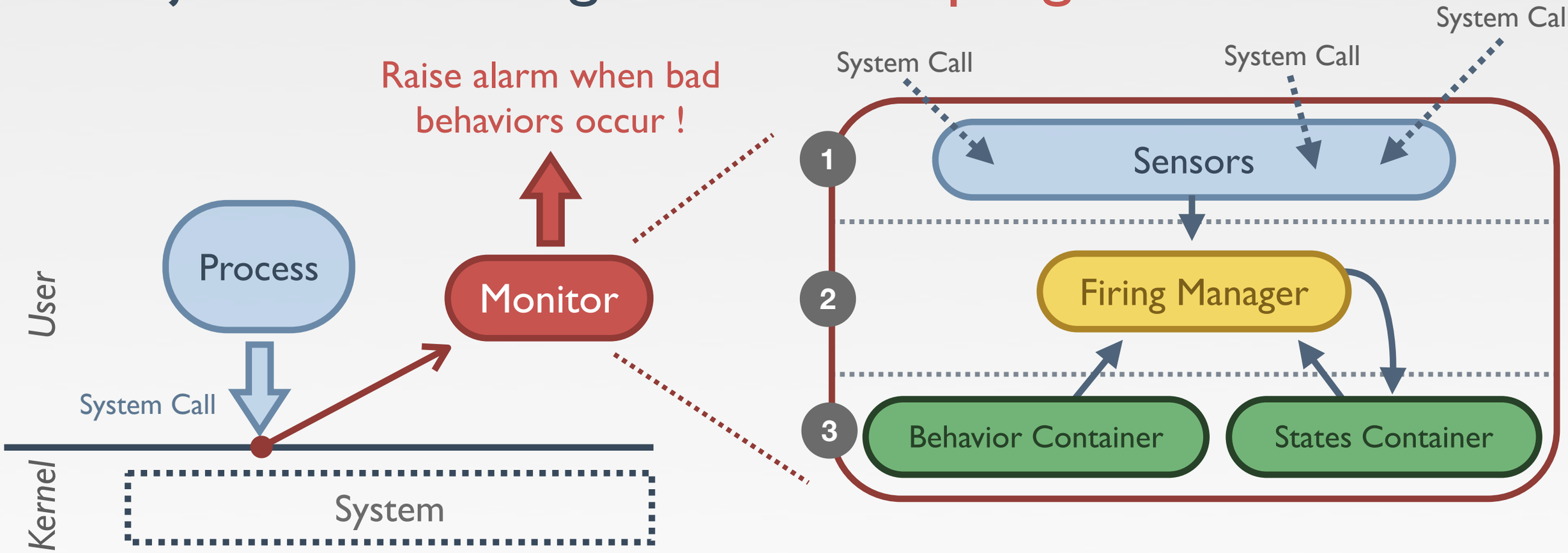


✓ Embed the Petri net built during previous phase

✓ Catch events related to the program execution

From Models to Monitor

- Objective : building a dedicated program monitor



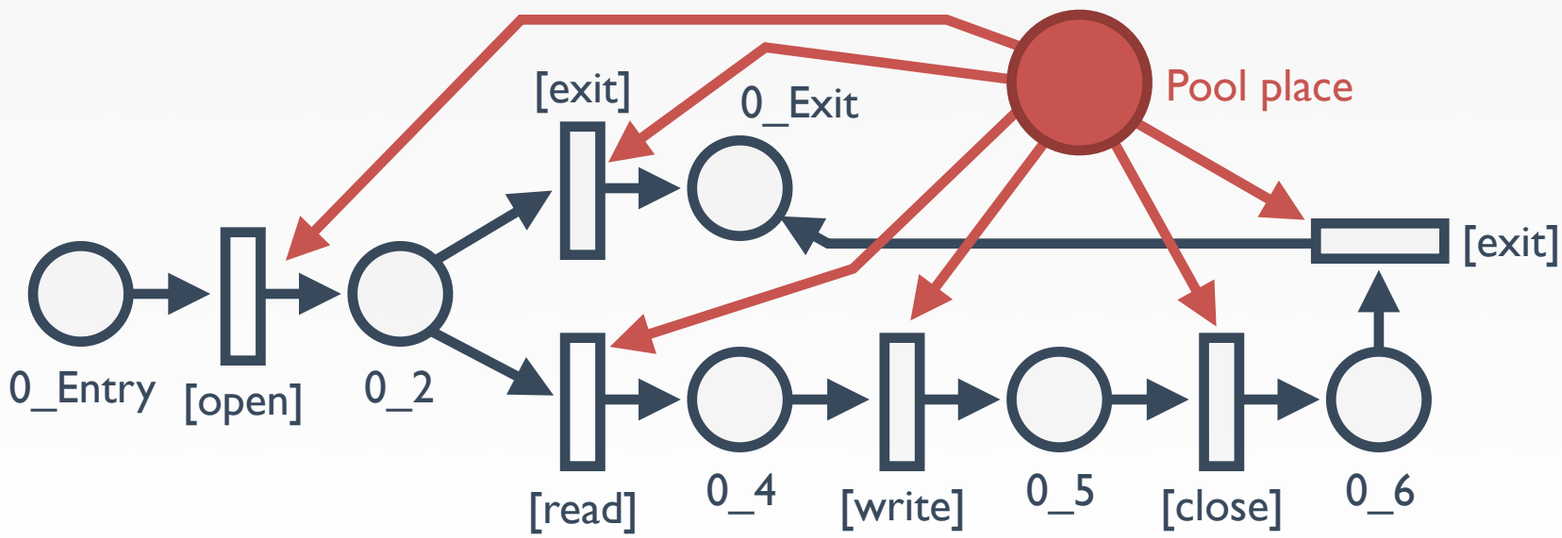
✓ Embed the Petri net built during previous phase

✓ Catch events related to the program execution

✓ Compare real events to expected events

Playing with Nets

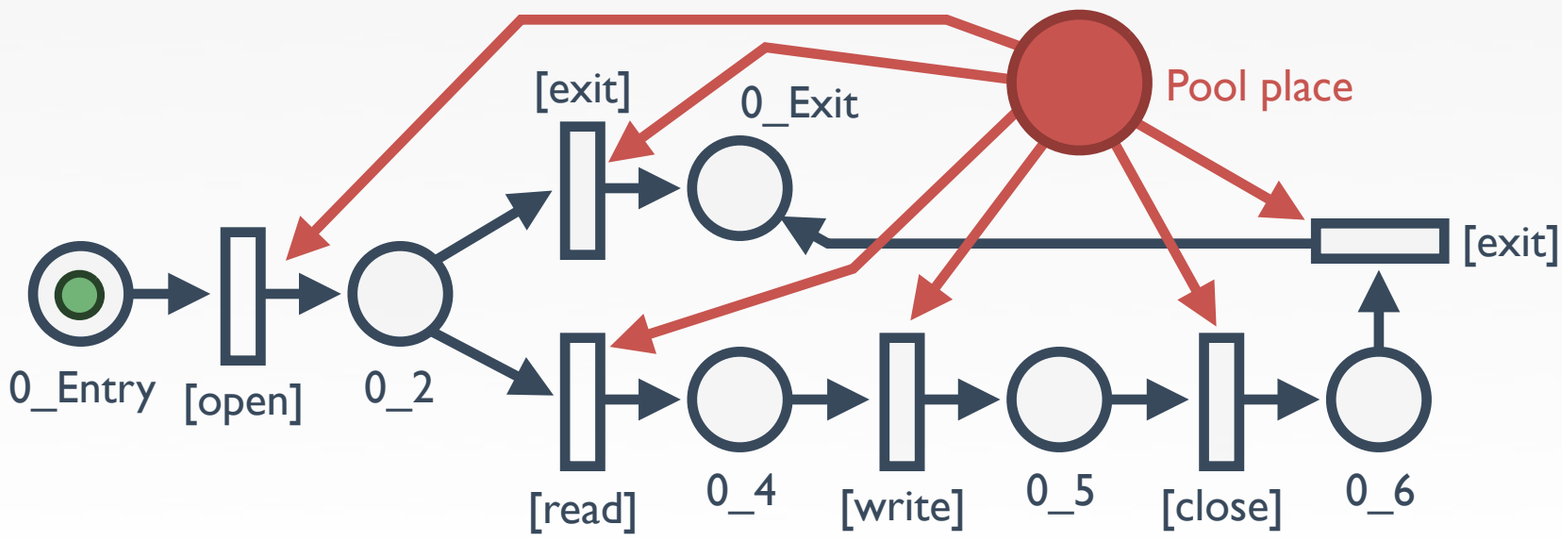
- All transitions can consume tokens from pool place
- Execution is divided into rounds
  - Catching an event = beginning of a round
  - When the pool place is empty = end of a round
  - The pool place must be empty before beginning a new round



Event	Color
open	“o”
read	“r”
write	“w”
close	“c”
exit	“e”

