A Layered Architecture for Detecting Malicious Behaviors

Lorenzo Martignoni[1] Elizabeth Stinson[2]
Matt Fredrikson[3] Somesh Jha[3] John Mitchell[2]

[1]Università degli Studi di Milano

[2]Stanford University

[3]University of Wisconsin–Madison

RAID 2008

The arm-race of malware and malware detectors

- Even the most effective malware detectors fail to detect more than 30% of malware seen in the wild
- More than five million active, distinct, bot-infected machines (Symantec report about the first half of 2007)

The arm-race of malware and malware detectors

- Even the most effective malware detectors fail to detect more than 30% of malware seen in the wild
- More than five million active, distinct, bot-infected machines (Symantec report about the first half of 2007)

- Traditional malware detectors are based on syntactic signatures
- Malware producers can easily generate malware variants capable of evading existing signatures (e.g., 40K variants of Storm in 2 weeks)

The arm-race of malware and malware detectors

- Even the most effective malware detectors fail to detect more than 30% of malware seen in the wild
- More than five million active, distinct, bot-infected machines (Symantec report about the first half of 2007)

- Traditional malware detectors are based on syntactic signatures
- Malware producers can easily generate malware variants capable of evading existing signatures (e.g., 40K variants of Storm in 2 weeks)

Malware detectors have a finite set of syntactic signatures, but malicious programs have infinitely mutable syntax

Behavior-based malware detection

- Detect high-level actions that financially motivate malware development & distribution
  - keystroke logging
  - data leaking
  - proxying

Behavior-based malware detection

- Detect high-level actions that financially motivate malware development & distribution
  - keystroke logging
  - data leaking
  - proxying

- Build models of the high-level actions that include only the essential components of each action
- Monitor the execution of suspicious programs and compare the observed behaviors with the models describing malicious actions

Behavior-based malware detection

- Detect high-level actions that financially motivate malware development & distribution
  - keystroke logging
  - data leaking
  - proxying

- Build models of the high-level actions that include only the essential components of each action
- Monitor the execution of suspicious programs and compare the observed behaviors with the models describing malicious actions

There are a finite number of ways to achieve each action
The models can encode all such ways

Semantic gap between models and monitored events

- Monitor execution of the program using an emulator (or similar)
- Lowest level events in behavior specifications are system calls
- Malicious behaviors are described as sequences of essential actions

Semantic gap between models and monitored events

- Monitor execution of the program using an emulator (or similar)
- Lowest level events in behavior specifications are system calls
- Malicious behaviors are described as sequences of essential actions

What we see

NtDeviceIo... NtOpenFile NtCreateSe... NtMapView ...

Semantic gap between models and monitored events

- Monitor execution of the program using an emulator (or similar)
- Lowest level events in behavior specifications are system calls
- Malicious behaviors are described as sequences of essential actions

What we see

NtDeviceIo... NtOpenFile NtCreateSe... NtMapView ...

is different from the essential actions we need to identify

download a file and execute it

Semantic gap between models and monitored events

- Monitor execution of the program using an emulator (or similar)
- Lowest level events in behavior specifications are system calls
- Malicious behaviors are described as sequences of essential actions

What we see

NtDeviceIo. . . NtOpenFile NtCreateSe. . . NtMapView . . .

is different from the essential actions we need to identify

download a file and execute it

There is a semantic gap between the stream of events generated by a program and the actions of behavior specifications [Chen & Noble '01]

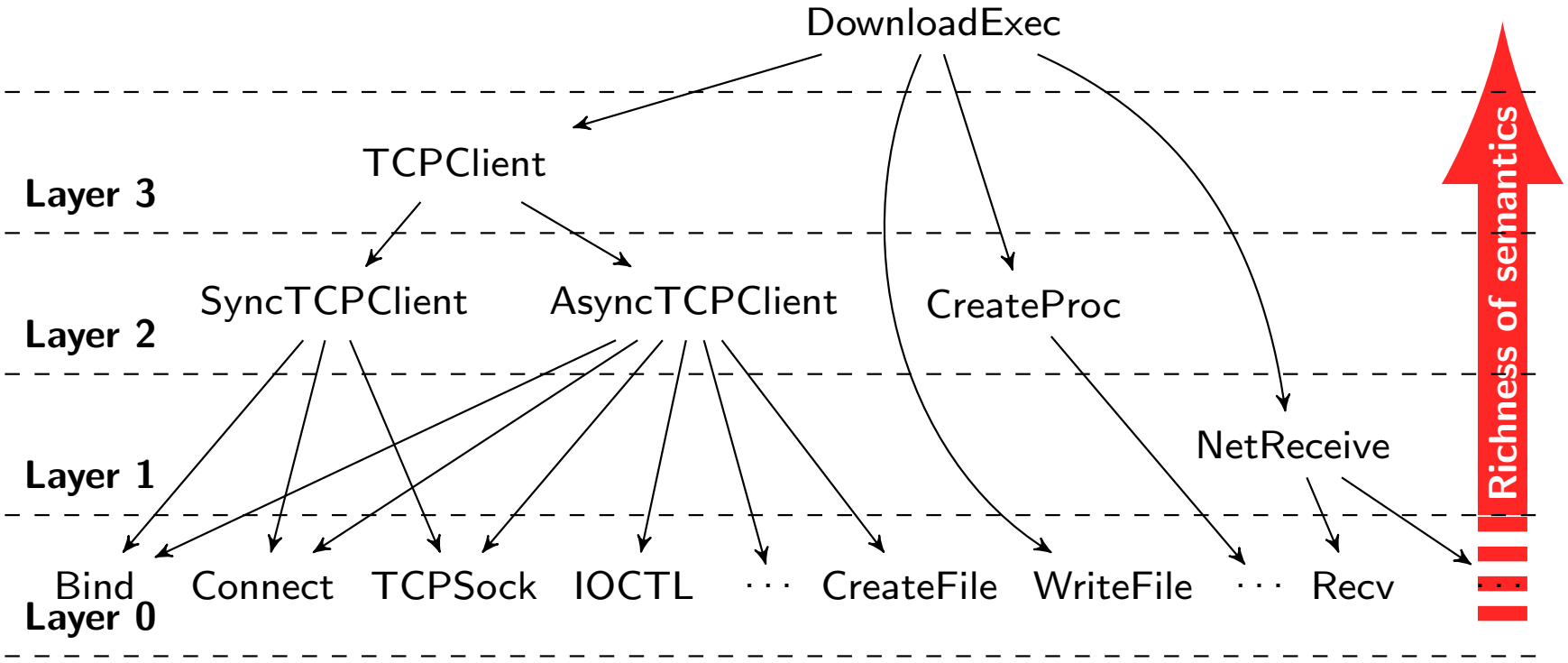
Outline

Our solution

- Complex & high-level behaviors are decomposed into multiple layers
- The lowest layer represents system call invocations
- Upper layers have a richer semantics

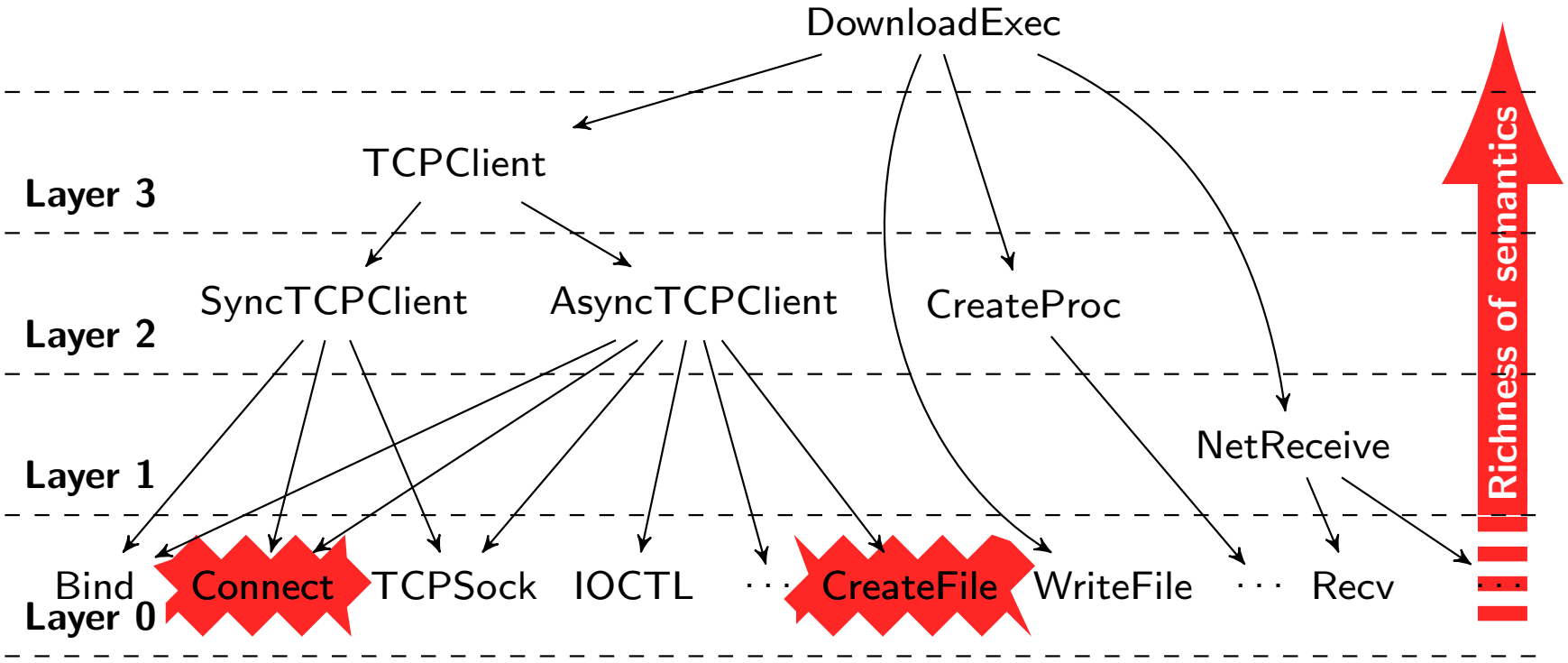
Our solution

- Complex & high-level behaviors are decomposed into multiple layers
- The lowest layer represents system call invocations
- Upper layers have a richer semantics



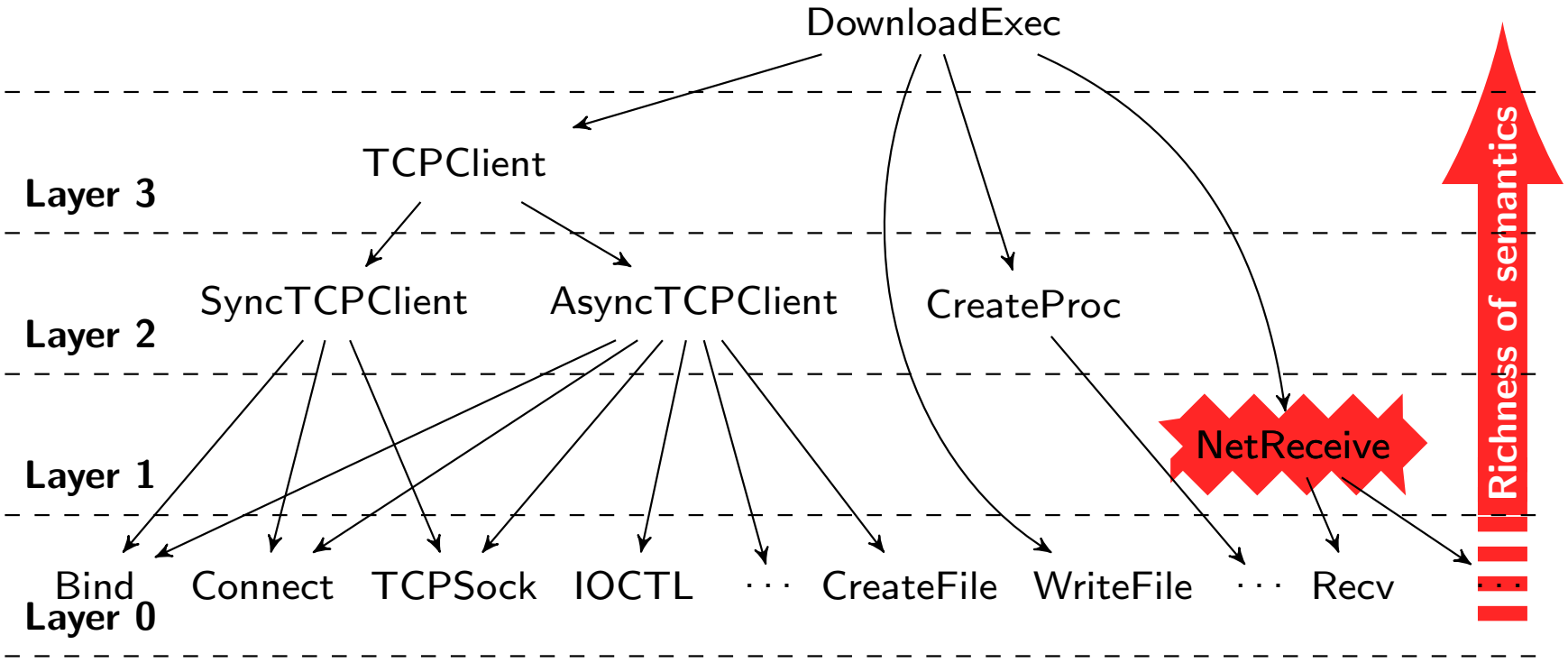
Our solution

- Complex & high-level behaviors are decomposed into multiple layers
- The lowest layer represents system call invocations
- Upper layers have a richer semantics