Playing with Nets

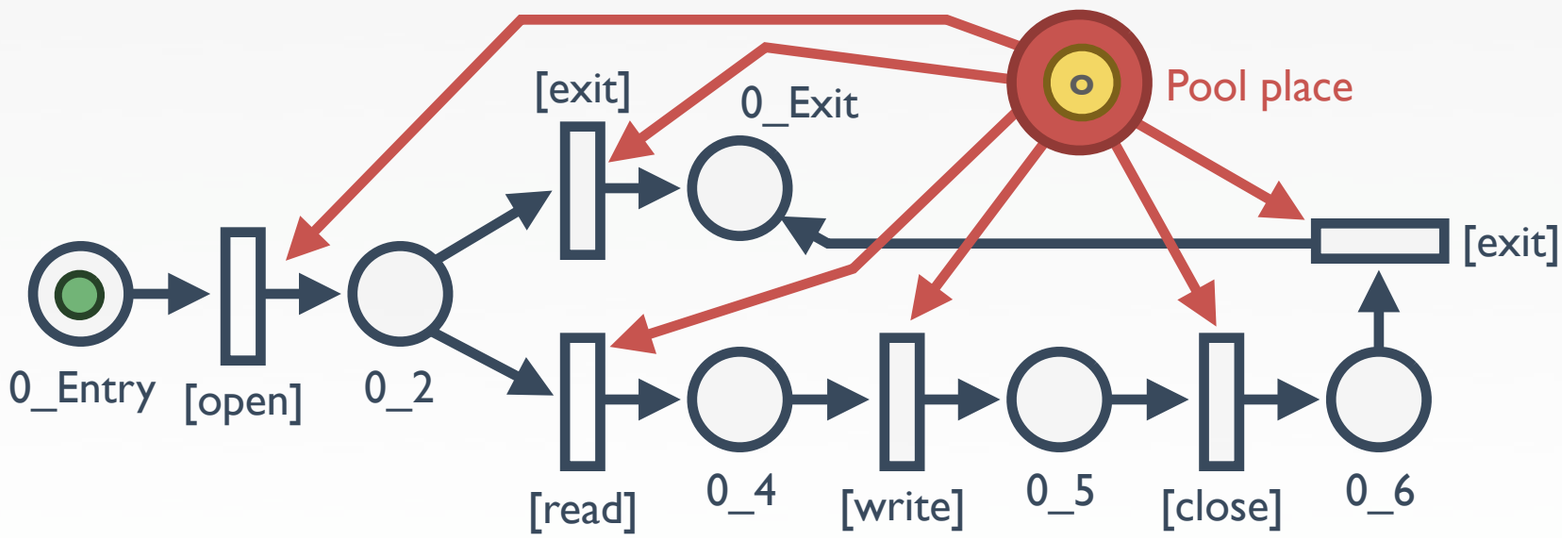
- All transitions can consume tokens from pool place
- Execution is divided into rounds
  - Catching an event = beginning of a round
  - When the pool place is empty = end of a round
  - The pool place must be empty before beginning a new round



Event	Color
open	“o”
read	“r”
write	“w”
close	“c”
exit	“e”

Playing with Nets

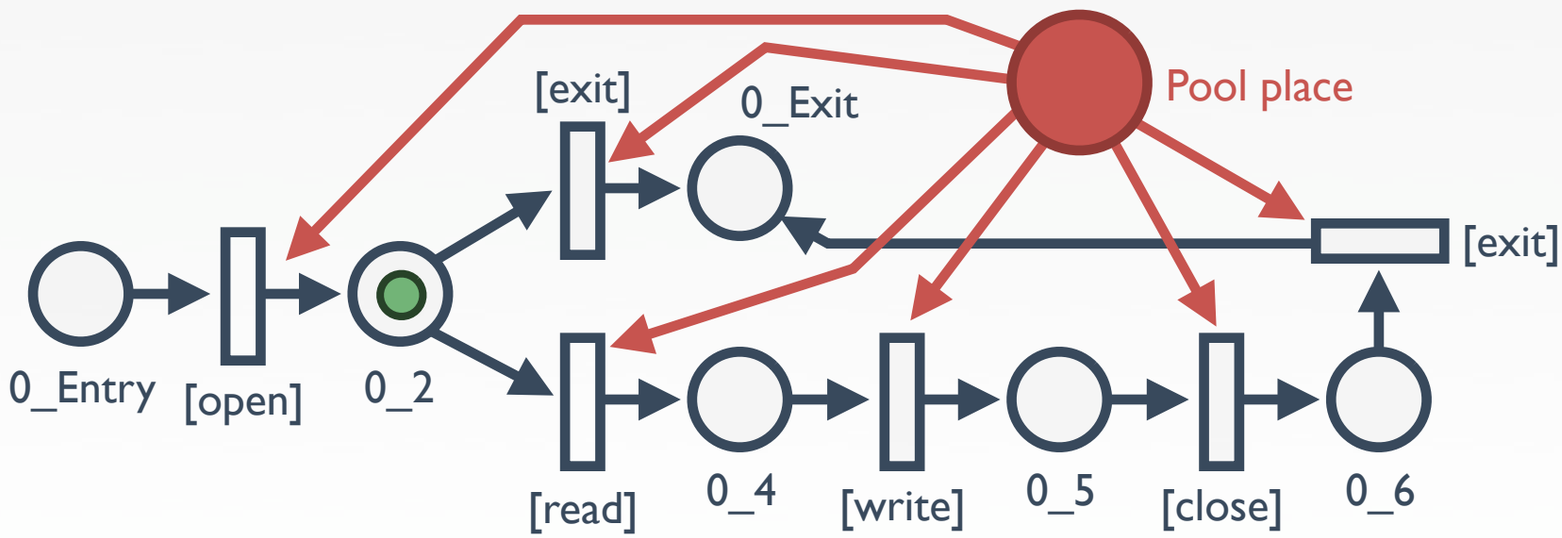
- All transitions can consume tokens from pool place
- Execution is divided into rounds
  - Catching an event = beginning of a round
  - When the pool place is empty = end of a round
  - The pool place must be empty before beginning a new round



Event	Color
open	“o”
read	“r”
write	“w”
close	“c”
exit	“e”

Playing with Nets

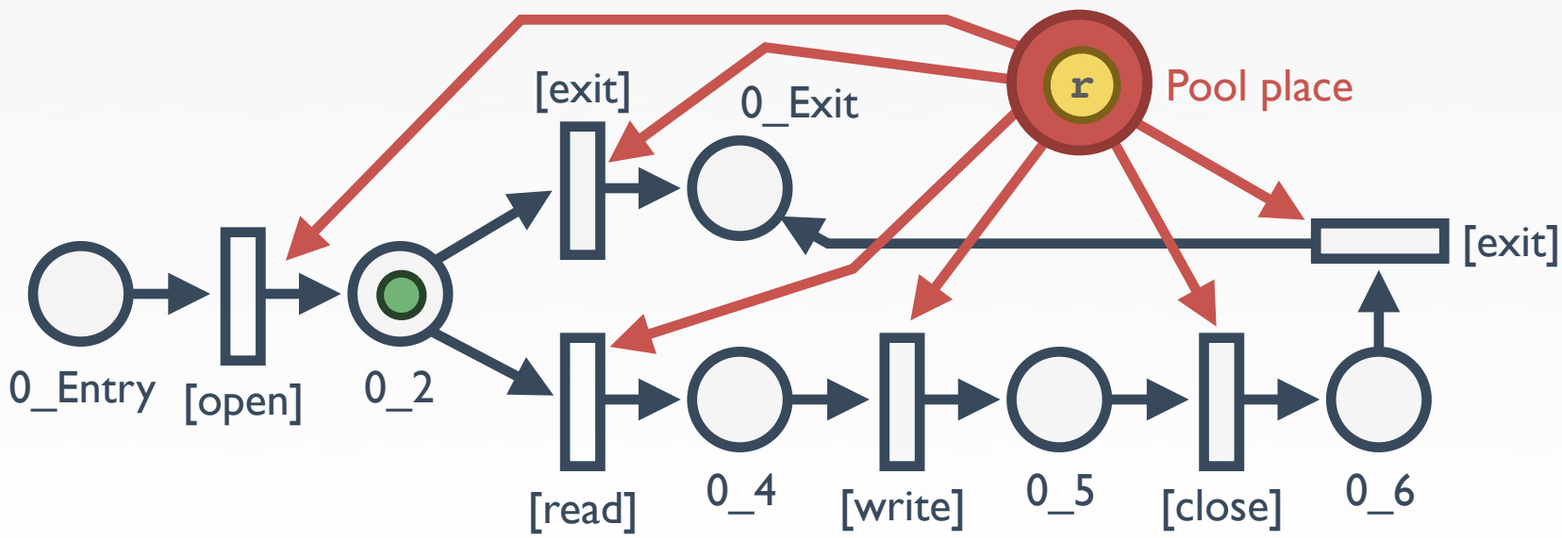
- All transitions can consume tokens from pool place
- Execution is divided into rounds
  - Catching an event = beginning of a round
  - When the pool place is empty = end of a round
  - The pool place must be empty before beginning a new round



Event	Color
open	“o”
read	“r”
write	“w”
close	“c”
exit	“e”