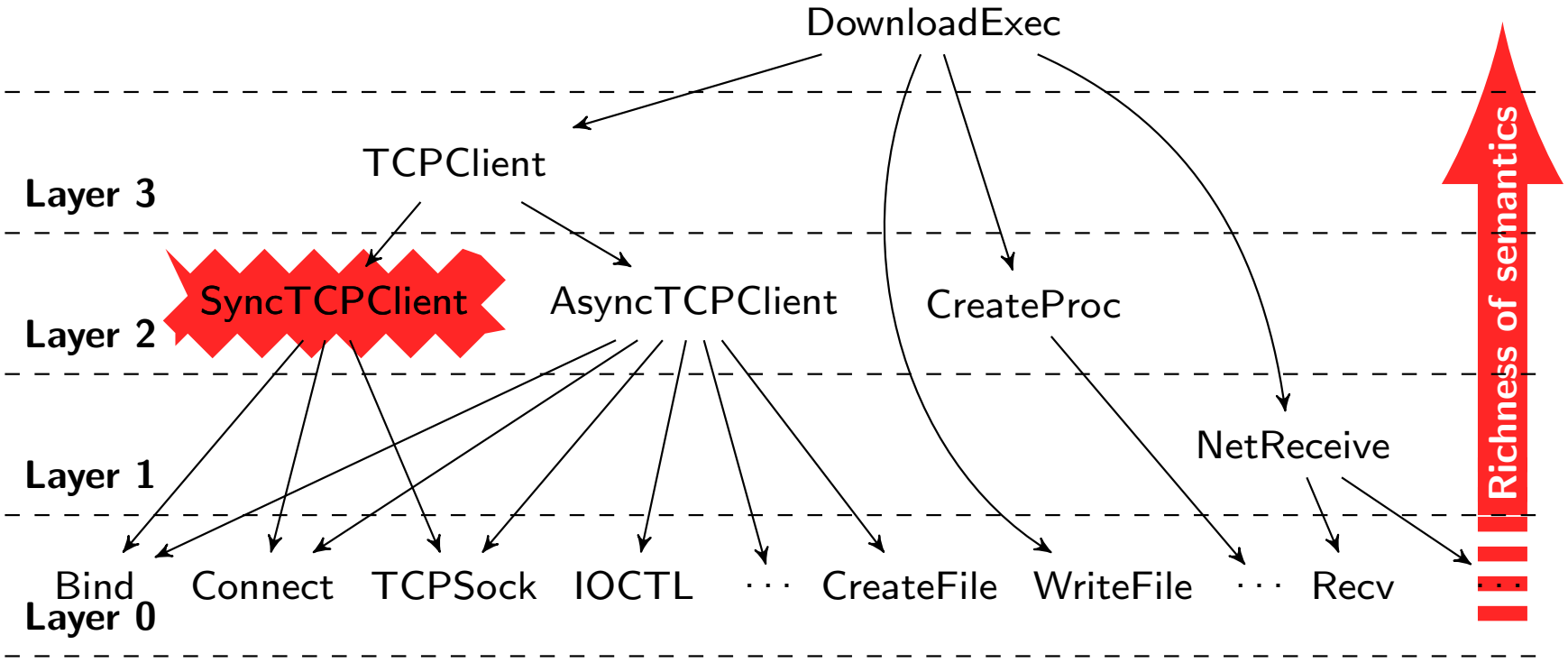
Our solution

- Complex & high-level behaviors are decomposed into multiple layers
- The lowest layer represents system call invocations
- Upper layers have a richer semantics



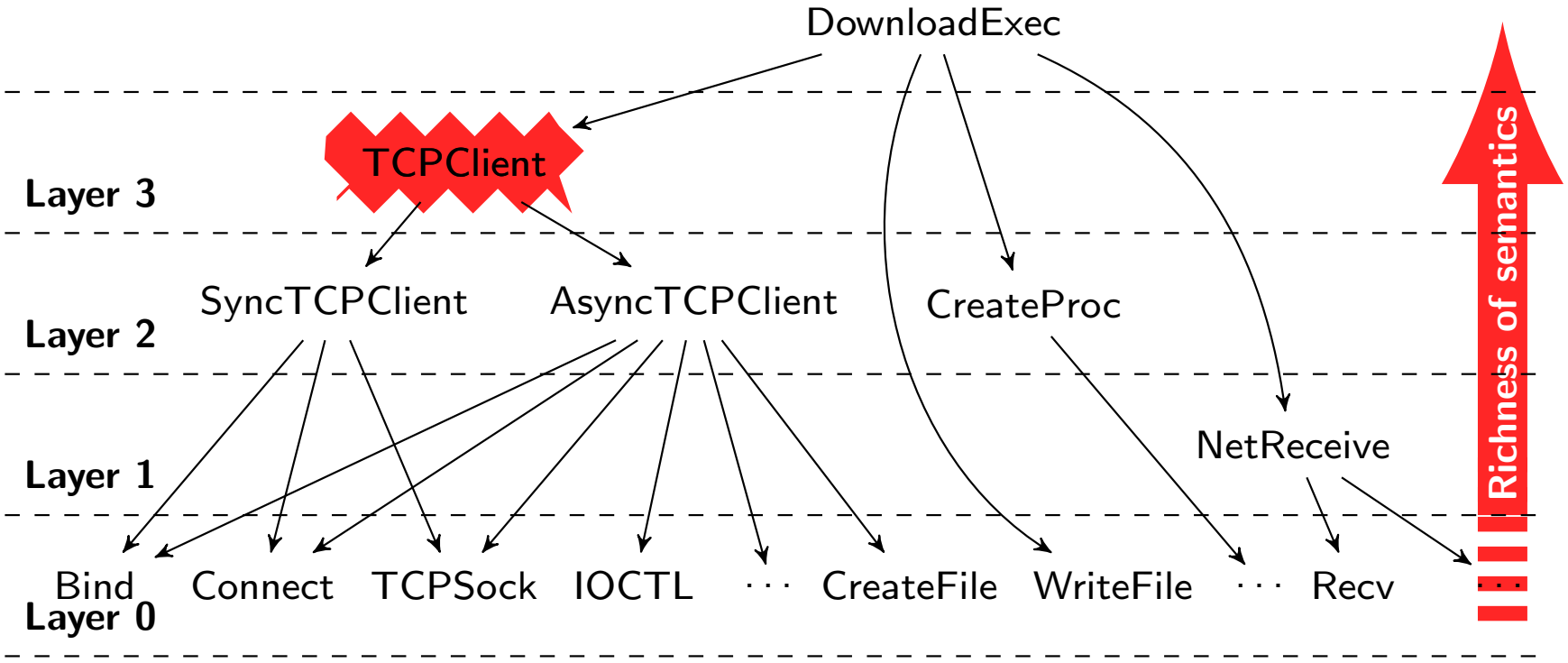
Our solution

- Complex & high-level behaviors are decomposed into multiple layers
- The lowest layer represents system call invocations
- Upper layers have a richer semantics



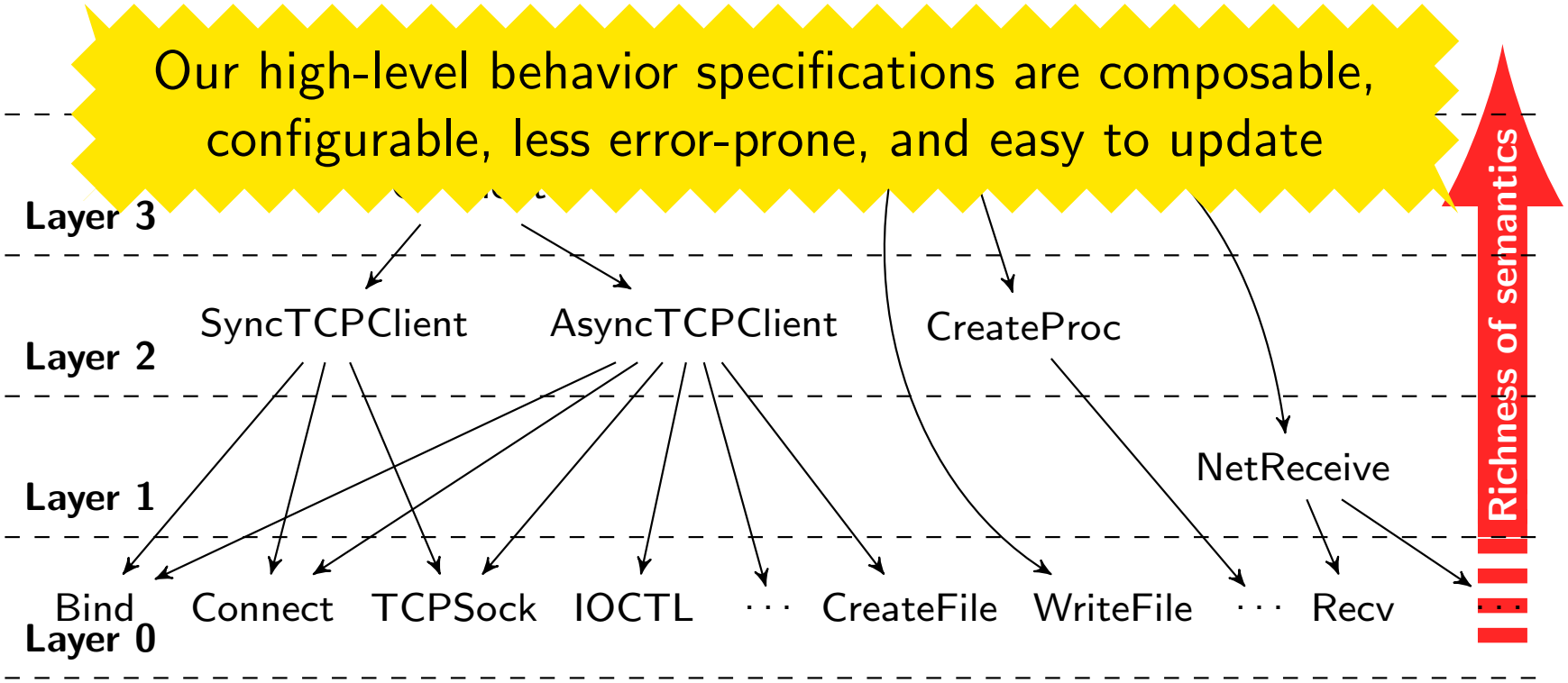
Our solution

- Complex & high-level behaviors are decomposed into multiple layers
- The lowest layer represents system call invocations
- Upper layers have a richer semantics



Our solution

- Complex & high-level behaviors are decomposed into multiple layers
- The lowest layer represents system call invocations
- Upper layers have a richer semantics



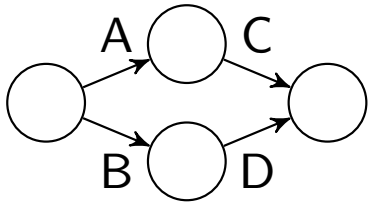
Technical challenges

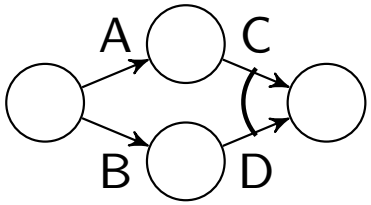
1. Find a good formalism to model complex high-level behaviors
2. Construct complete models of complex high-level behaviors

Technical challenges

1. *Find a good formalism to model complex high-level behaviors*

Behavior graphs (derived from AND-OR graphs)





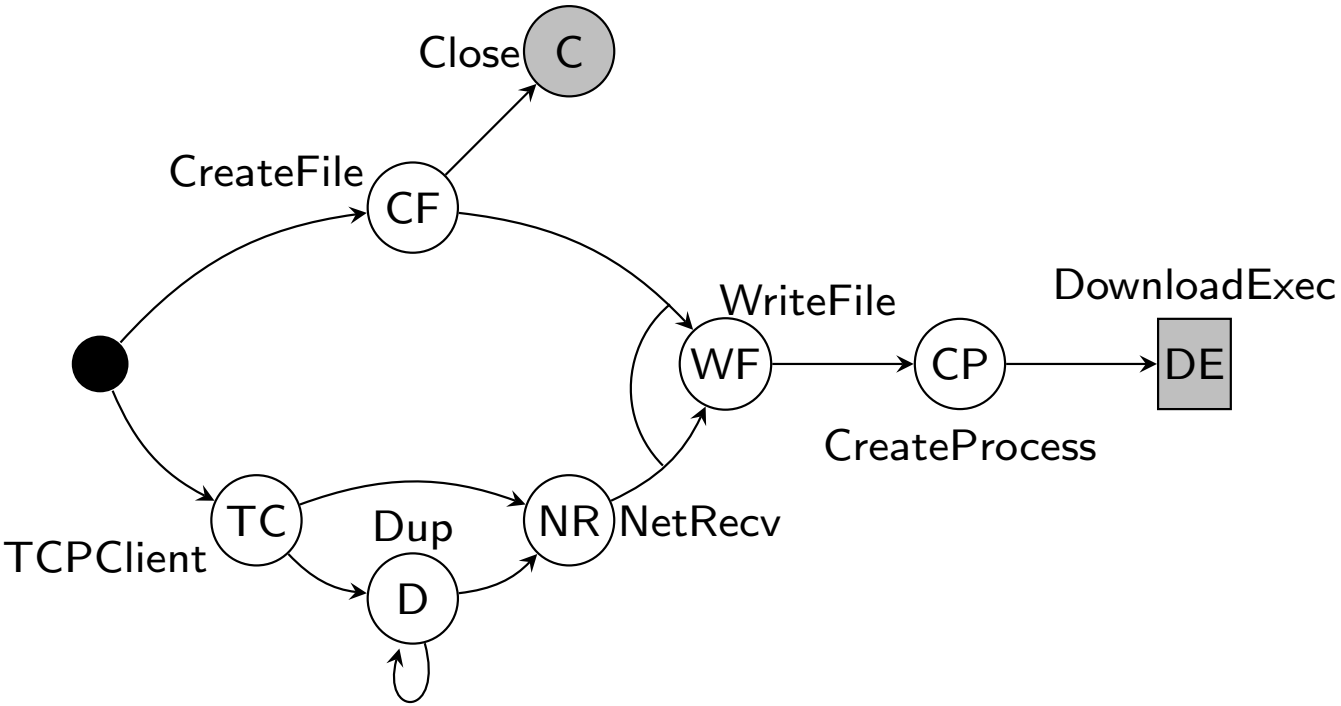
Technical challenges

2. *Construct complete models of complex high-level behaviors*

Manual analysis of execution traces & validation

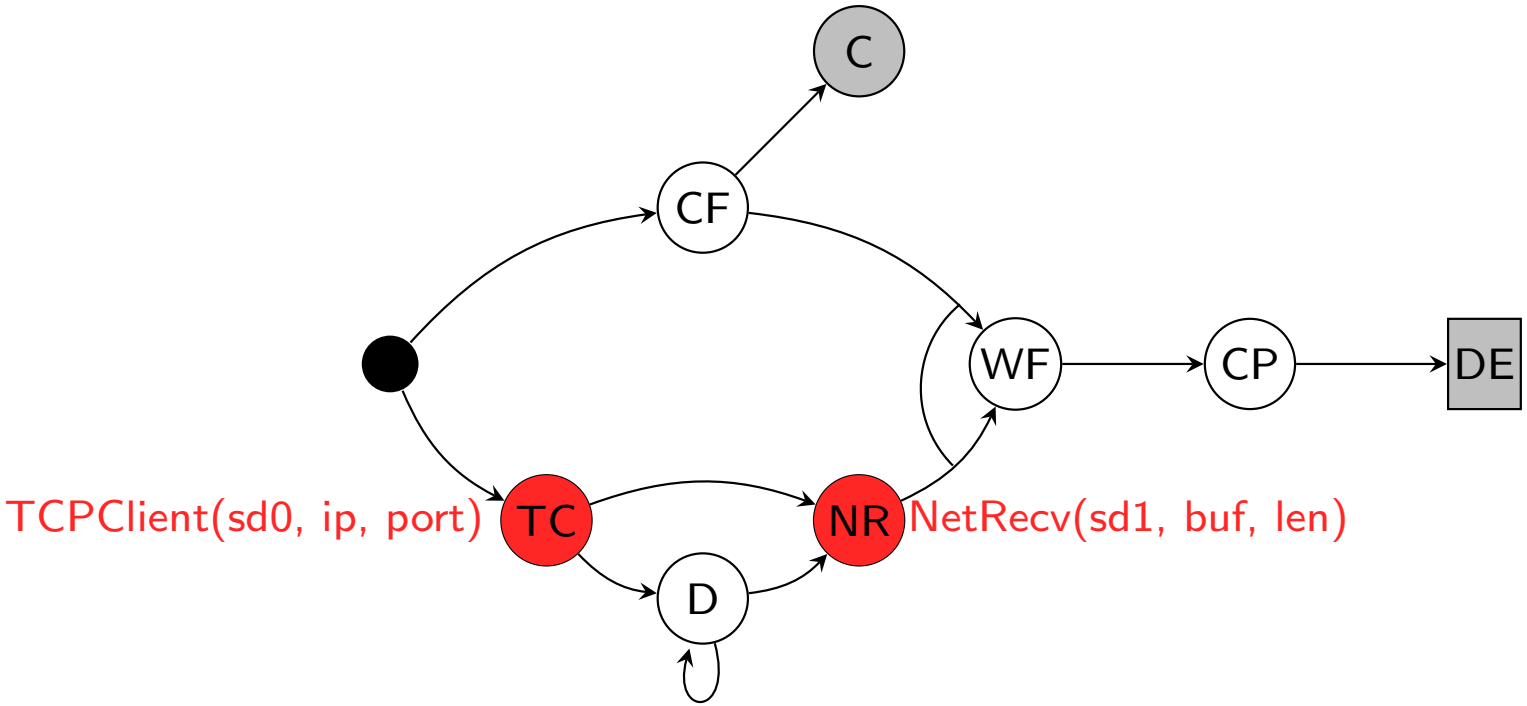
Behavior graphs

Download & execute



Behavior graphs

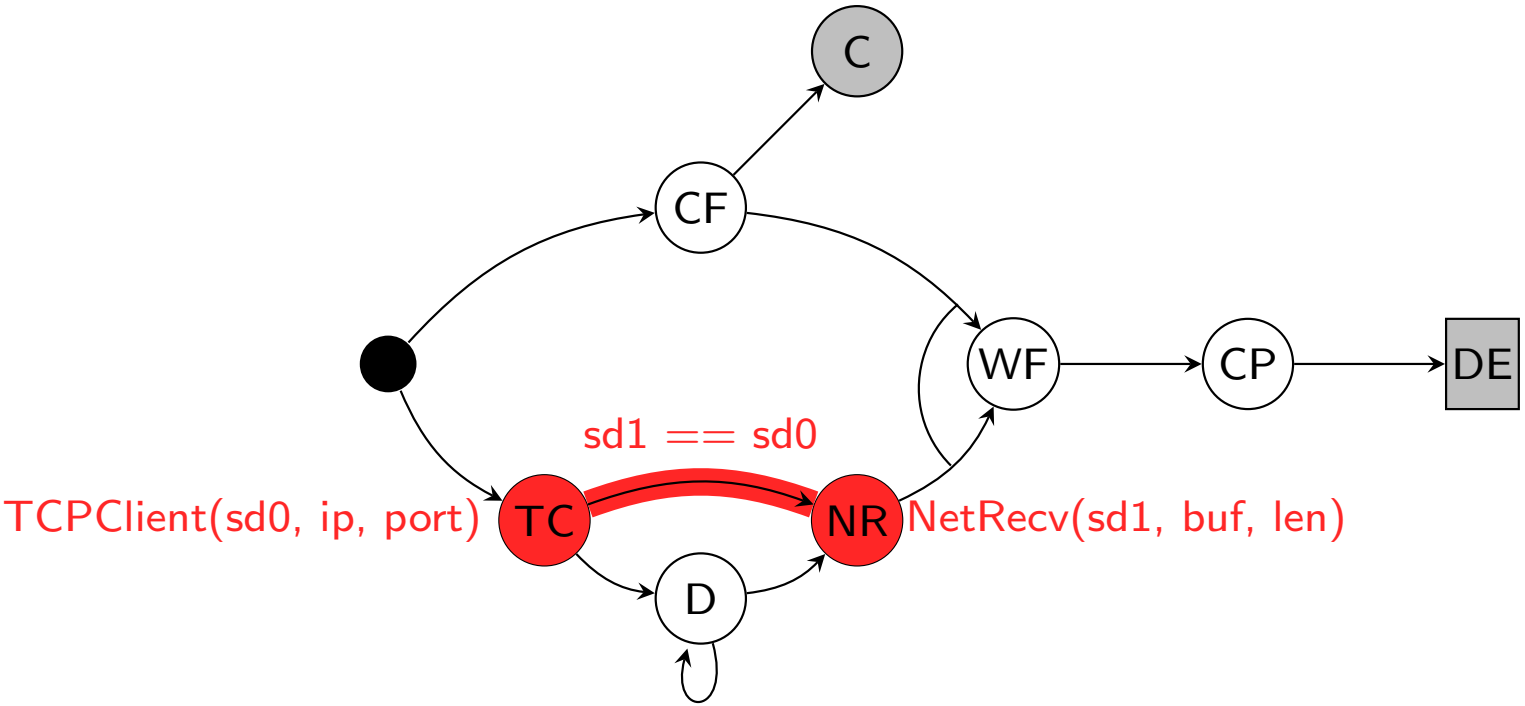
Download & execute



Internal nodes represent events (with formal parameters)

Behavior graphs

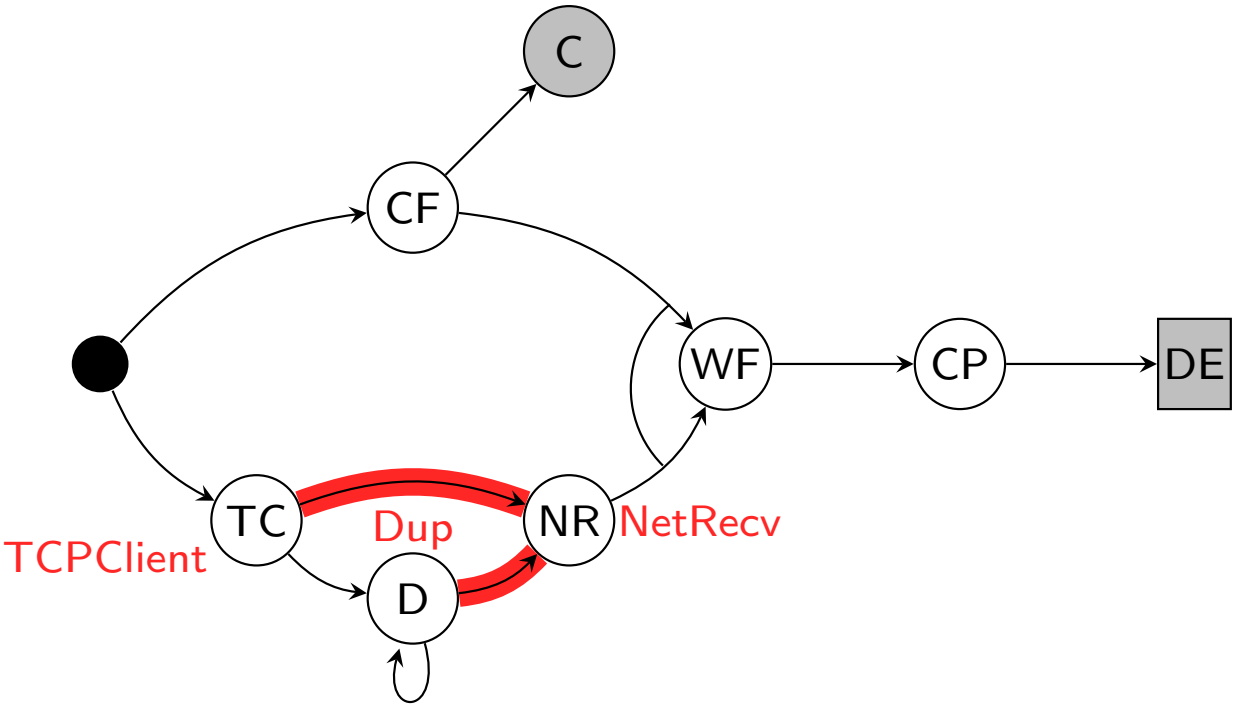
Download & execute



Edges represent predicates on events arguments

Behavior graphs

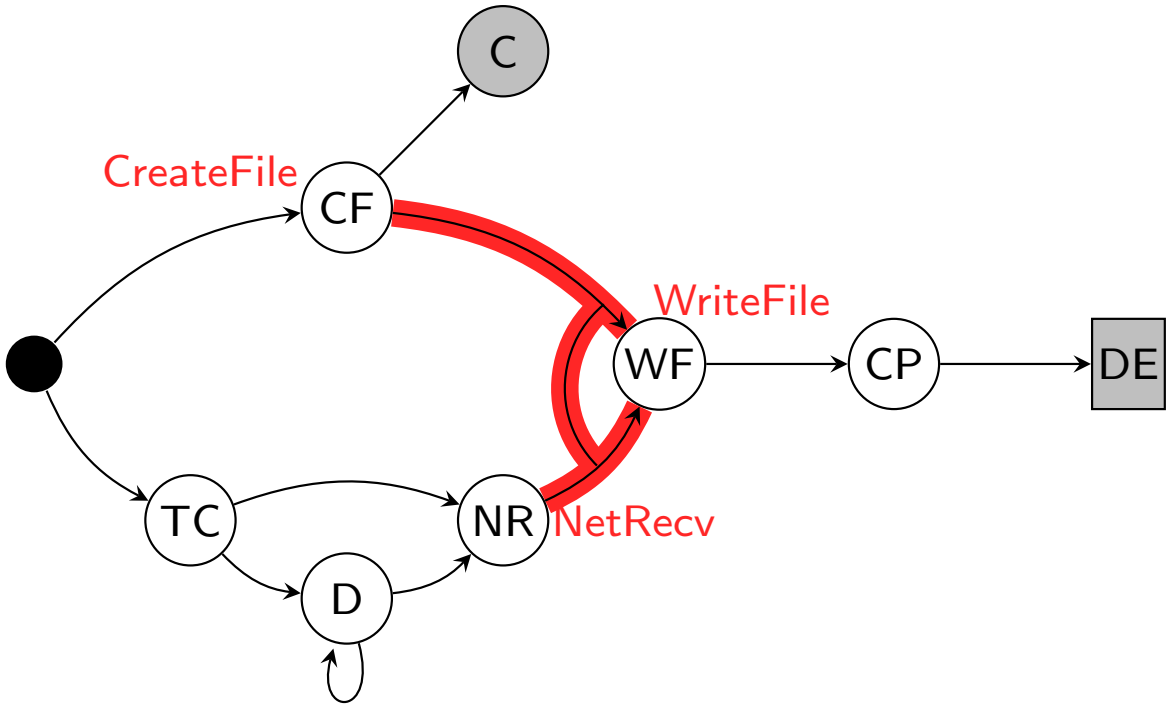
Download & execute



ORed edges represent events of which at least one has to occur

Behavior graphs

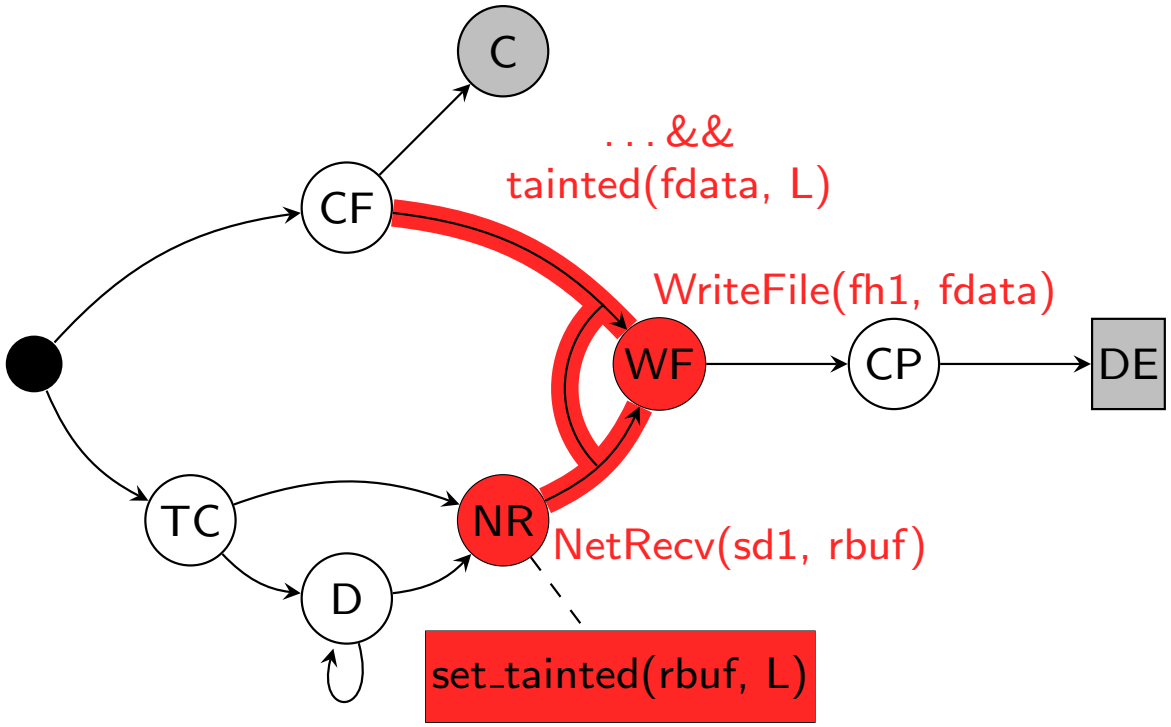
Download & execute



ANDed edges represent events that all have to occur (but can occur in any order)