Playing with Nets

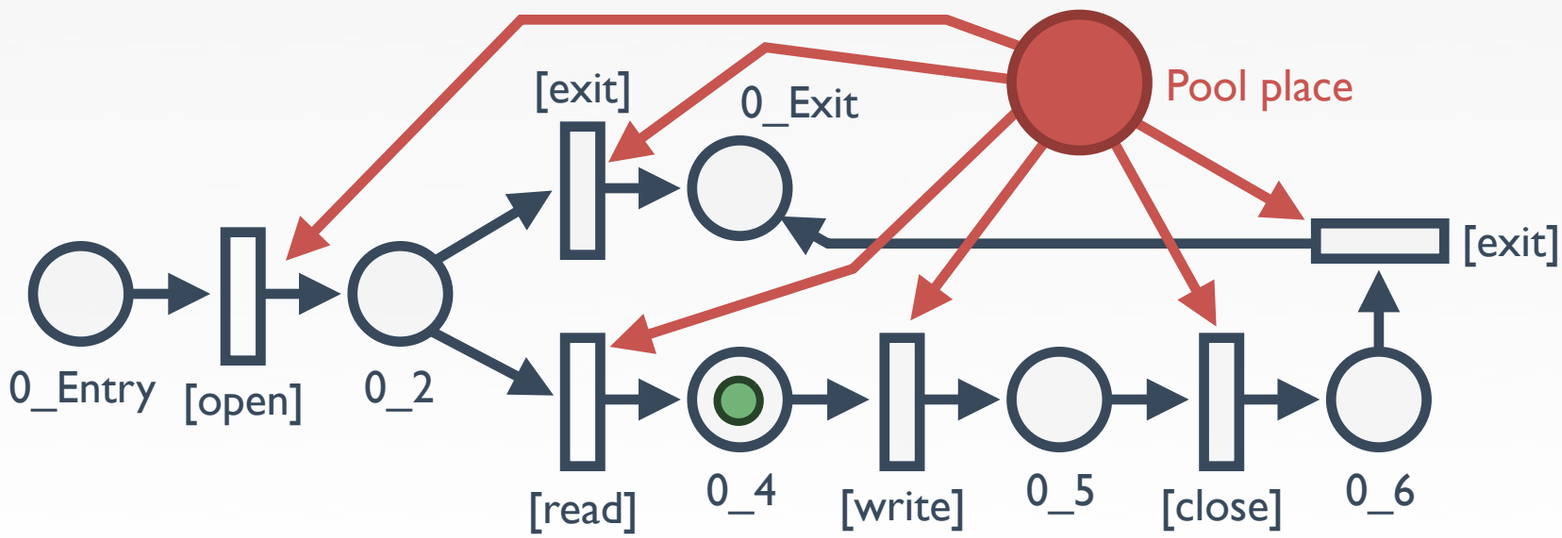
- All transitions can consume tokens from pool place
- Execution is divided into rounds
  - Catching an event = beginning of a round
  - When the pool place is empty = end of a round
  - The pool place must be empty before beginning a new round



Event	Color
open	“o”
read	“r”
write	“w”
close	“c”
exit	“e”

Playing with Nets

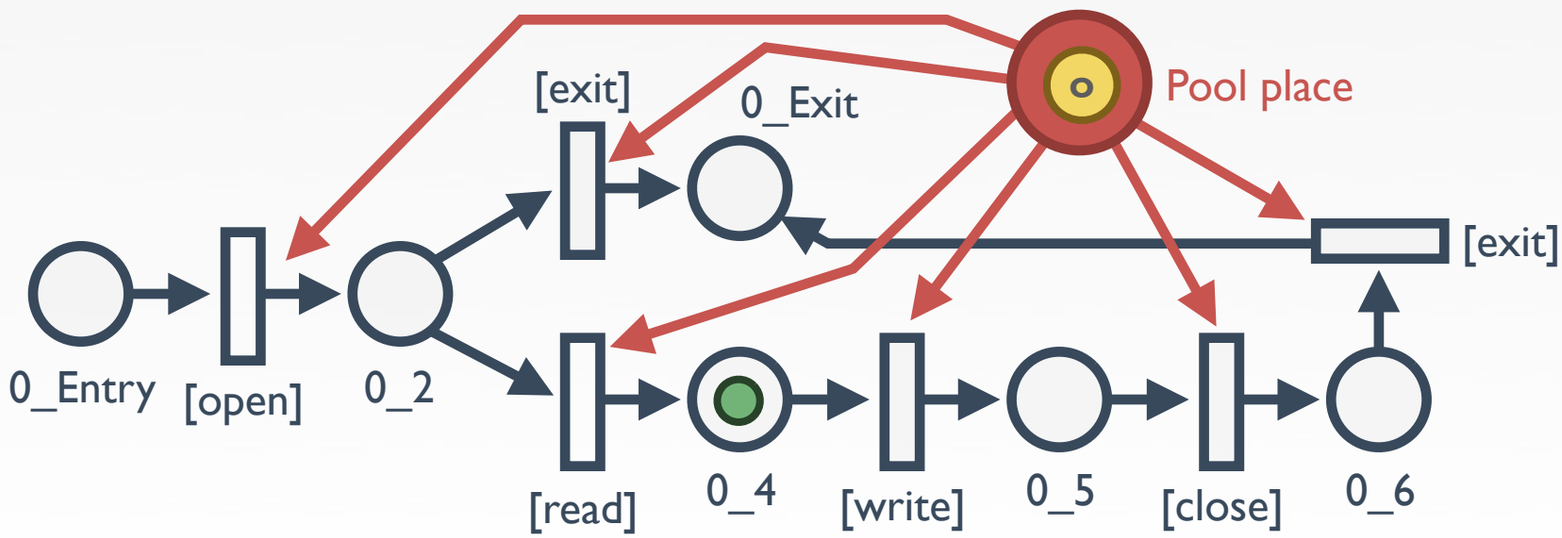
- All transitions can consume tokens from pool place
- Execution is divided into rounds
  - Catching an event = beginning of a round
  - When the pool place is empty = end of a round
  - The pool place must be empty before beginning a new round



Event	Color
open	“o”
read	“r”
write	“w”
close	“c”
exit	“e”

Playing with Nets

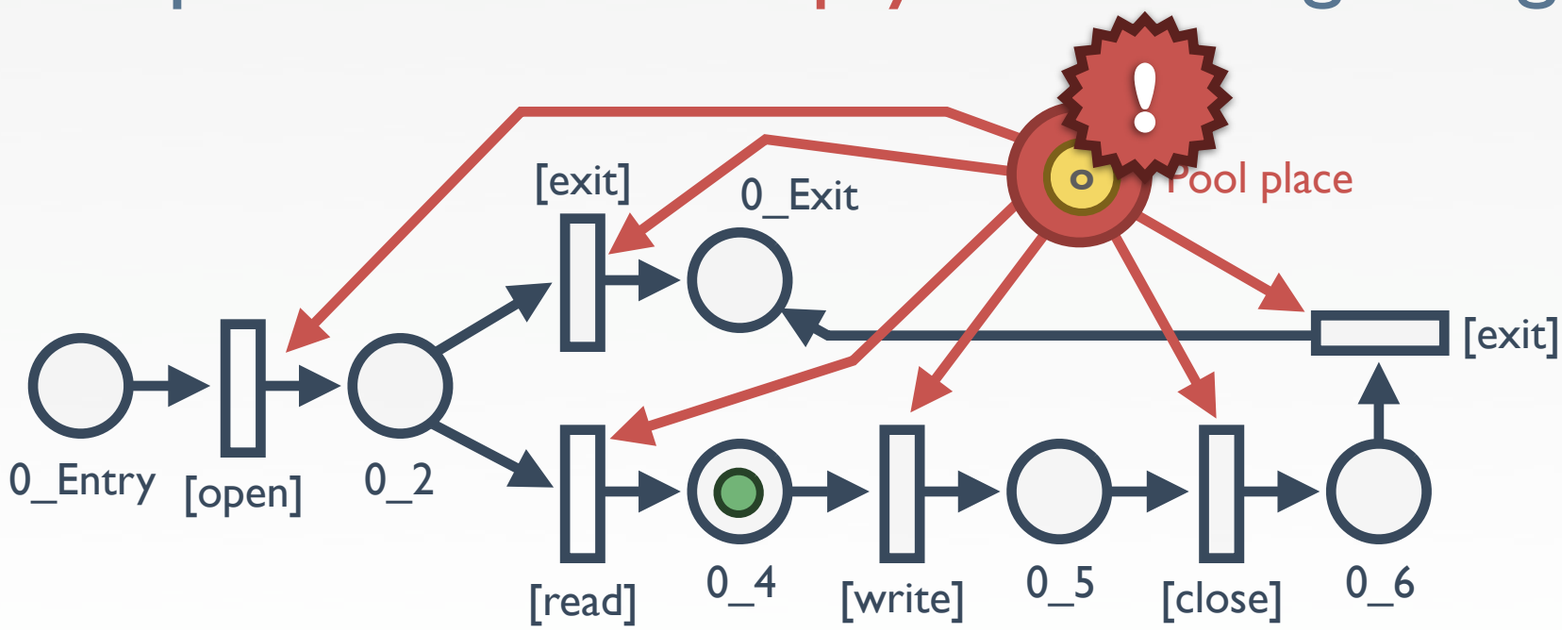
- All transitions can consume tokens from pool place
- Execution is divided into rounds
  - Catching an event = beginning of a round
  - When the pool place is empty = end of a round
  - The pool place must be empty before beginning a new round