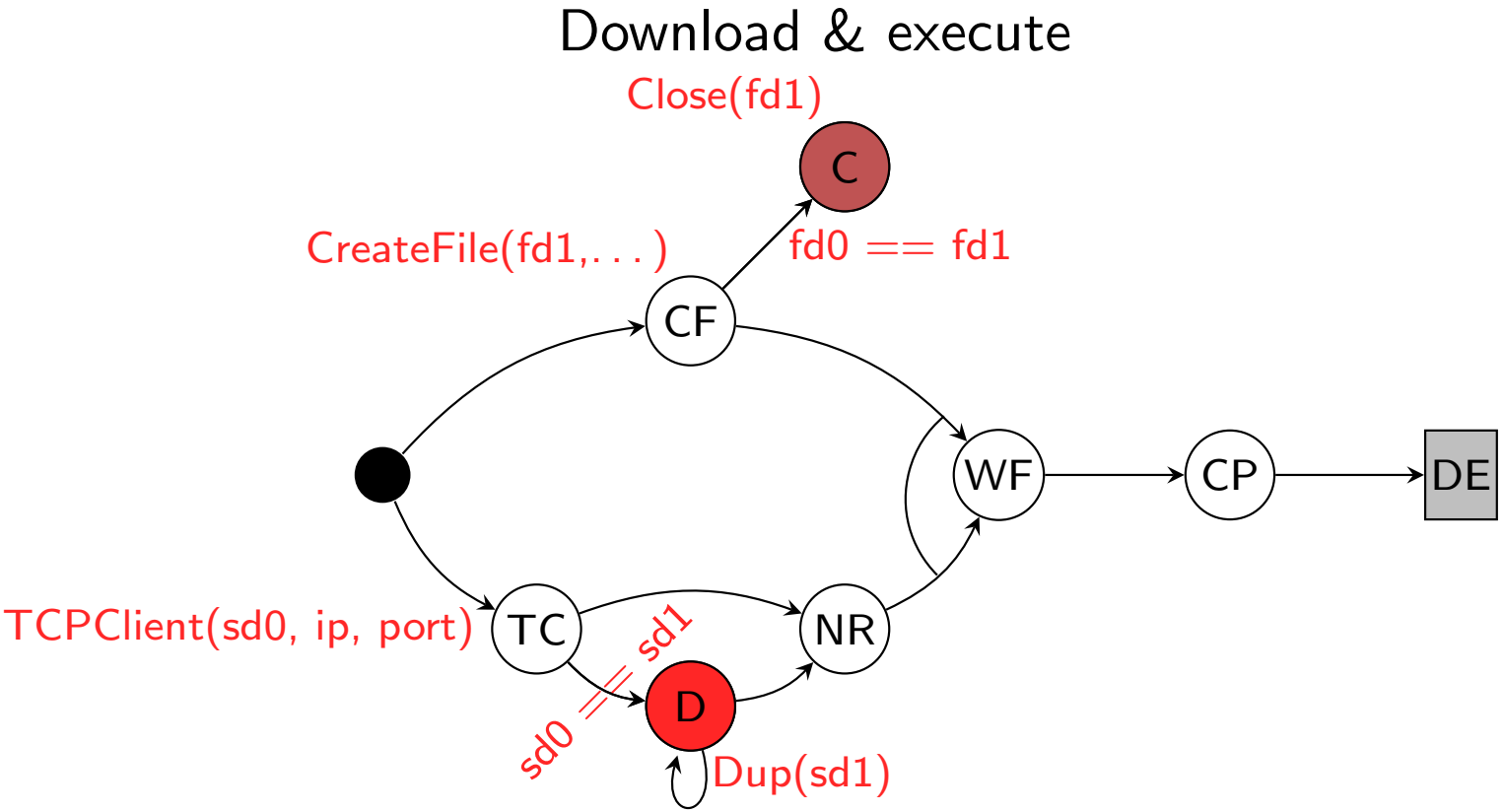
Behavior graphs

Download & execute



On-reach actions are actions performed by the monitoring system in response to reaching a given node

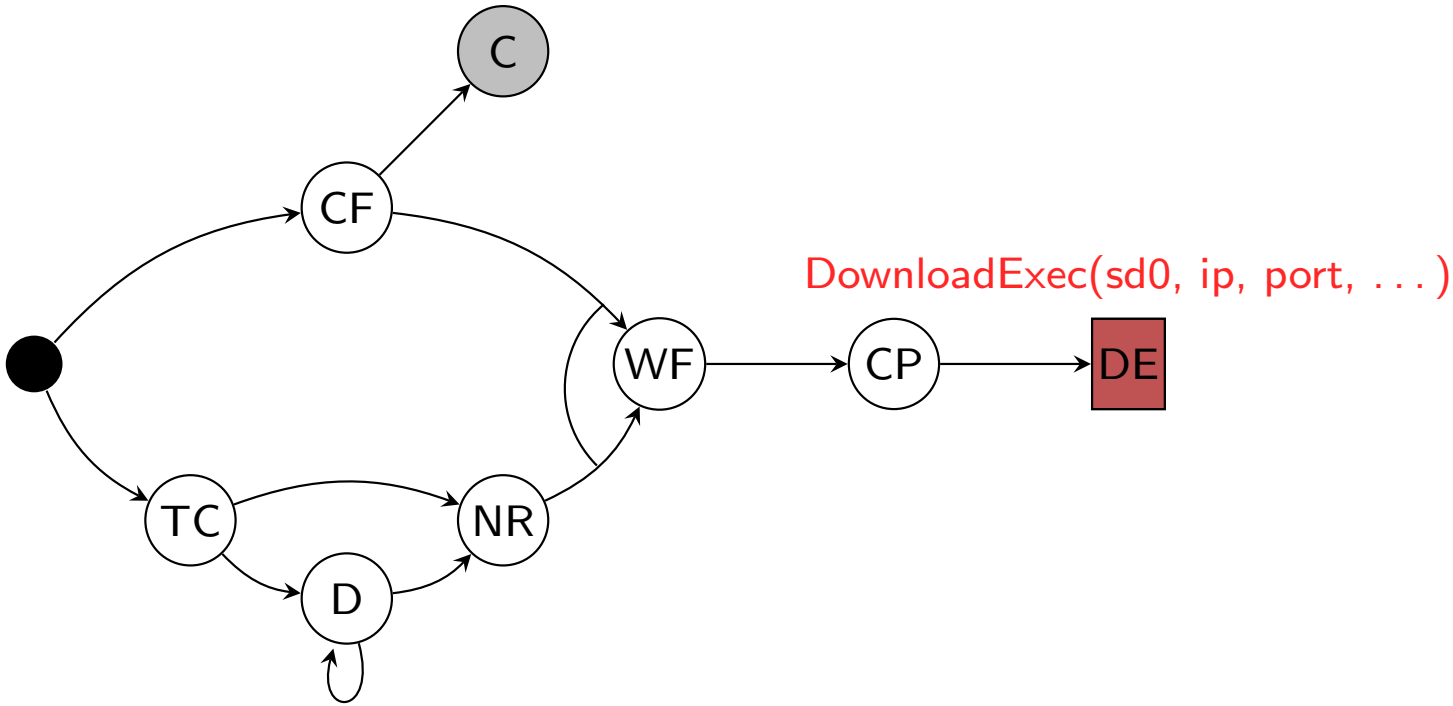
Behavior graphs



Annihilator and replicator nodes represent events that destroy and duplicate resources

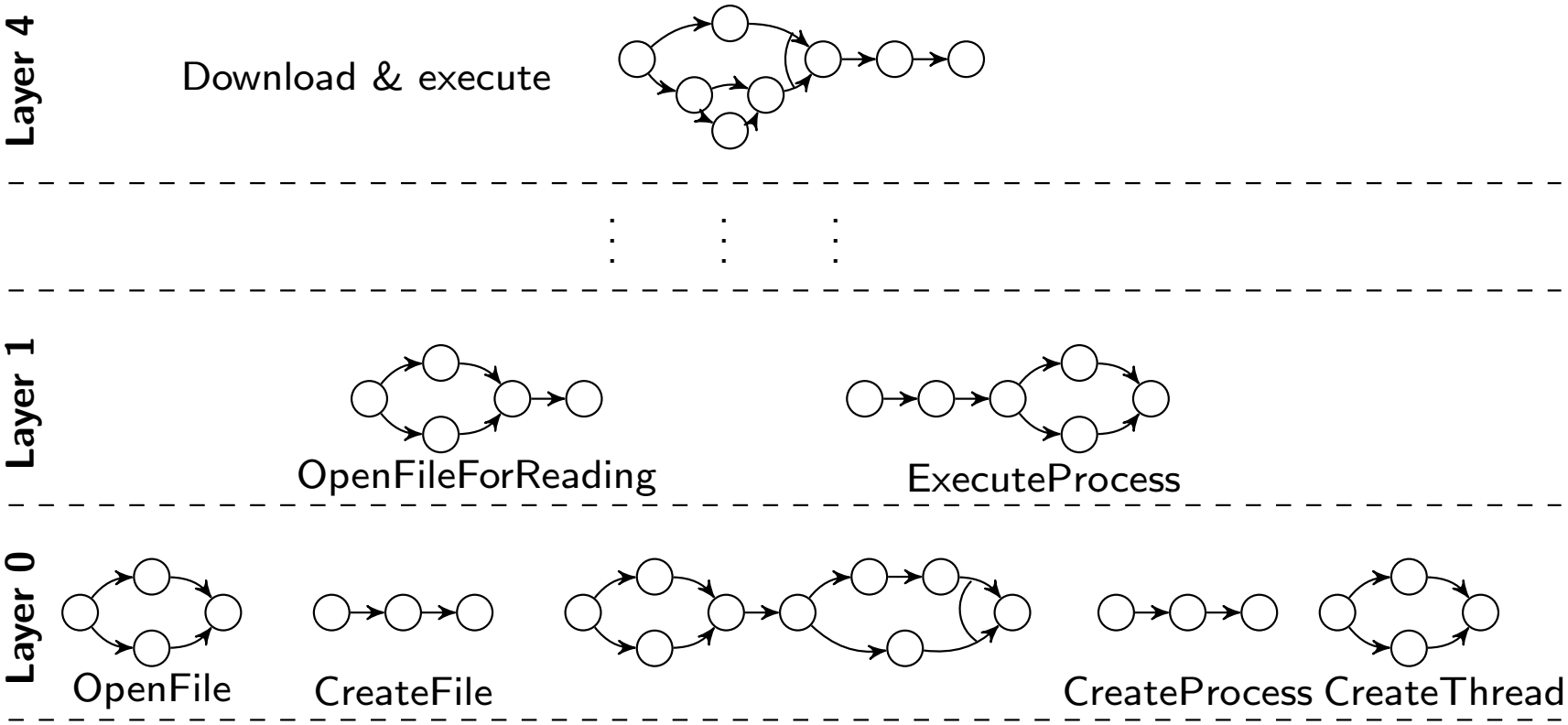
Behavior graphs

Download & execute

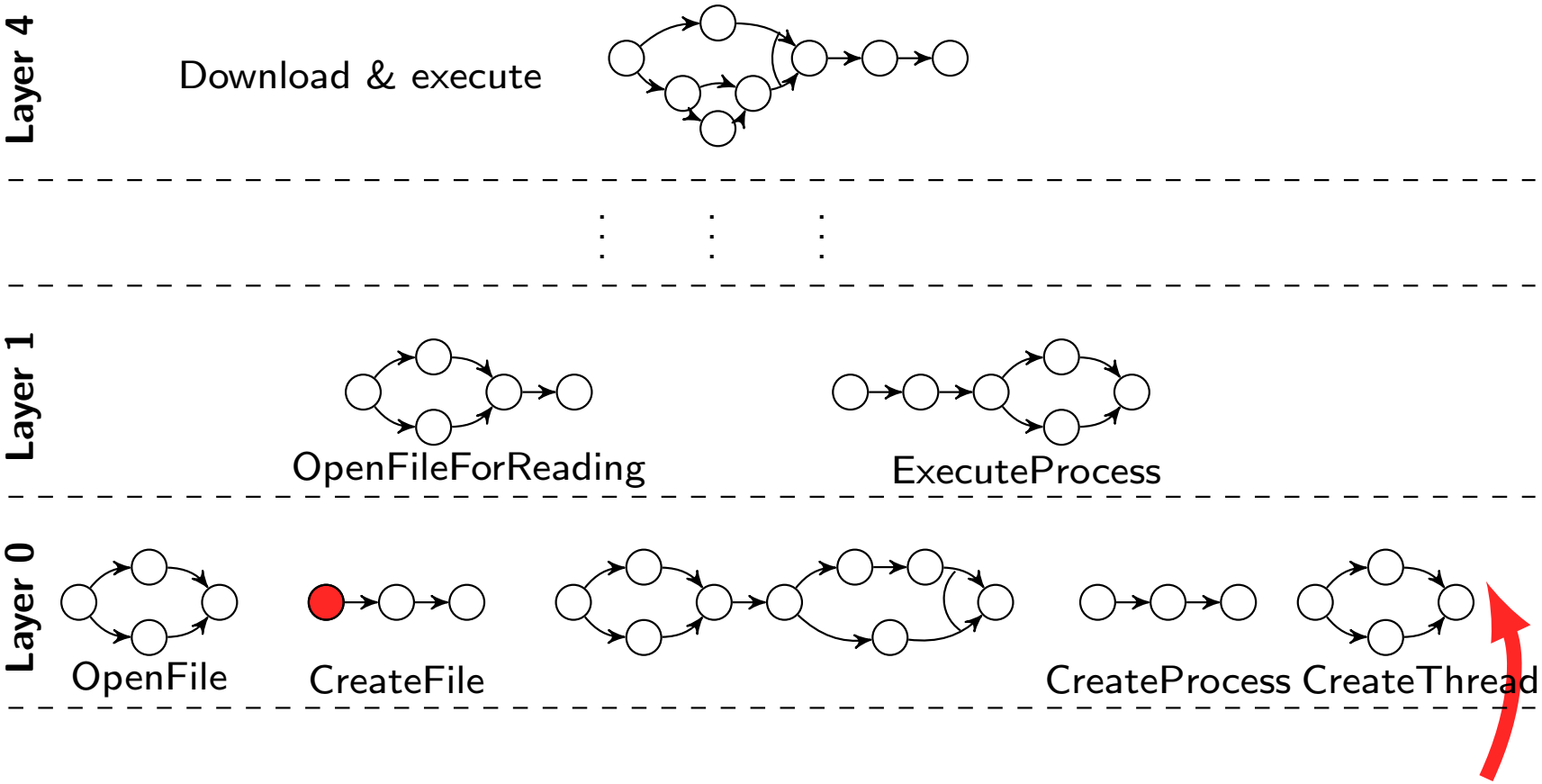


Acceptor nodes represent actions taken by our system when behaviors are matched

Matching malicious behaviors

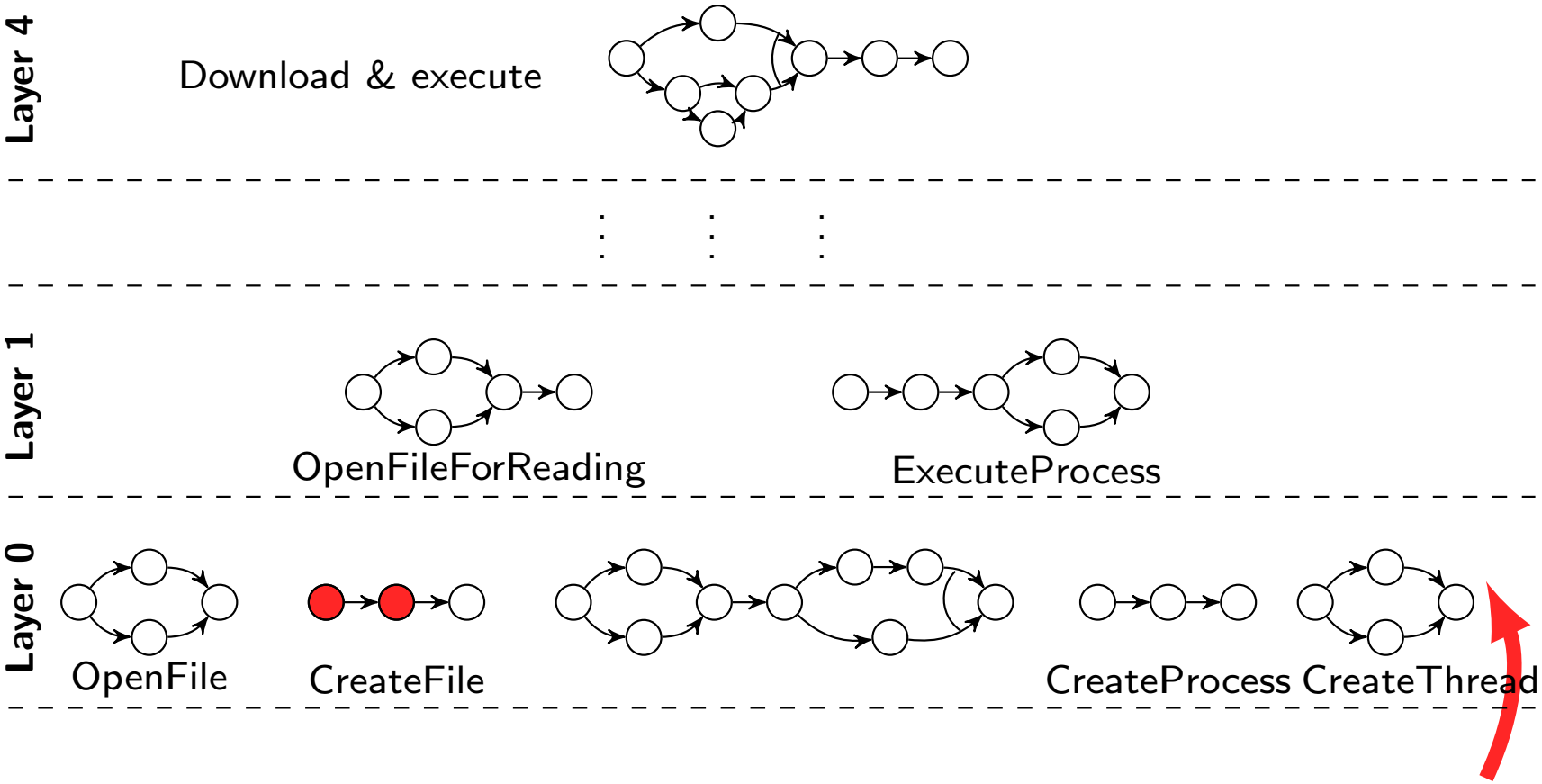


Matching malicious behaviors



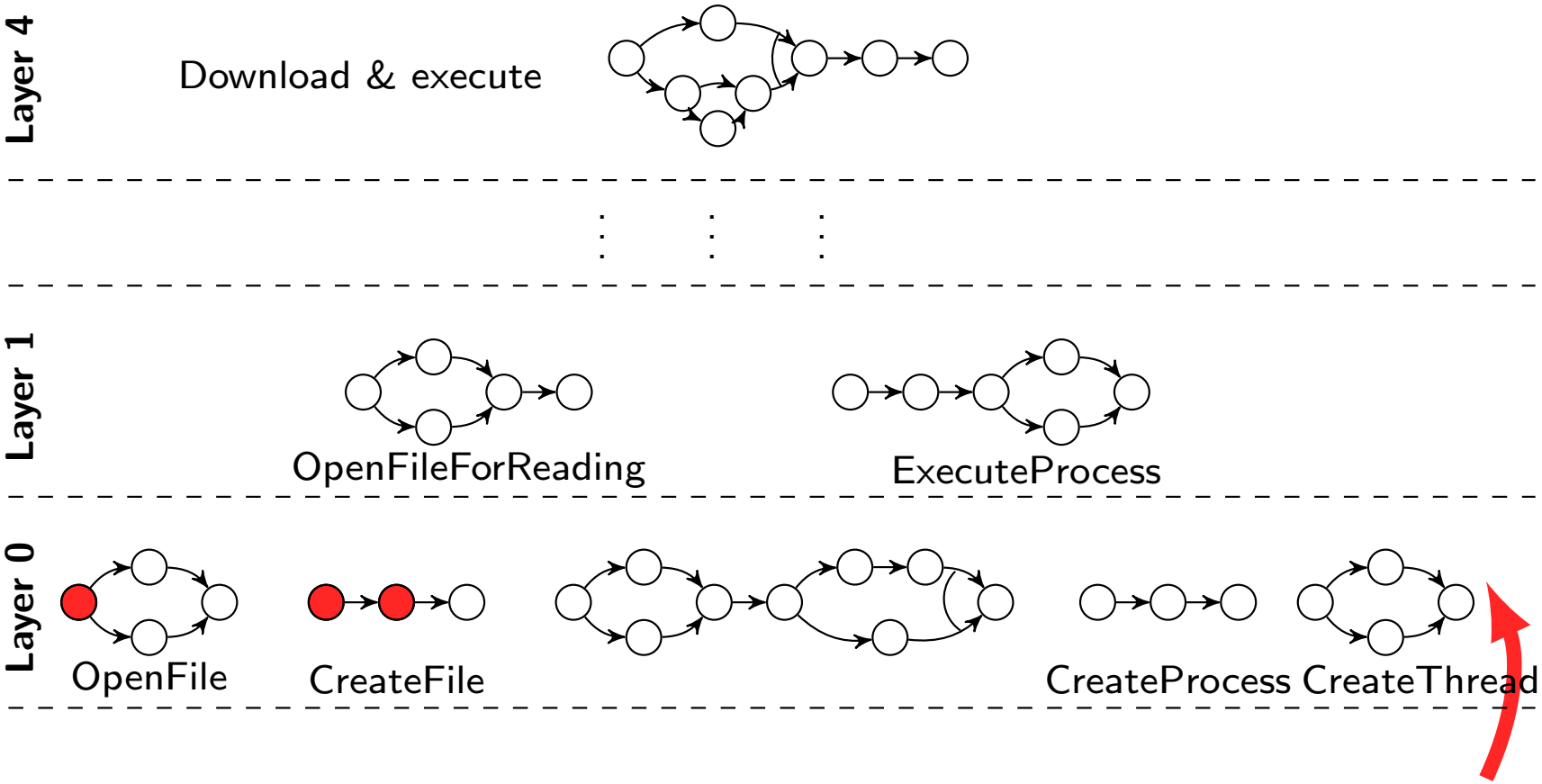
OS events are passed to the lowest layer

Matching malicious behaviors



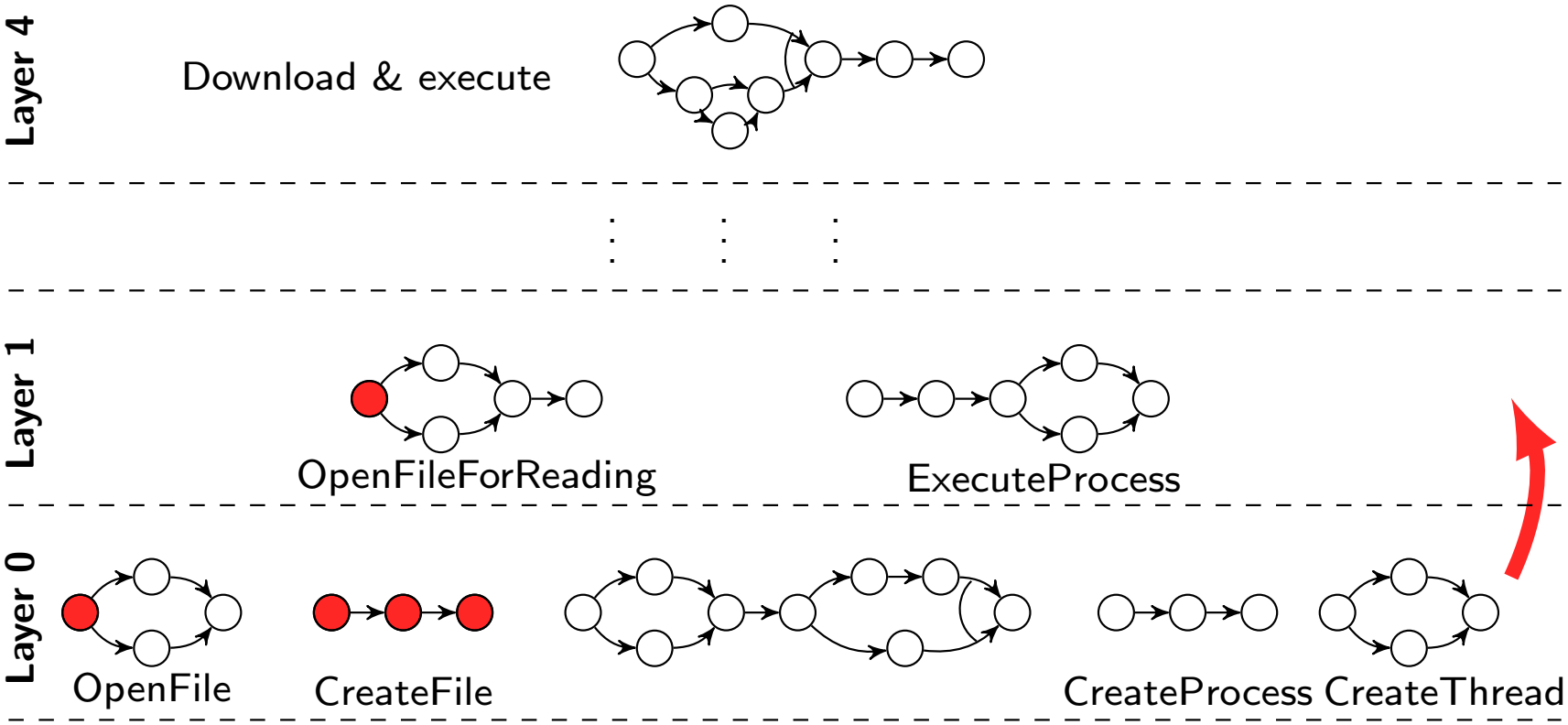
OS events are passed to the lowest layer