Event	Color
open	“o”
read	“r”
write	“w”
close	“c”
exit	“e”

Playing with Nets

- All transitions can consume tokens from pool place
- Execution is divided into rounds
  - Catching an event = beginning of a round
  - When the pool place is empty = end of a round
  - The pool place must be empty before beginning a new round

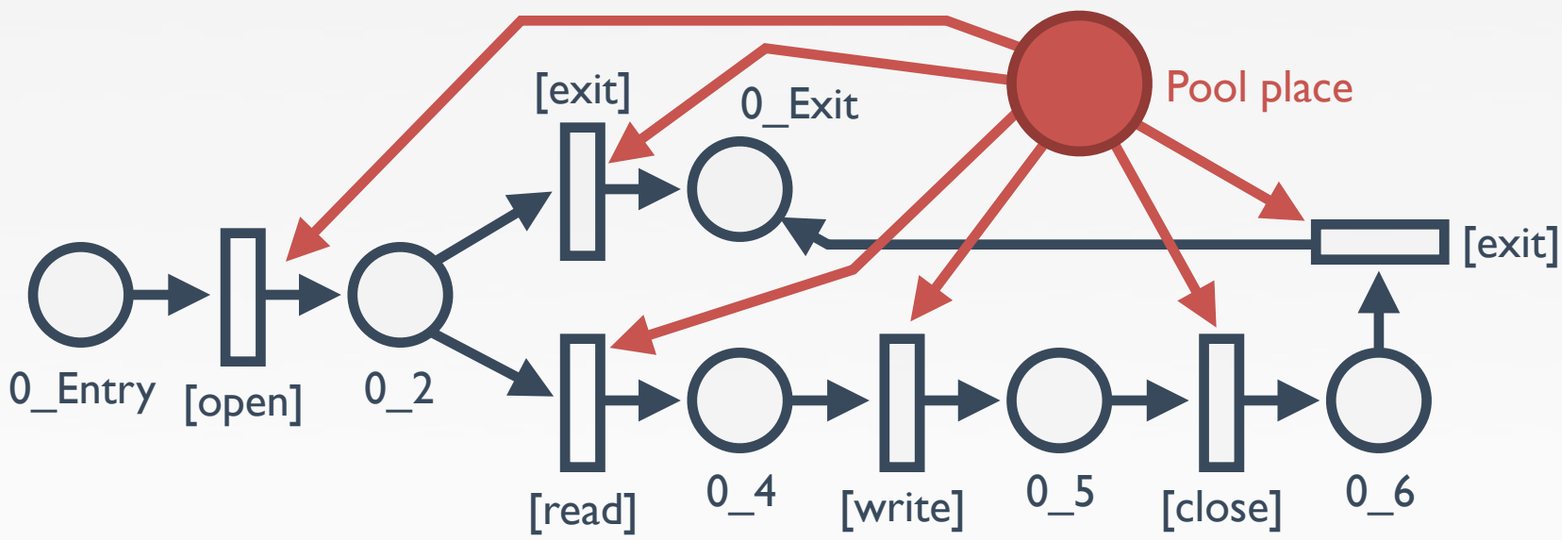


Event	Color
open	“o”
read	“r”
write	“w”
close	“c”
exit	“e”

Playing with Nets & Stack !

- Not sufficient to handle mimicry attacks
  - Attacks that mimic a correct behavior but doing "*bad things*"
    - System call sequence of these attacks is correct
    - Introduce anomalies into call stack

Event	Color
open	"o"
read	"r"
write	"w"
close	"c"
exit	"e"

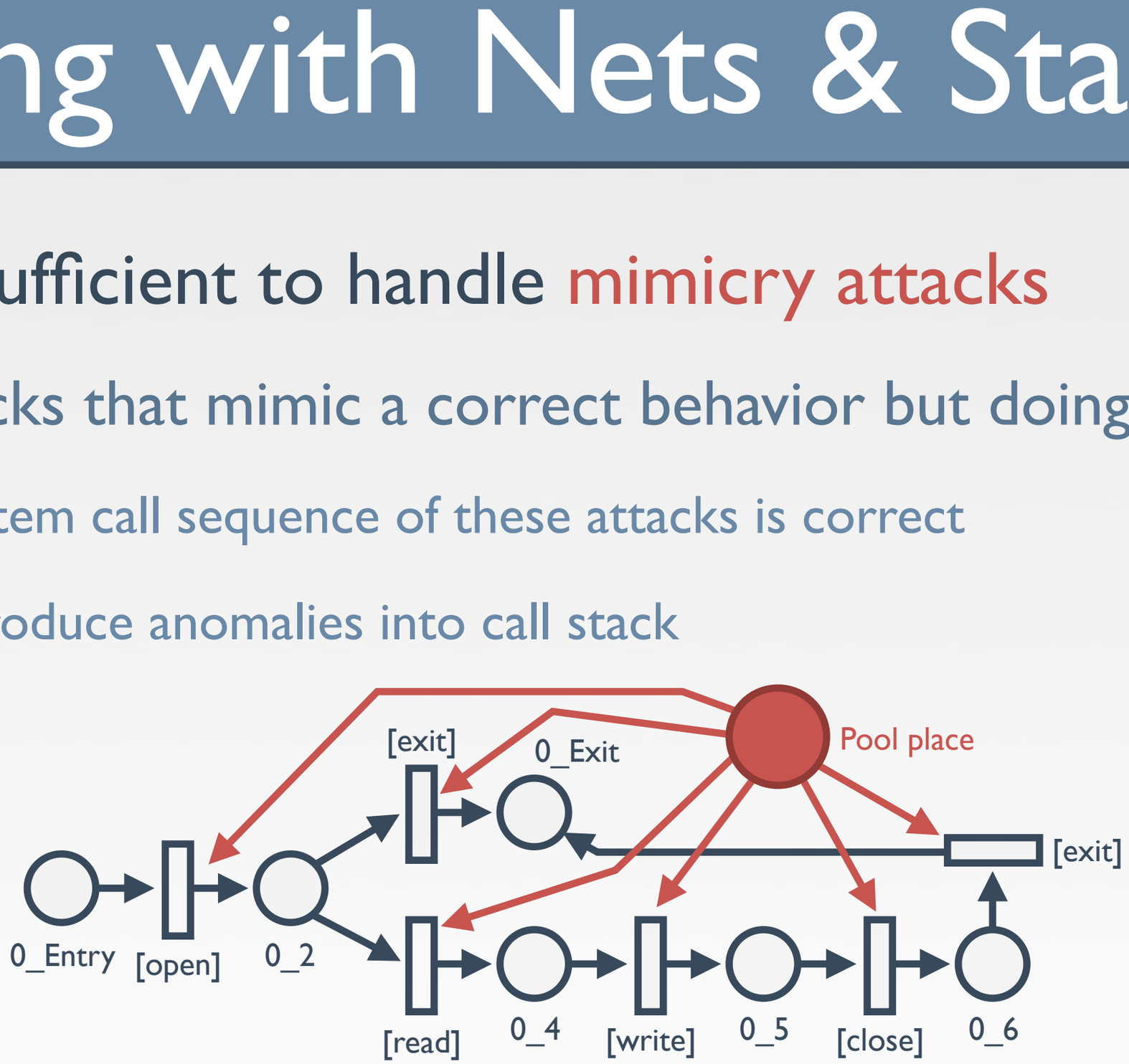


Playing with Nets & Stack !

- Not sufficient to handle mimicry attacks
  - Attacks that mimic a correct behavior but doing "*bad things*"
    - System call sequence of these attacks is correct
    - Introduce anomalies into call stack

Event	Color
open	"o"
read	"r"
write	"w"
close	"c"
exit	"e"