Matching malicious behaviors



OS events are passed to the lowest layer

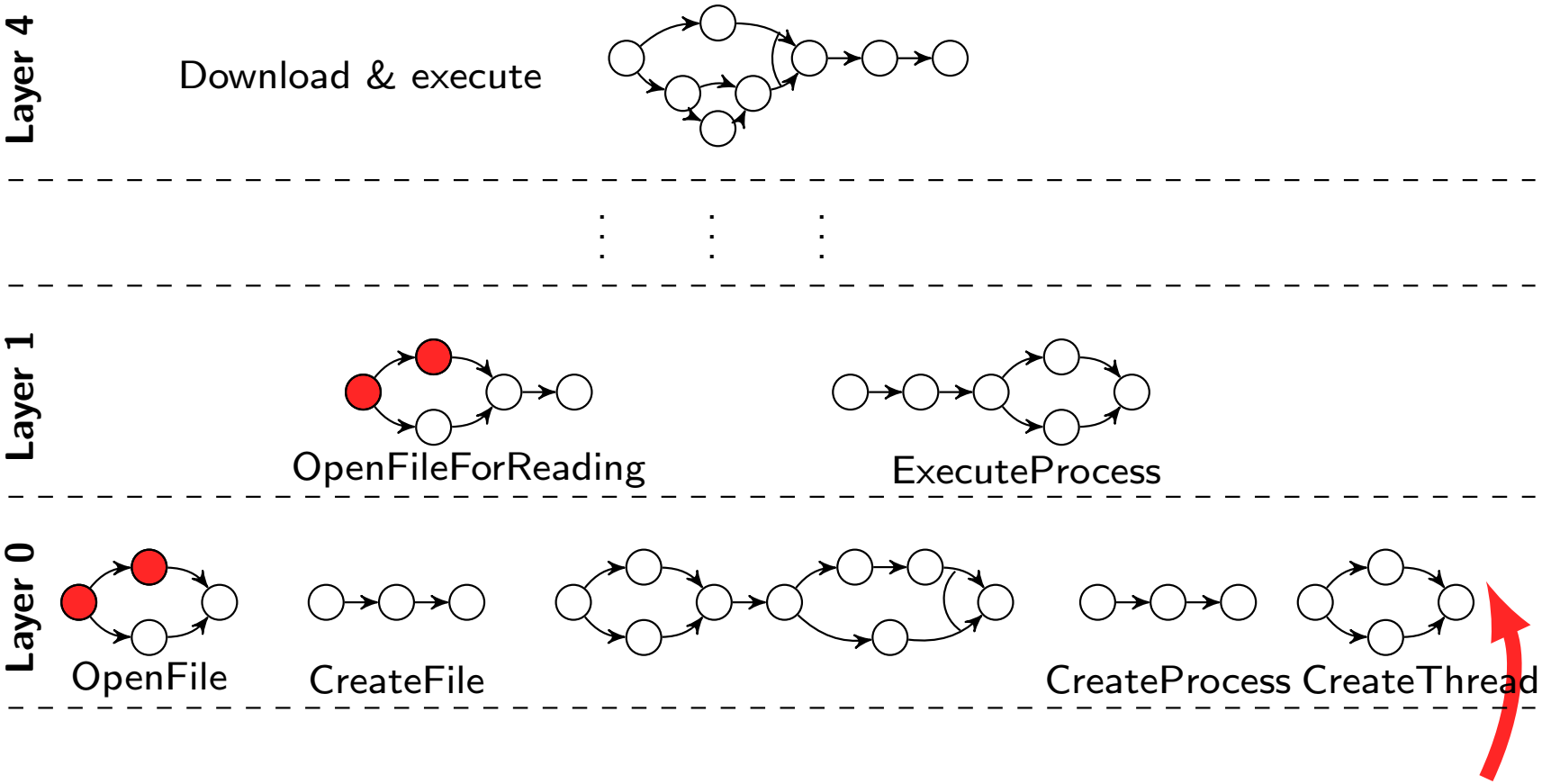
Matching malicious behaviors



The low-level behavior is matched

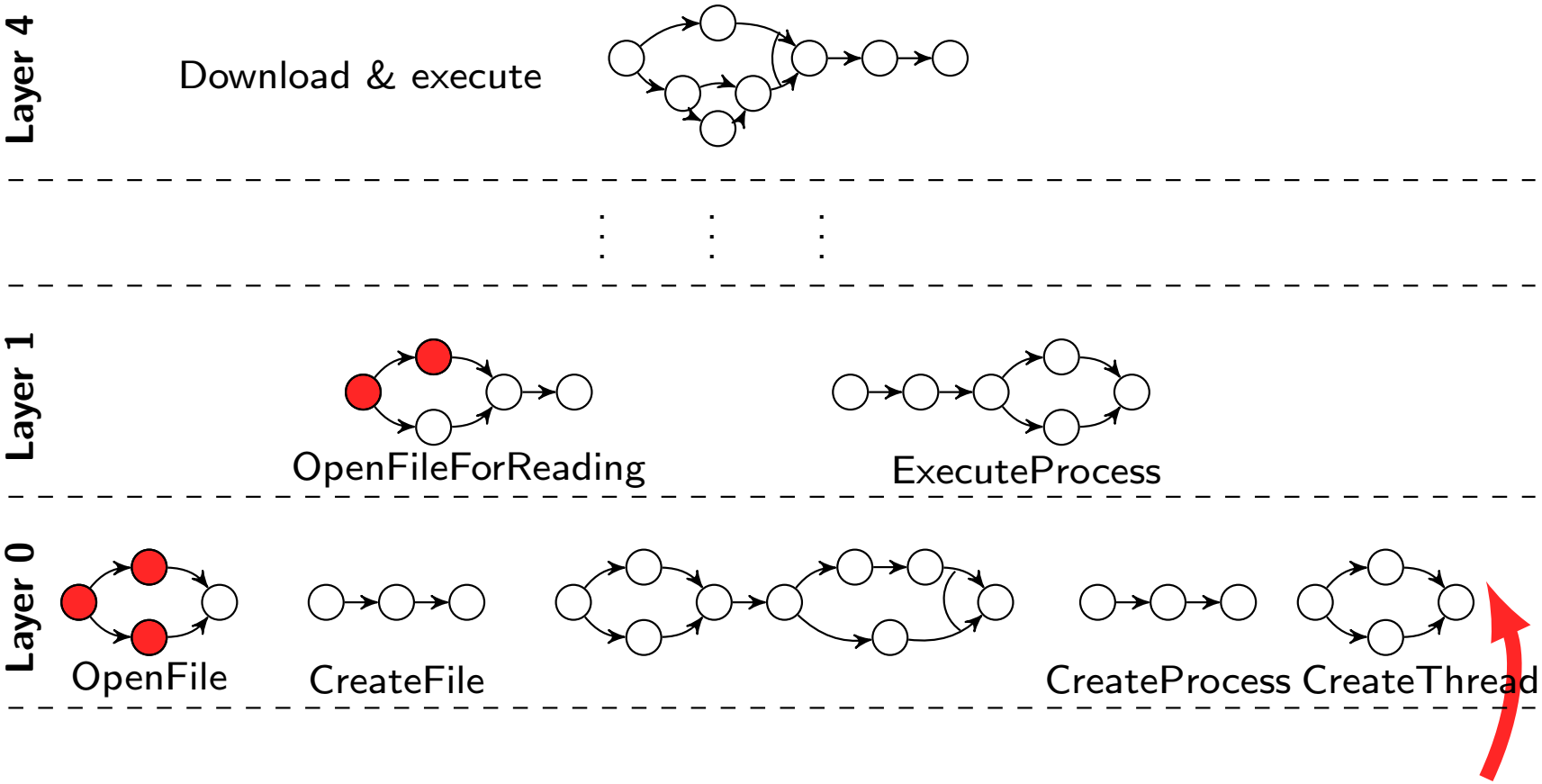
The synthetic event (acceptor node) is passed to the upper layer

Matching malicious behaviors



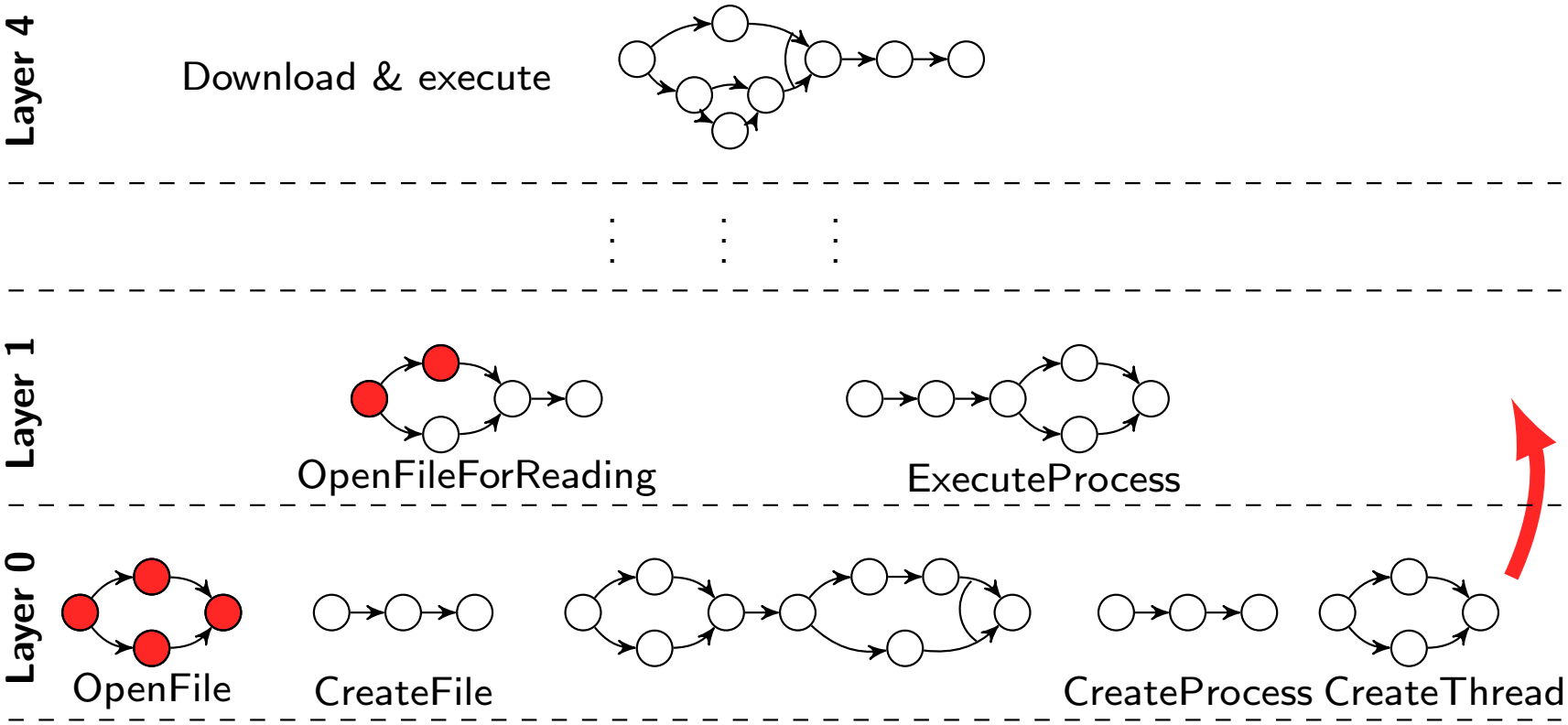
OS events are passed to the lowest layer