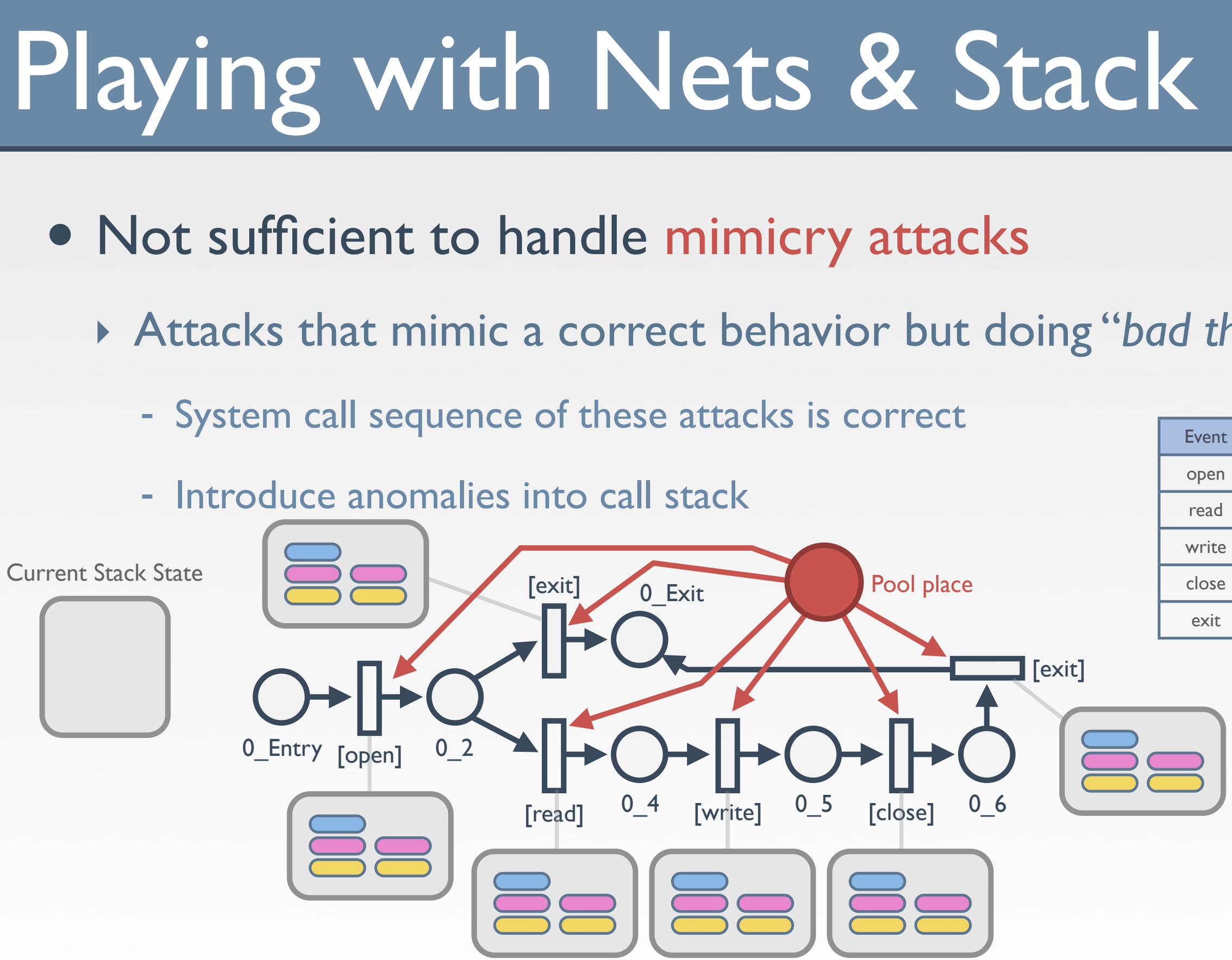
Current Stack State



Playing with Nets & Stack !

- Not sufficient to handle mimicry attacks
  - Attacks that mimic a correct behavior but doing "*bad things*"
    - System call sequence of these attacks is correct
    - Introduce anomalies into call stack

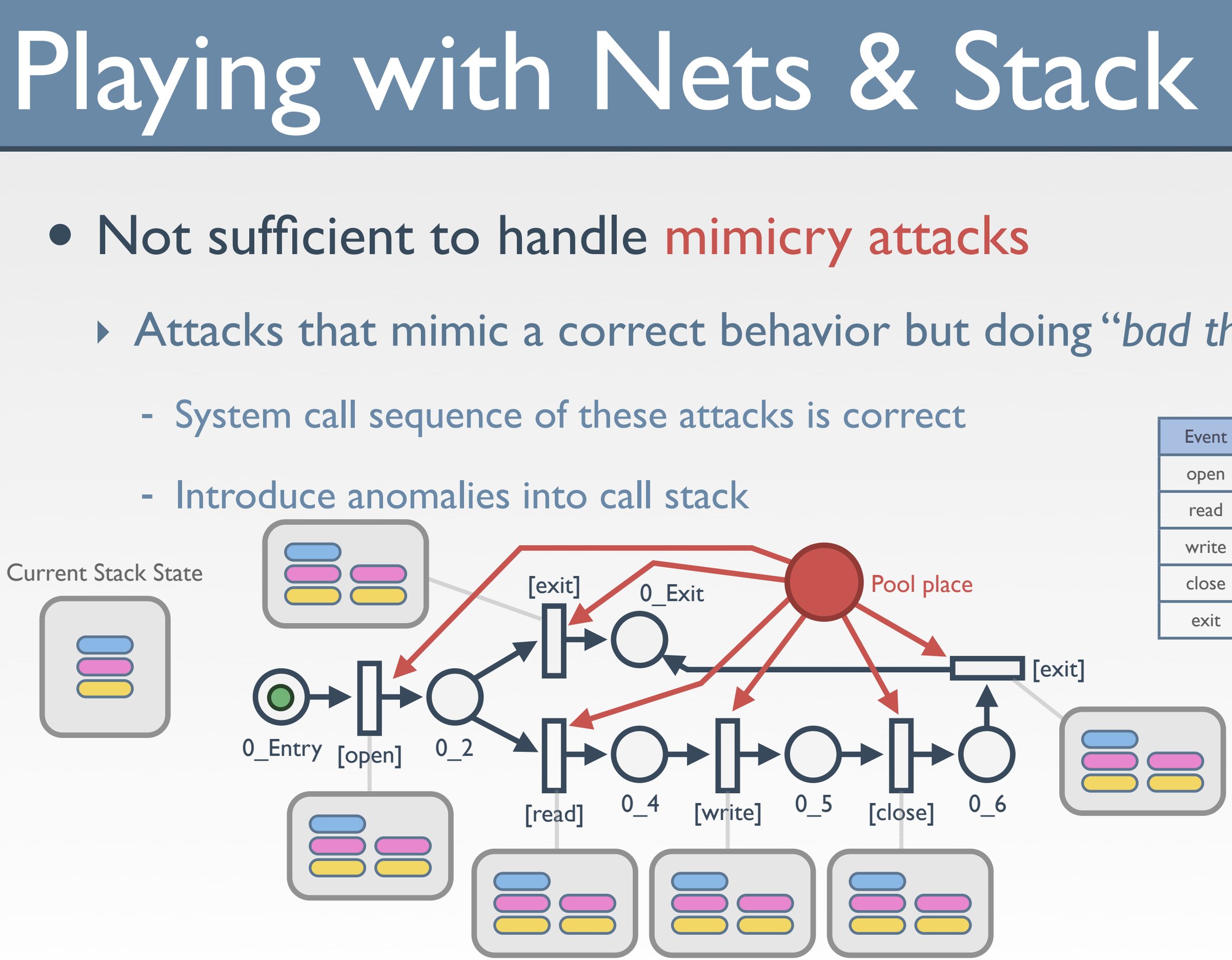
Event	Color
open	"o"
read	"r"
write	"w"
close	"c"
exit	"e"



Playing with Nets & Stack !

- Not sufficient to handle mimicry attacks
  - Attacks that mimic a correct behavior but doing "*bad things*"
    - System call sequence of these attacks is correct
    - Introduce anomalies into call stack

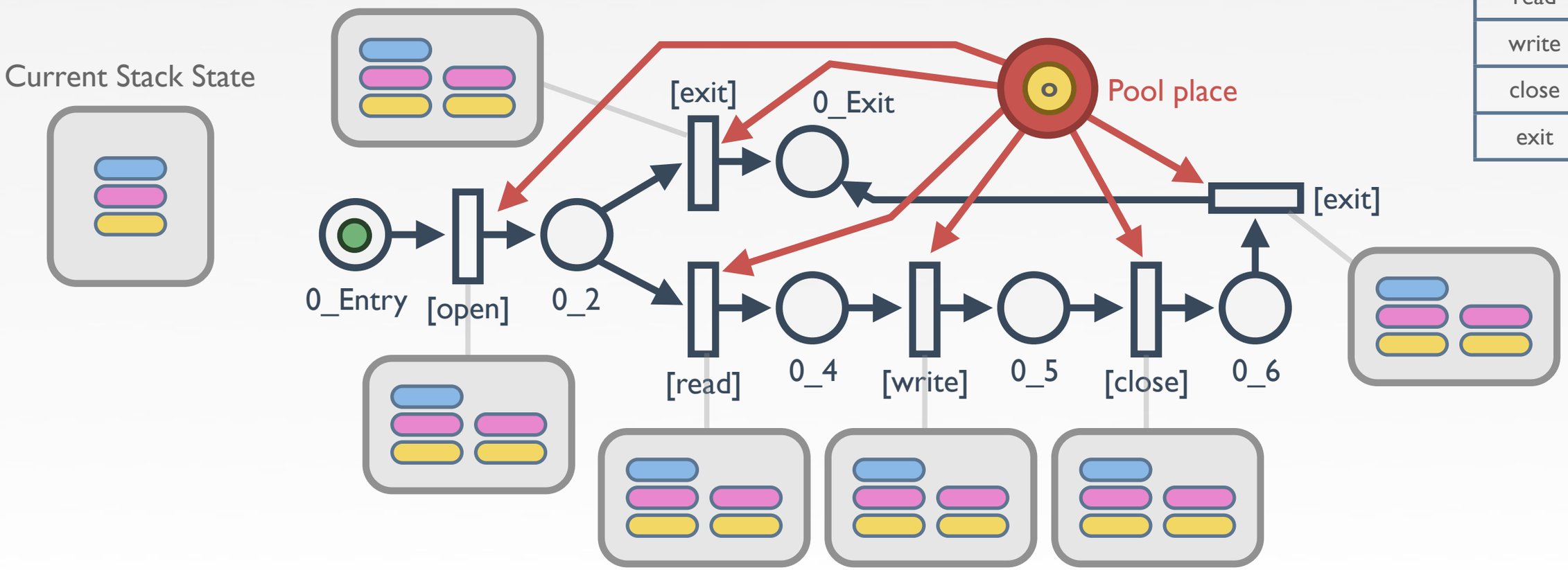
Event	Color
open	"o"
read	"r"
write	"w"
close	"c"
exit	"e"



Playing with Nets & Stack !