Matching malicious behaviors



OS events are passed to the lowest layer

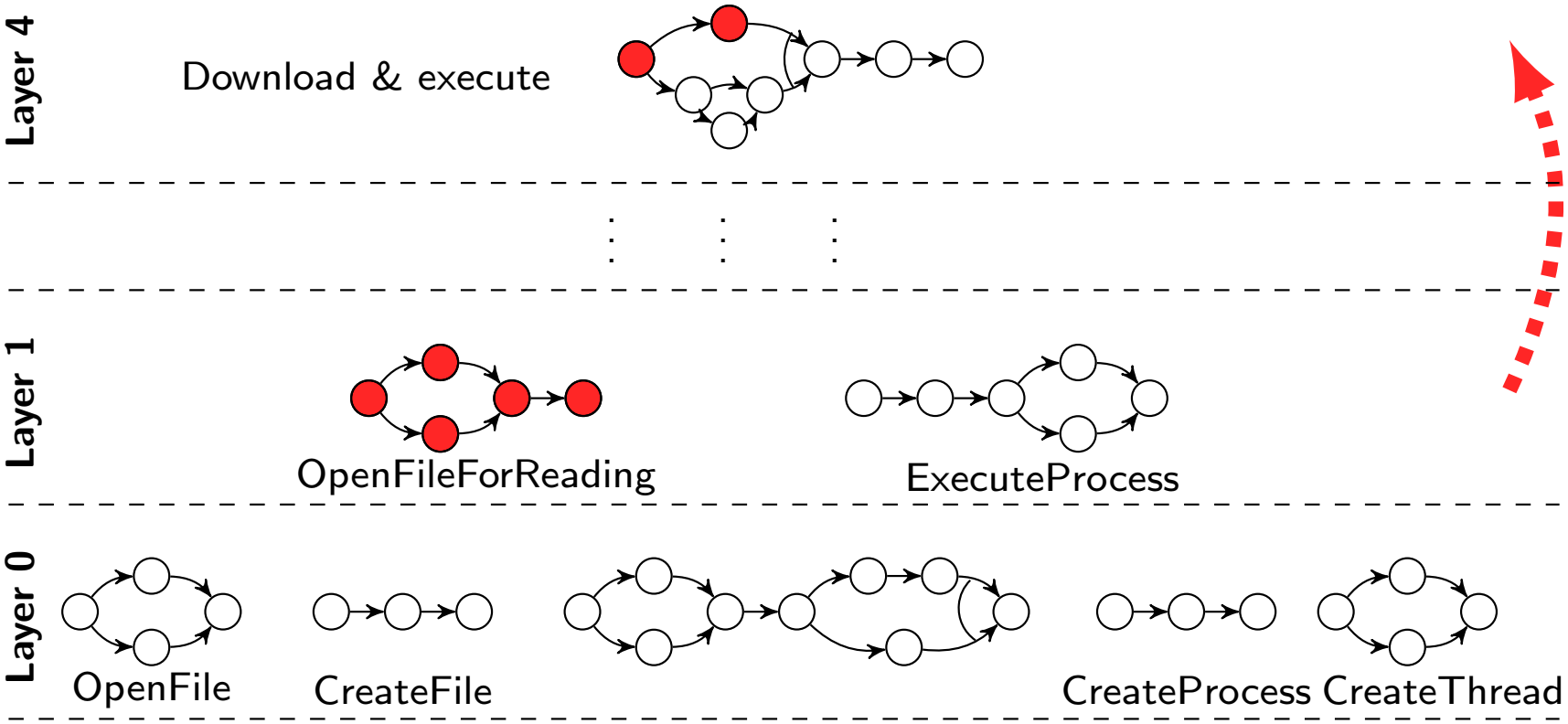
Matching malicious behaviors



The low-level behavior is matched

The synthetic event (acceptor node) is passed to the upper layer

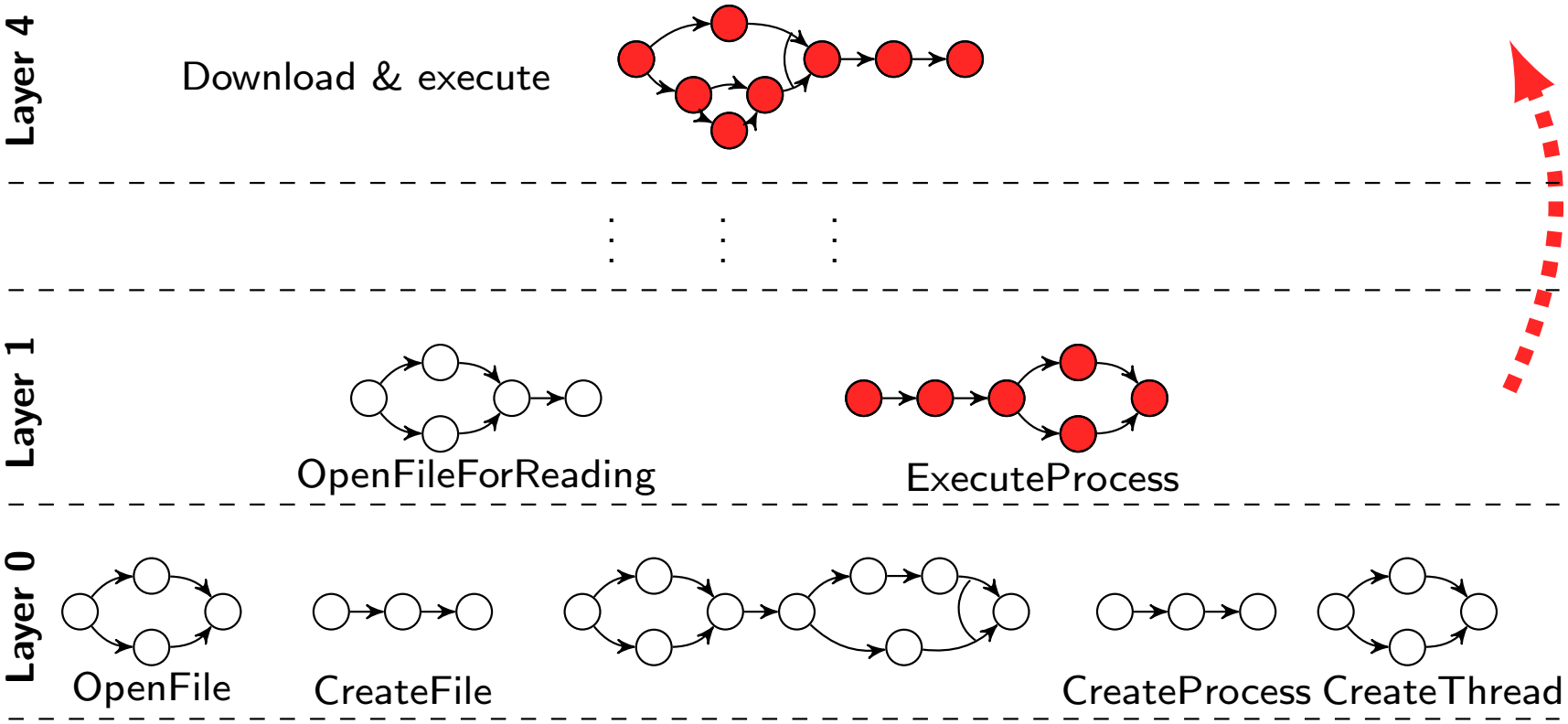
Matching malicious behaviors



The middle-level behavior is matched

The synthetic event (acceptor node) is passed to the upper layer

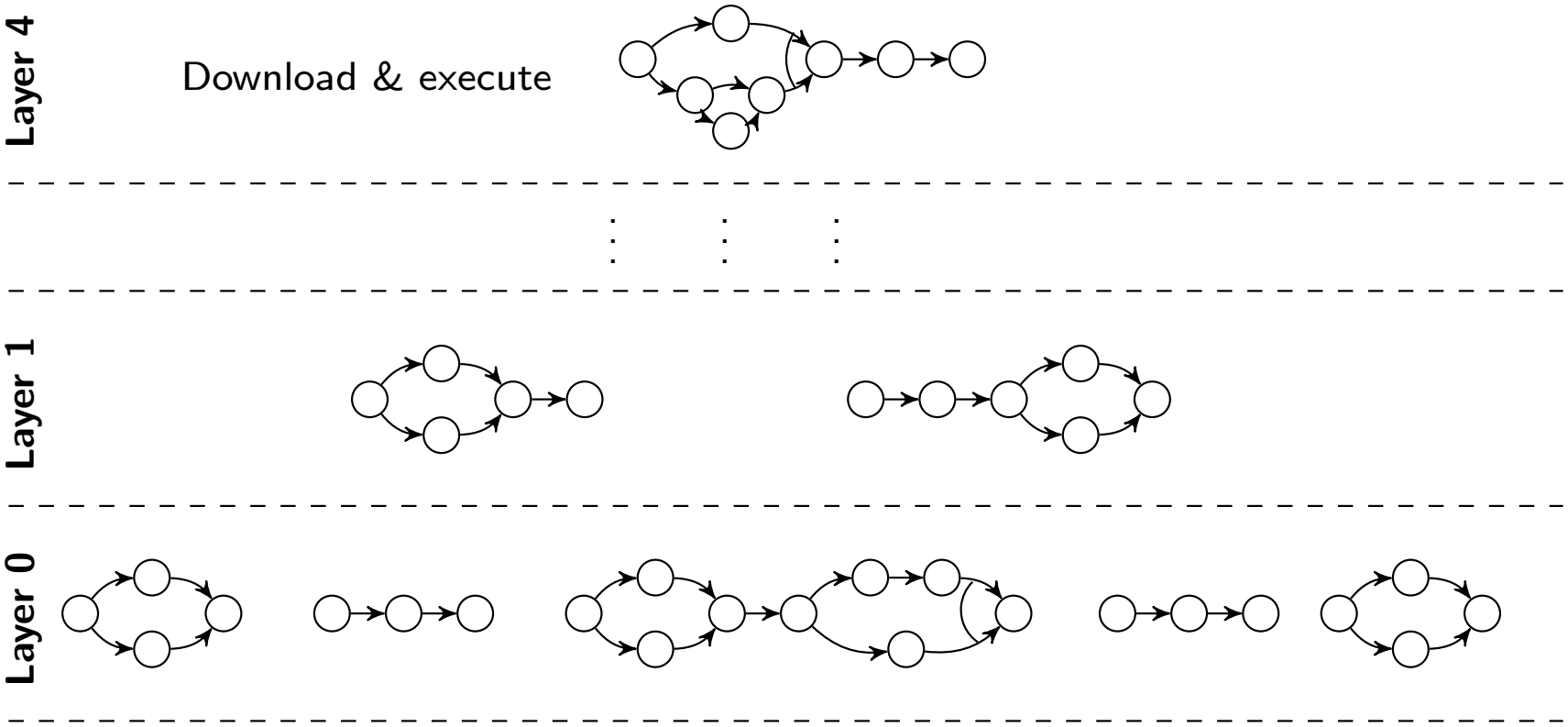
Matching malicious behaviors



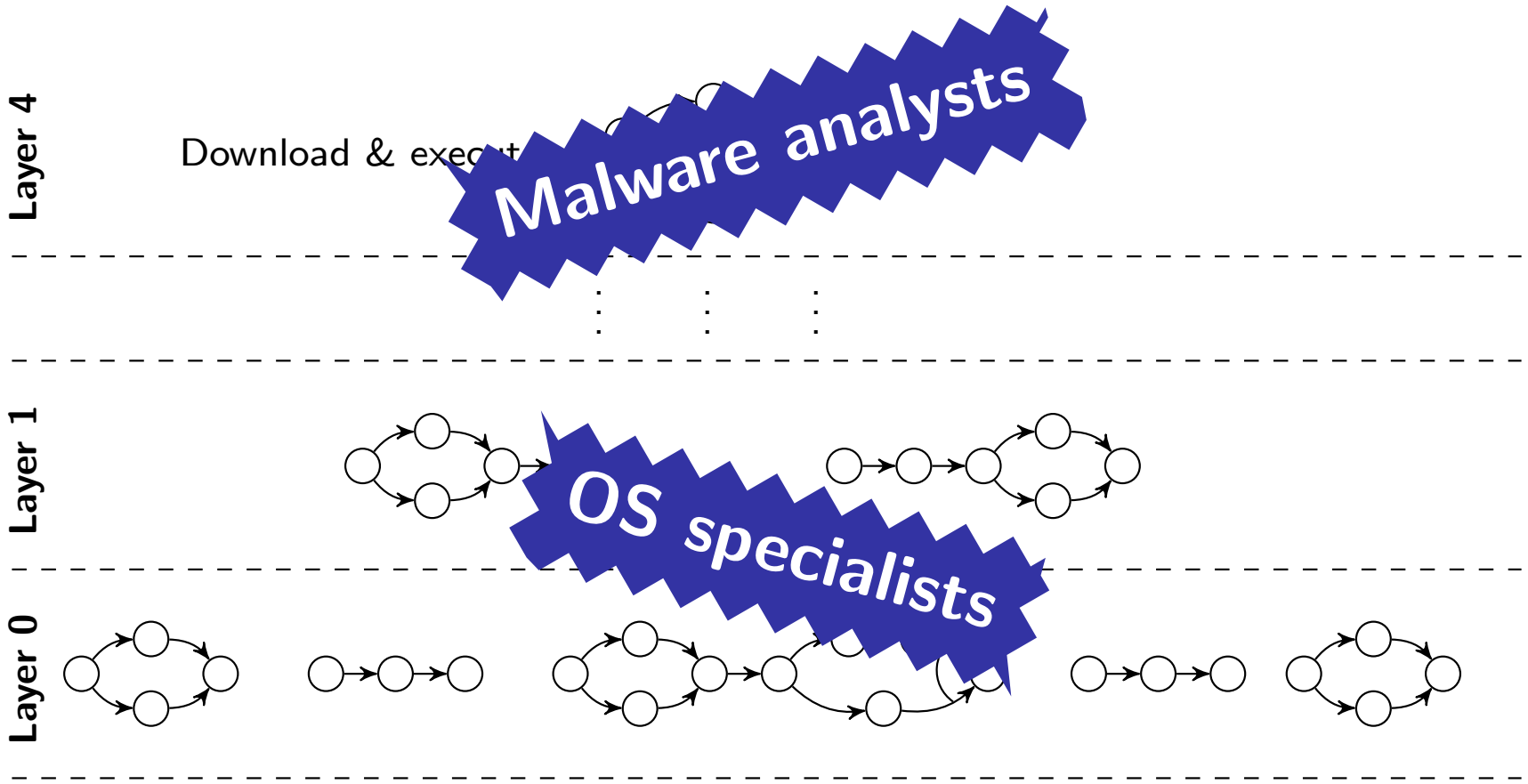
The highest-level behavior is matched

The program is malicious

Are we managing the complexity?