- Not sufficient to handle mimicry attacks
  - Attacks that mimic a correct behavior but doing "*bad things*"
    - System call sequence of these attacks is correct
    - Introduce anomalies into call stack

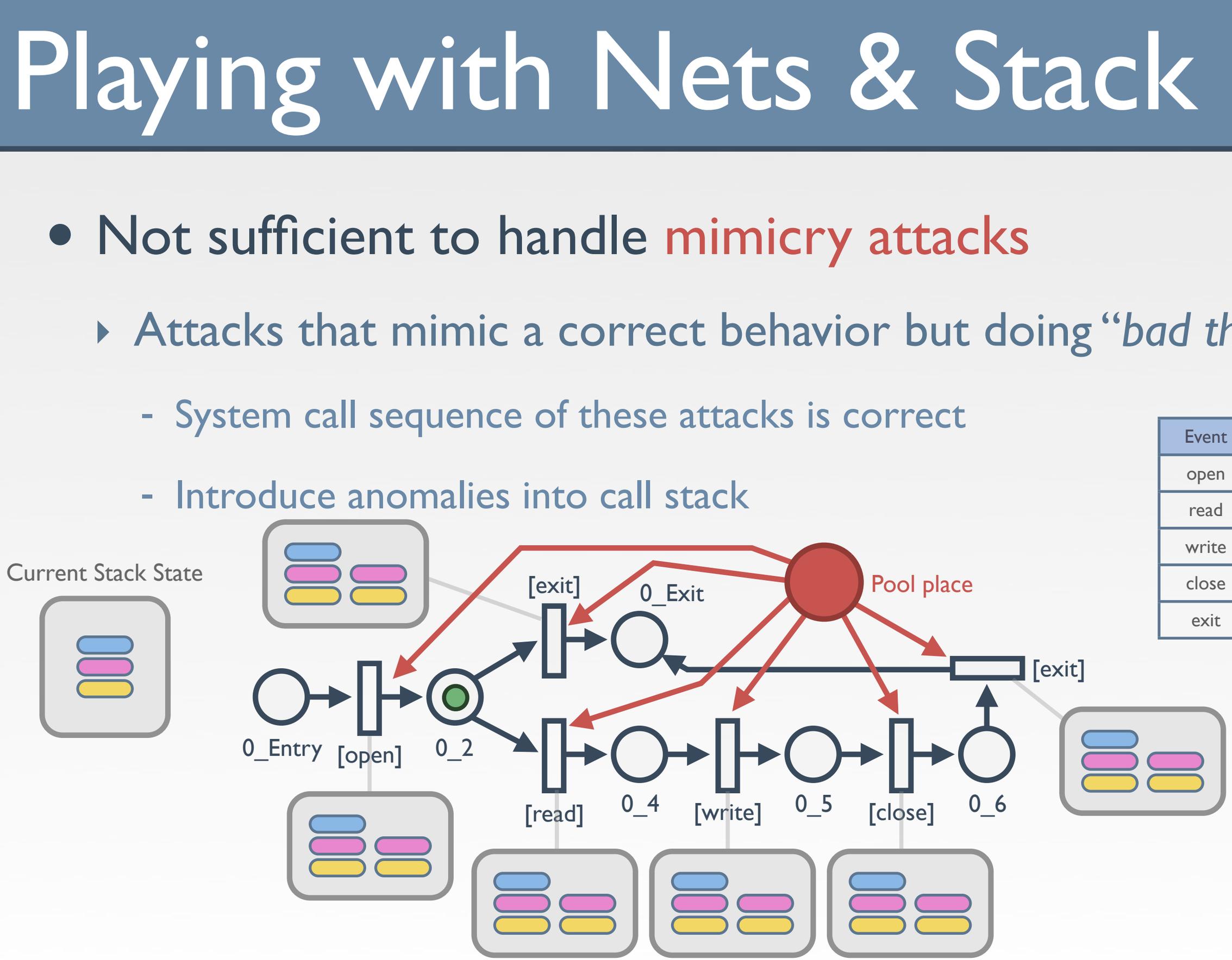
Event	Color
open	"o"
read	"r"
write	"w"
close	"c"
exit	"e"



Playing with Nets & Stack !

- Not sufficient to handle mimicry attacks
  - Attacks that mimic a correct behavior but doing "*bad things*"
    - System call sequence of these attacks is correct
    - Introduce anomalies into call stack

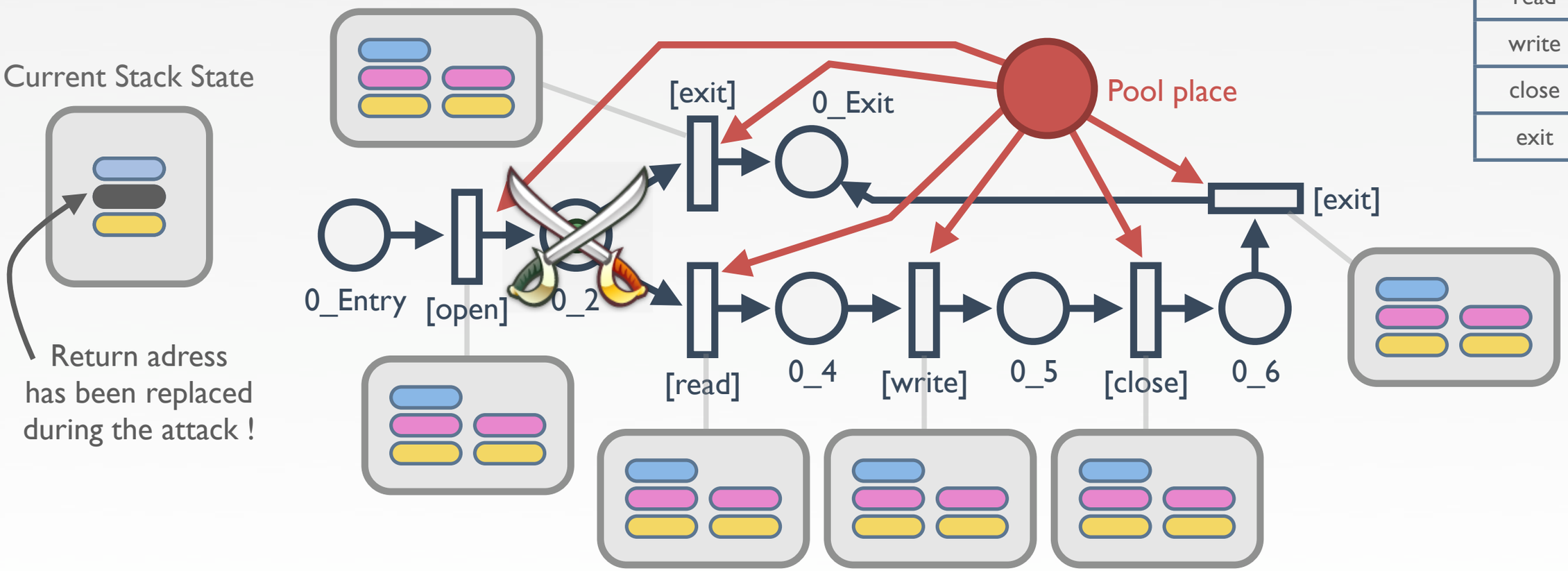
Event	Color
open	"o"
read	"r"
write	"w"
close	"c"
exit	"e"



Playing with Nets & Stack !

- Not sufficient to handle mimicry attacks
  - Attacks that mimic a correct behavior but doing "*bad things*"
    - System call sequence of these attacks is correct
    - Introduce anomalies into call stack

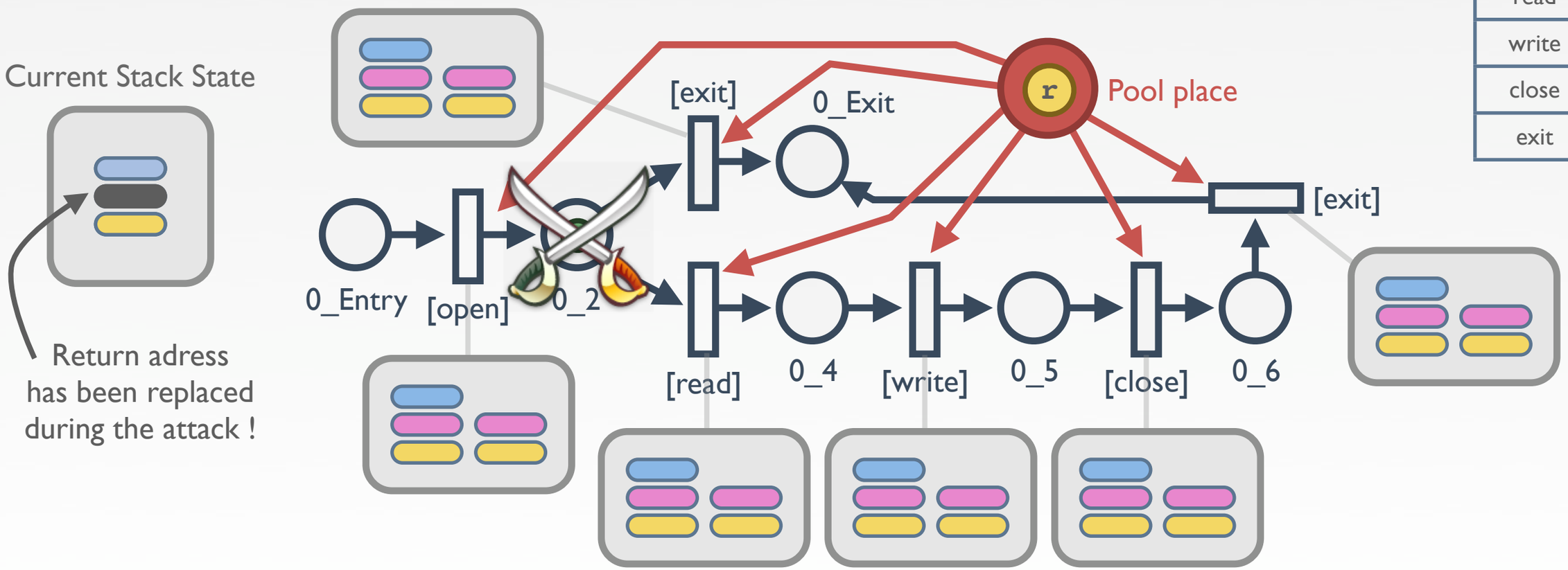
Event	Color
open	"o"
read	"r"
write	"w"
close	"c"
exit	"e"



Playing with Nets & Stack !

- Not sufficient to handle mimicry attacks
  - Attacks that mimic a correct behavior but doing "*bad things*"
    - System call sequence of these attacks is correct
    - Introduce anomalies into call stack

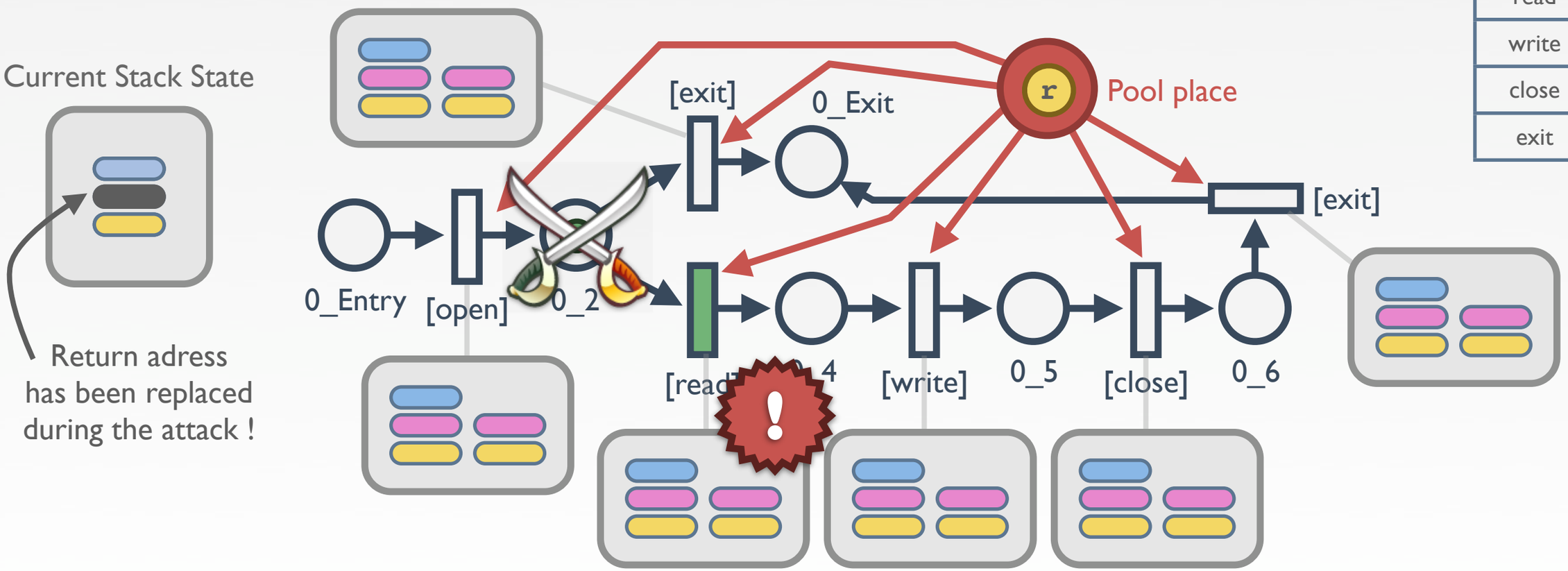
Event	Color
open	"o"
read	"r"
write	"w"
close	"c"
exit	"e"



Playing with Nets & Stack !

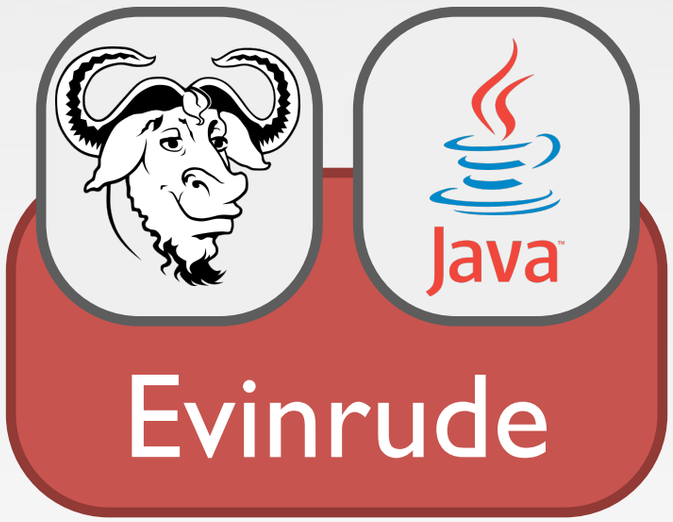
- Not sufficient to handle mimicry attacks
  - Attacks that mimic a correct behavior but doing "*bad things*"
    - System call sequence of these attacks is correct
    - Introduce anomalies into call stack

Event	Color
open	"o"
read	"r"
write	"w"
close	"c"
exit	"e"