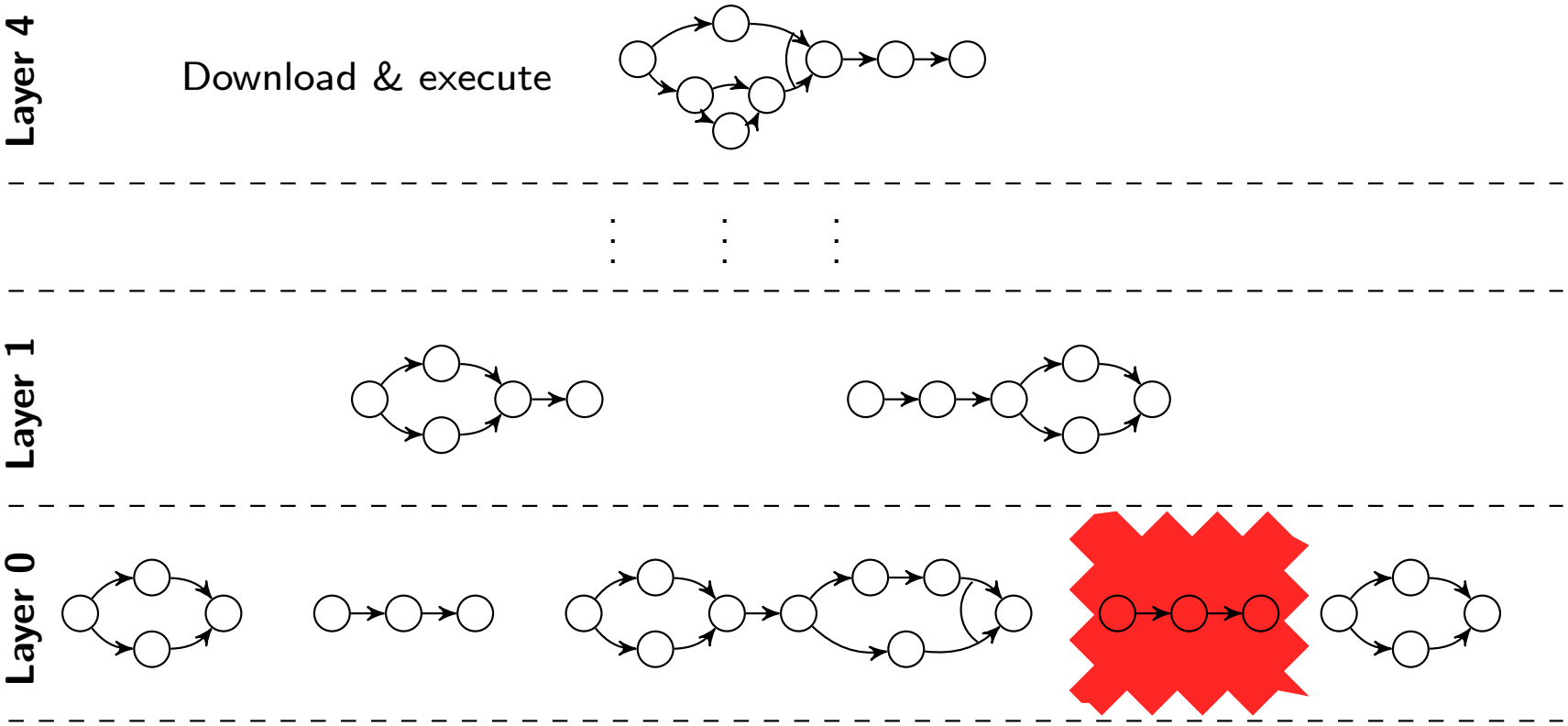


Are we managing the complexity?



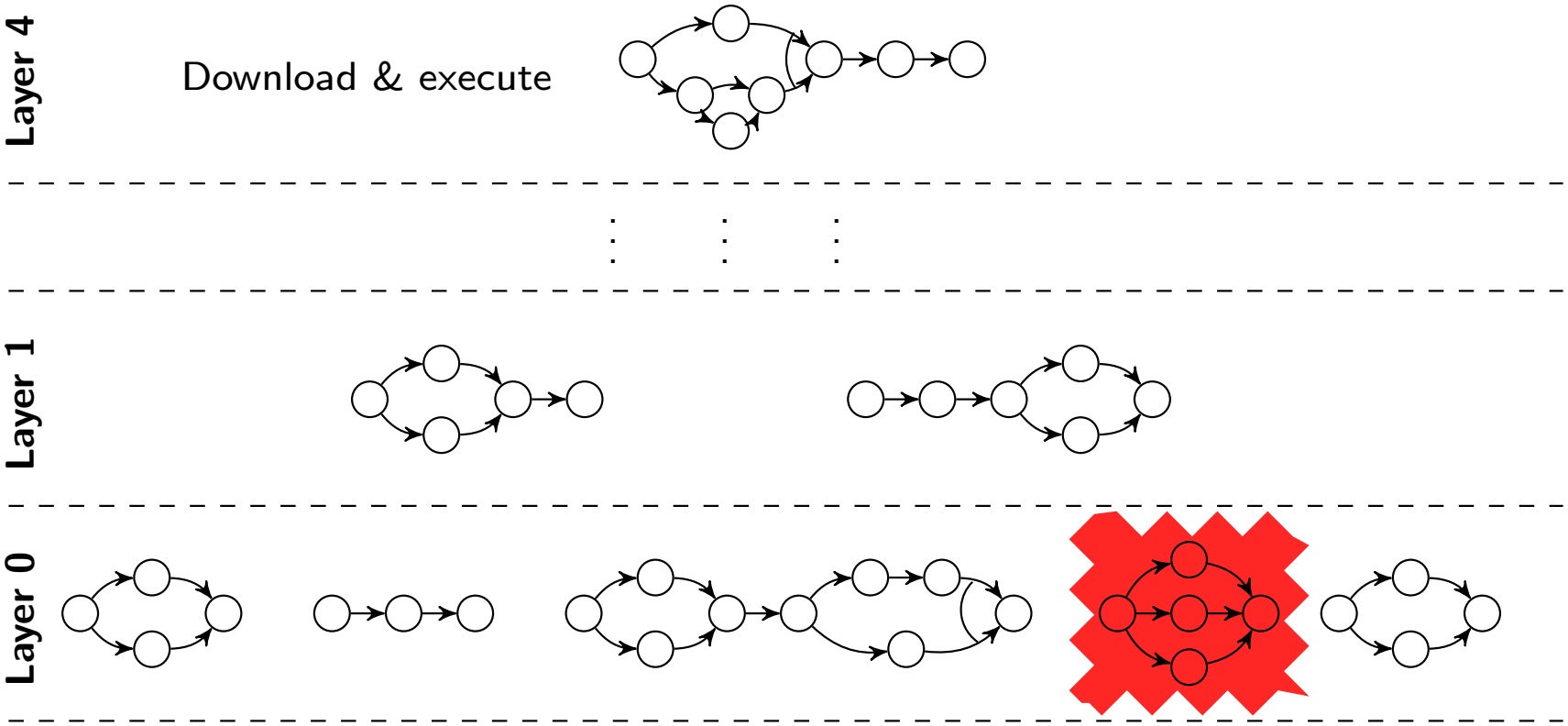
Imagine how easy it is to manage our models of malicious behaviors

Are we managing the complexity?



A behavior graph can be easily extended or replaced without needing to change the upper layers

Are we managing the complexity?



A behavior graph can be easily extended or replaced without needing to change the upper layers

Construction of behavior graphs

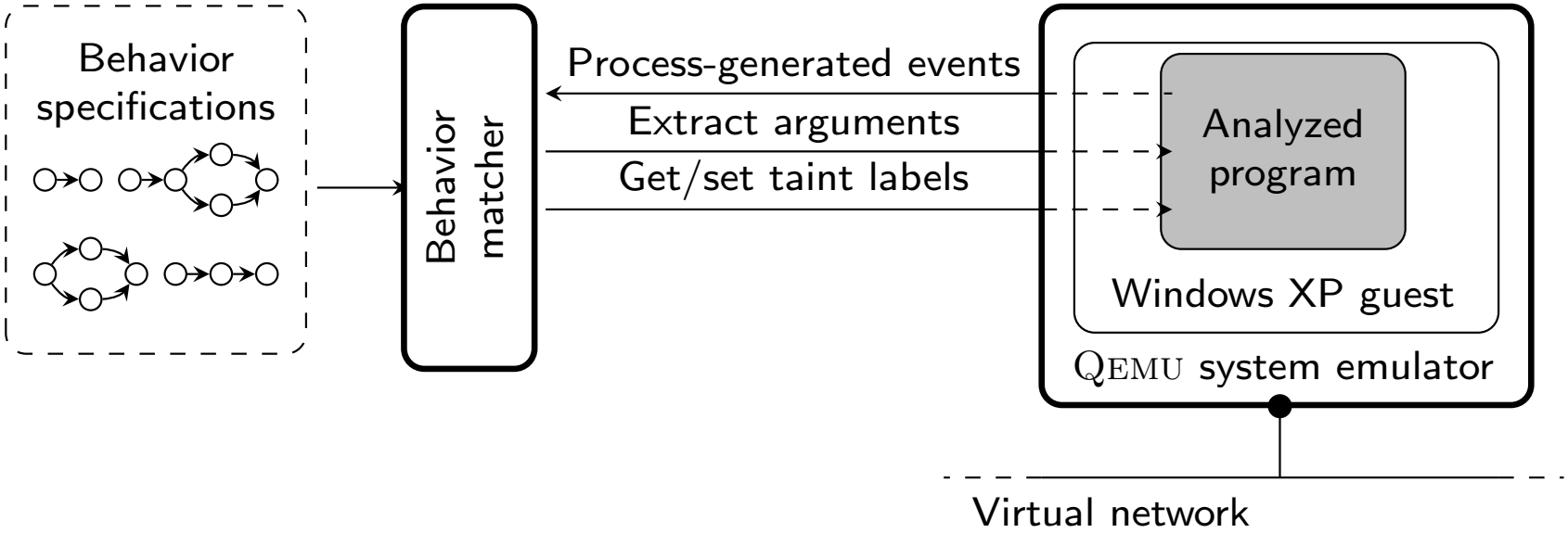
- Native API are not documented! We analyzed tens of gigabytes of execution traces of various applications to build the most complete behavior graphs
- We currently have more the 40 behavior graphs (for WinXP)

Construction of behavior graphs

- Native API are not documented! We analyzed tens of gigabytes of execution traces of various applications to build the most complete behavior graphs
- We currently have more the 40 behavior graphs (for WinXP)
- Some primitives of our language resulted from this effort:
  - TCPClient (8 graphs)
  - TCPServer (6 graphs)
  - TCPProxy (21 graphs)
  - NetDownload (15 graphs)
  - SendEmail (12 graphs)
  - KeyLogging

It becomes very easy, using our primitives, to model new behaviors (e.g., a malware that acts as a SMTP relay or that sends copies of itself via email)

Architecture of the system



- Customized QEMU that instruments the guest code to monitor system call invocations, to perform taint analysis, and to track local user input
- A behavior matcher that receives events in real-time and tries to match each behavior graph loaded

Evaluation

1. Do our behavior specifications adequately cover semantically-equivalent but programmatically-different execution paths?
2. Are our behavior specifications good for detecting bots and other kinds of malware?
3. Do our behavior specifications report any benign program as malicious?
4. What is the performance impact of the system?

Evaluation

1. *Do our behavior specifications adequately cover semantically-equivalent but programmatically-different execution paths?*

We ran a diverse suite of benign applications and we performed matching against a set of some innocuous intermediate behaviors

Evaluation

1. *Do our behavior specifications adequately cover semantically-equivalent but programmatically-different execution paths?*

We ran a diverse suite of benign applications and we performed matching against a set of some innocuous intermediate behaviors

Our graphs have a very good coverage

Evaluation

2. *Are our behavior specifications good for detecting bots and other kinds of malware?*

The system and the behavior specification were tested against 7 well known bots and 7 malware:

Behaviors:

- Remotely initated (RI) sendto
- RI create and execute file
- RI net download
- RI send email
- RI TCP proxy
- Keylogging
- Data leak
- Self propagation via email

Bots & other malware:

- rBot
- Agobot
- SpyBot
- ...
- Banker
- Delf
- Bagle

Evaluation

2. *Are our behavior specifications good for detecting bots and other kinds of malware?*

The system and the behavior specification were tested against 7 well known bots and 7 malware:

A single behavior graph (signature) can be used to identify the same malicious behavior in a variety of malware

- RI net download
- RI send email
- RI TCP proxy
- Keylogging
- Data leak
- Self propagation via email

- Agobot
- SpyBot
- ...
- Banker
- Delf
- Bagle

Evaluation

3. *Do our behavior specifications report any benign program as malicious?*

- The same behavior specifications used for malware detection were tested against 11 benign applications
- Each application was tested twice: with and without user input tracking

Evaluation

3. *Do our behavior specifications report any benign program as malicious?*

- The same behavior specifications used for malware detection were tested against 11 benign applications
- Each application was tested twice: with and without user input tracking

User input tracking is very important to distinguish between behaviors triggered by the user and behaviors triggered automatically:

- ∼25% FP without user input tracking
- ∼9% FP with user input tracking (Data leak is too coarse)

Evaluation

3. *Do our behavior specifications report any benign program as malicious?*

- The same behavior specifications used for malware detection were tested against 11 benign applications
- Each application was tested twice: with and without user input tracking

User input tracking is very important to distinguish between behaviors triggered by the user and behaviors triggered automatically:

- ∼25% FP without user input tracking
- ∼1% FP with user input tracking (Outlook)

Evaluation

4. *What is the performance impact of the system?*

Wallclock time between visible events for various configurations of the system:

	Tainting	**RICE**	**RISE**	**KL**
Internet Explorer	5.25	11.53	7.19	5.64
pSCP	7.32	8.08	19.62	7.42
Agobot	3.01	16.40	23.73	16.84
rBot	9.50	11.20	11.08	9.62

The overhead of vanilla QEMU (not included) is ∼7X-23X

Evaluation

4. *What is the performance impact of the system?*

Wallclock time between visible events for various configurations of the system:

	Tainting	**RICE**	**RISE**	**KL**
Internet Explorer	5.25	11.53	7.19	5.64
pSCP	7.32	8.08	19.62	7.42
Agobot	3.01	16.40	23.73	16.84
rBot	9.50	11.20	11.08	9.62

The overhead of vanilla QEMU (not included) is ∼7X-23X

The overhead introduced does not allow real-time analysis, but . . .

Usage scenarios

. . . there are many others alternative usage scenarios:

- Off-line detailed analysis
- Provide support to domain experts
- Distributed analysis

Take away

We have developed an effective framework for managing the semantic gap problem in behavior-based malware detection:

- Hierarchical behavior graphs allow to easily describe novel semantically meaningful behaviors
- The behavior graphs we have developed are good behavioral signatures for malware detection

A Layered Architecture for Detecting Malicious Behaviors

Lorenzo Martignoni Elizabeth Stinson
Matt Fredrikson Somesh Jha John Mitchell

RAID 2008

Thank you!
Any questions?