Implementation: Evinrude

- Needs GCC
- Written in Java



- Provide flexibility API
- Multi-platform
- Well known by engineers

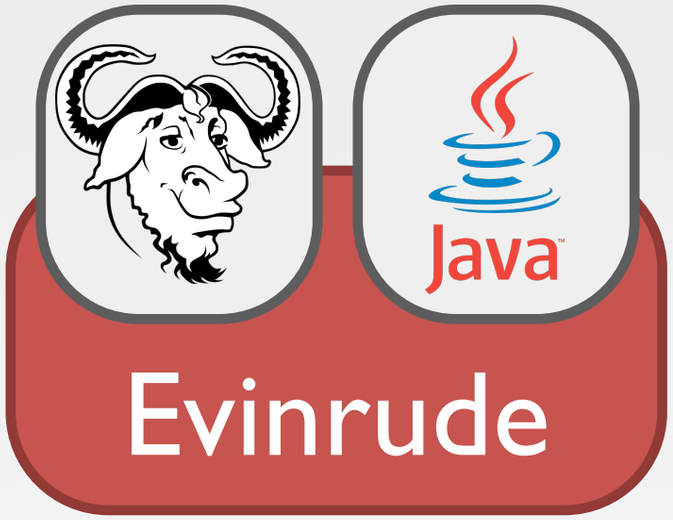
Implementation: Evinrude

- Needs GCC
- Written in Java

- Can be plugged to Coloane

- GUI for CPN-AMI platform
- View / Edit / Save models

- Provide flexibility API
- Multi-platform
- Well known by engineers



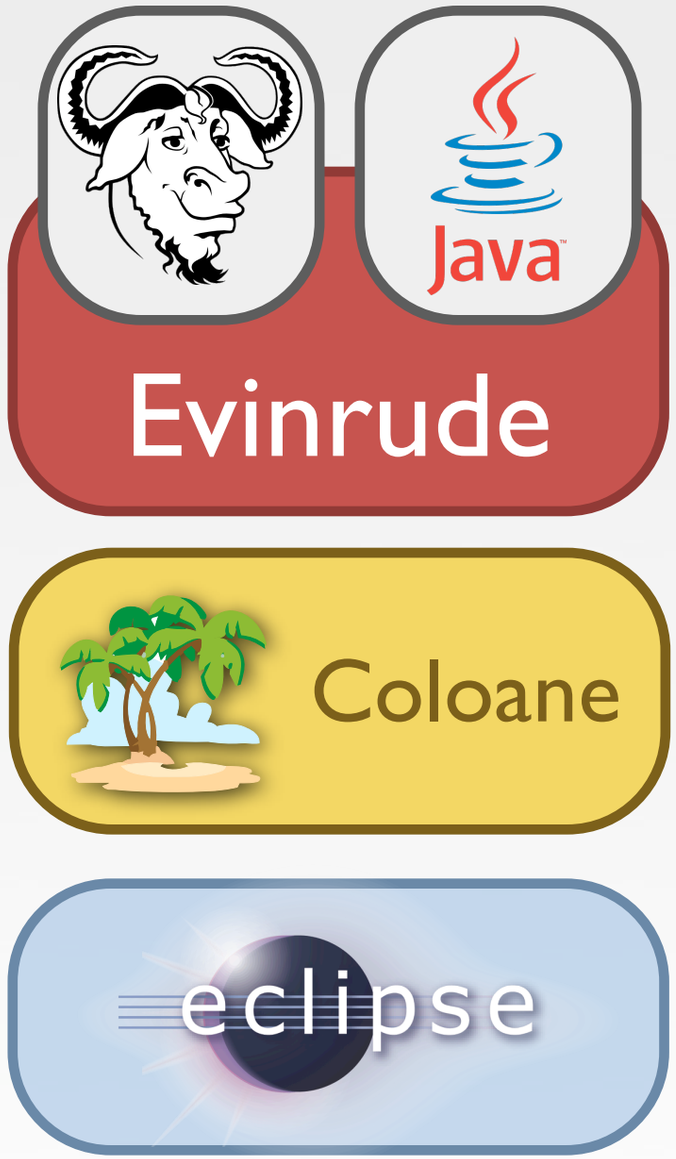
Implementation: Evinrude

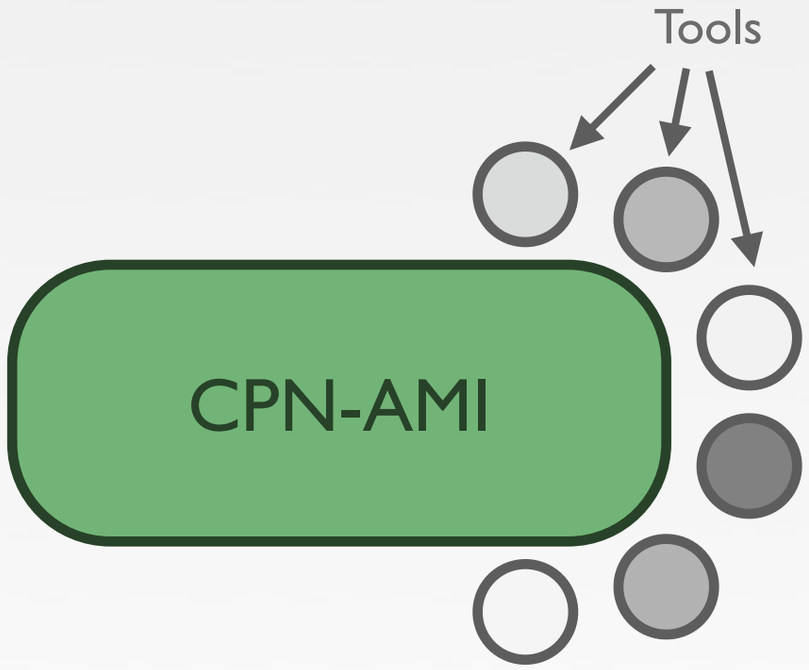
- Needs GCC
- Written in Java

- Can be plugged to Coloane

- GUI for CPN-AMI platform
- View / Edit / Save models

- Provide flexibility API
- Multi-platform
- Well known by engineers





- Check Petri nets syntax
- Allow offline analysis

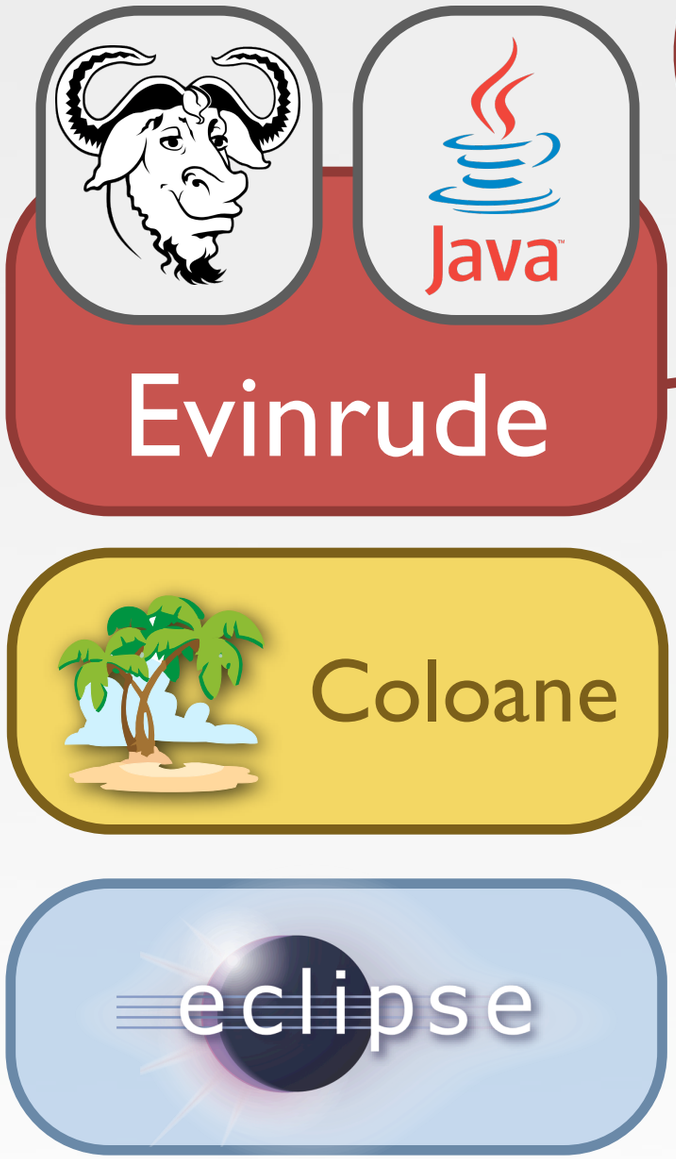
Implementation: Evinrude

- Needs GCC
- Written in Java

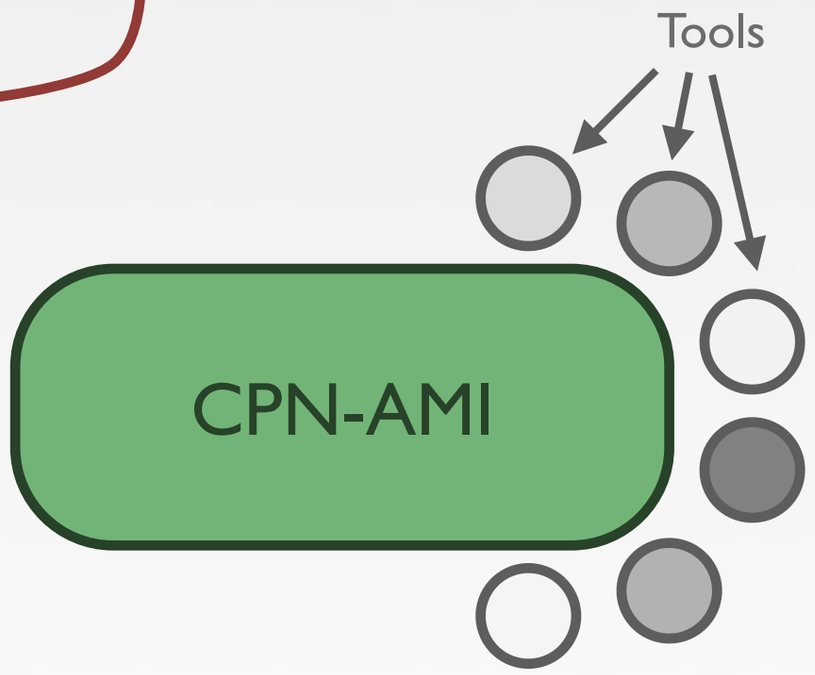
- Can be plugged to Coloane

- GUI for CPN-AMI platform
- View / Edit / Save models

- Provide flexibility API
- Multi-platform
- Well known by engineers



- Dynamic Library
- LD_PRELOAD
- Export before run



- Check Petri nets syntax
- Allow offline analysis

Conclusion

- Automatic construction of a program dedicated IDS
- Select subset of information: Perspective mechanism
- Use the same model for both offline & online analysis
  - WYCIWYC : What You've Checked is What You'll Check
- Evinrude has already produced models for:

	Before reduction	*After reduction*
GZip	842 / 1119 / 2406	149 / 165 / 498
Wu-FTPD	4132 / 5331 / 11754	829 / 963 / 3018
LigHTTPd	3403 / 4264 / 8399	673 / 761 / 2392

places / transitions / arcs

Questions ?

- Thank you for your attention...