CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

Do You Even Lift? Strengthening Compiler Security Guarantees Against Spectre Attacks

Xaver Fabian[1,3], Marco Guarnieri[2], Marco Patrignani[3] ,Michael Backes[1]

1 CISPA Helmholtz Center for Information Security
2 IMDEA Software Institute
3 University of Trento

Spectre

Spectre attacks exploit speculative execution to leak confidential information

Spectre

Spectre attacks exploit speculative execution to leak confidential information

```
1. if idx < A_size
2.     y = A[idx]
3.     z = B[y]
```

← branch misprediction

Spectre

Spectre attacks exploit speculative execution to leak confidential information

```
1. if idx < A_size
2.     y = A[idx]
3.     z = B[y]
```

branch misprediction

- conditional branch prediction 

- indirect jump predictions 

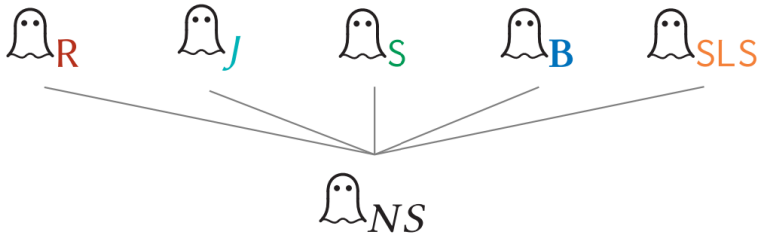
- return prediction 

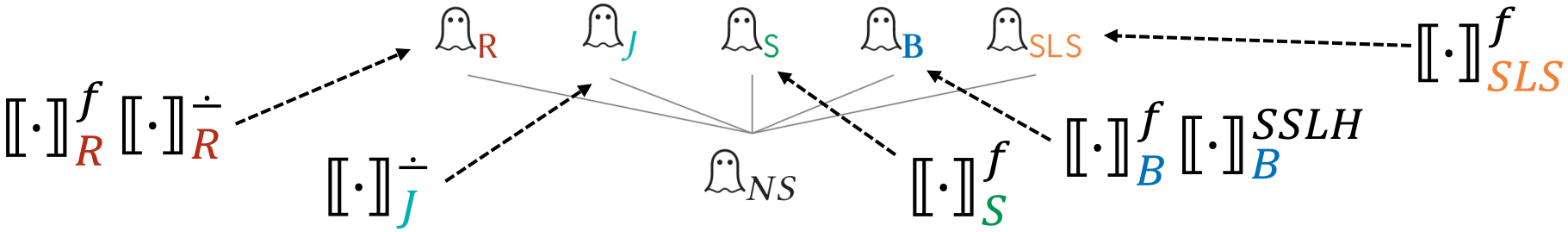
- store prediction 

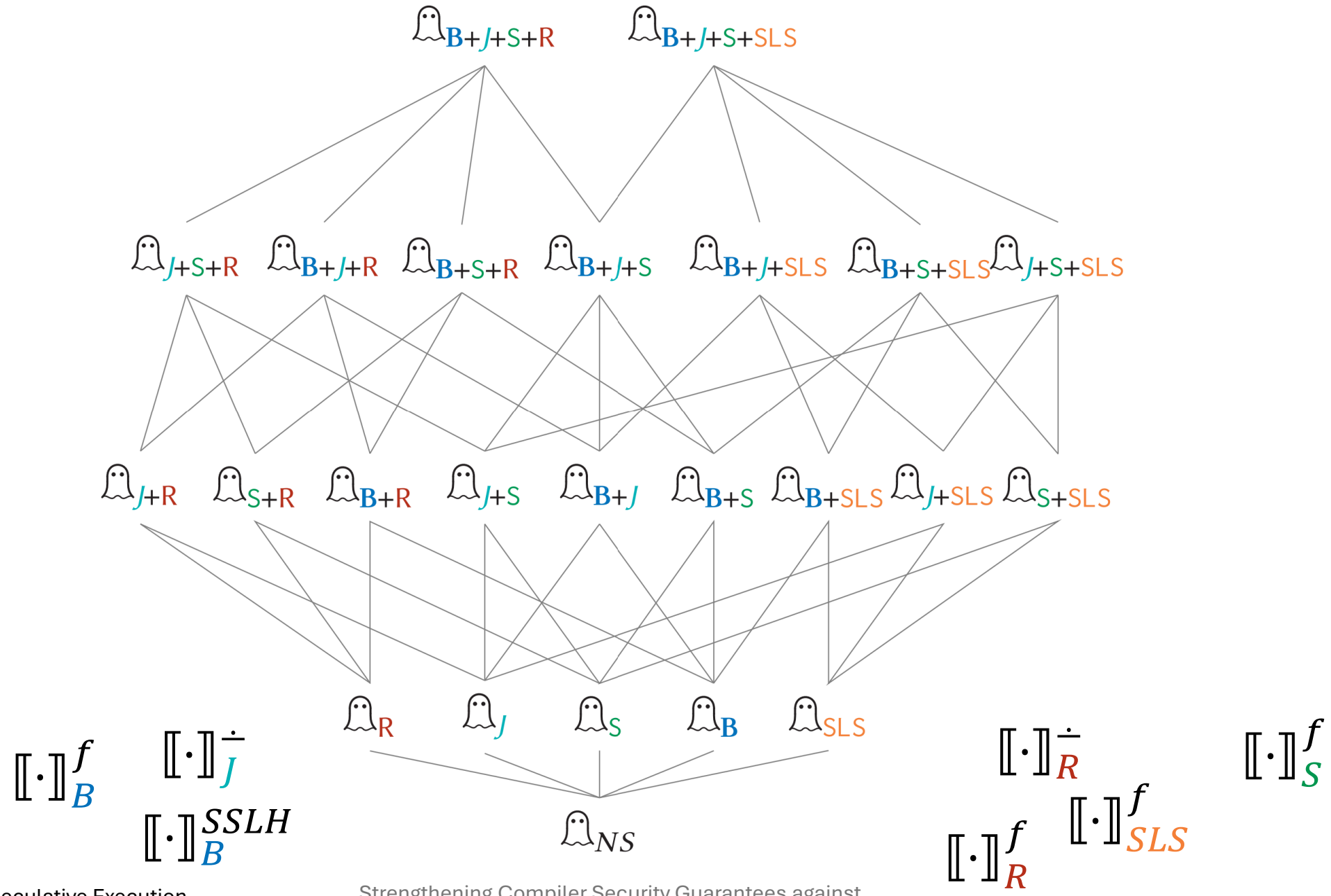
- branch type prediction 

Spectre Countermeasures

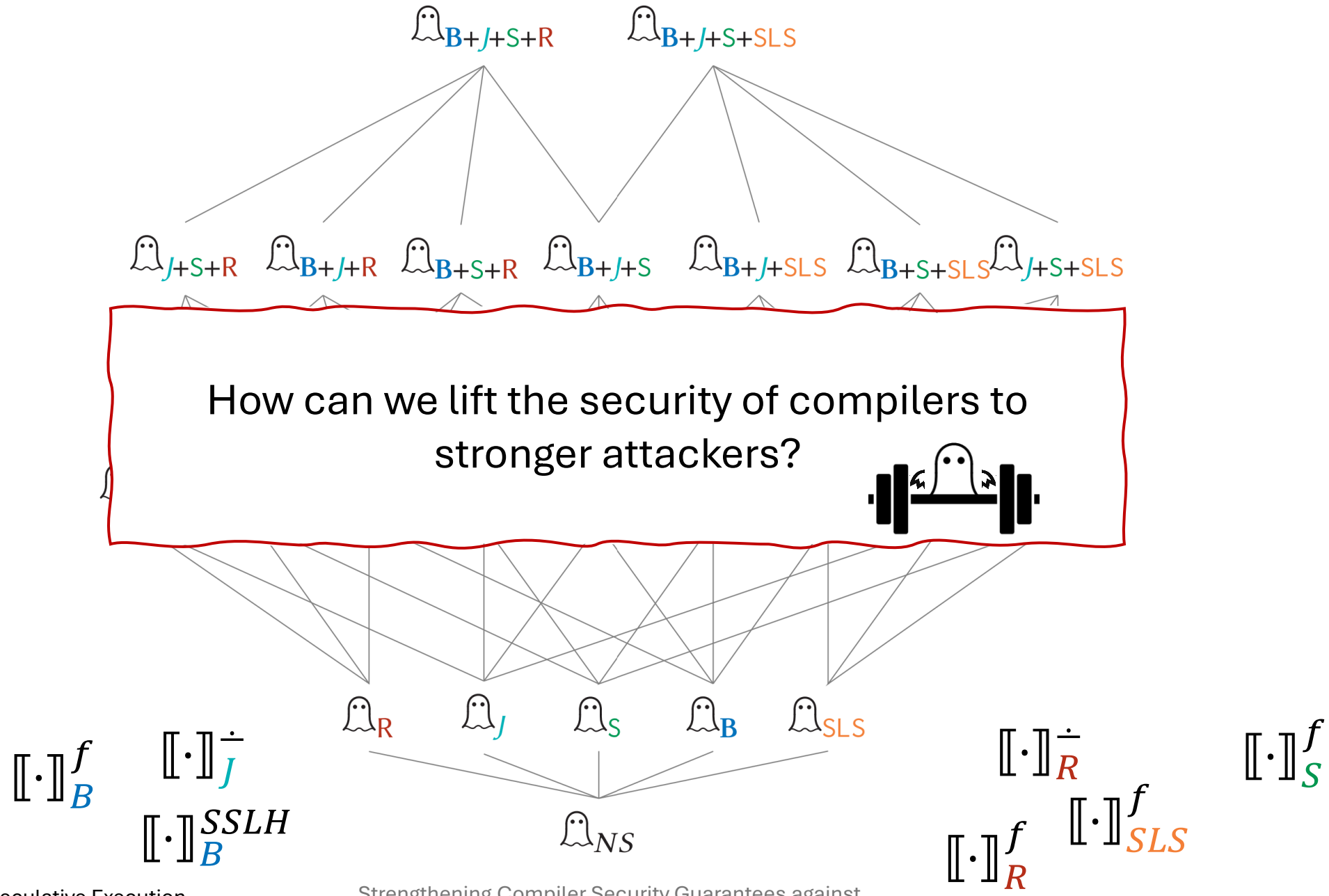
Name	Symbol	Base Semantics	Source
Fences for Returns Straight-Line	$[\![\cdot]\!]^{f}_{\mathsf{SLS}}$	SLS	GCC/Clang
Retpoline for Jumps	$[\![\cdot]\!]^{rpl}_{\mathcal{J}}$	$\mathcal{J}$	GCC/Clang/[36]
Retpoline with fence for Jumps	$[\![\cdot]\!]^{rplf}_{\mathcal{J}}$	$\mathcal{J}$	Gcc/[36]
Retpoline for Returns	$[\![\cdot]\!]^{rpl}_{\mathsf{R}}$	R	Gcc/[43]
Fences for Returns	$[\![\cdot]\!]^{f}_{\mathsf{R}}$	R	[43]
Fences for Stores	$[\![\cdot]\!]^{f}_{\mathsf{S}}$	S	[37]
Ultimate SLH for Branches	$[\![\cdot]\!]^{USLH}_{\mathbf{B}}$	$\mathbf{B}$	[59] (extends Clang's SLH)
Strong SLH for Branches	$[\![\cdot]\!]^{SSLH}_{\mathbf{B}}$	$\mathbf{B}$	[49] (extends Clang's SLH)
Fences for Branches	$[\![\cdot]\!]^{f}_{\mathbf{B}}$	$\mathbf{B}$	ICC/Clang


R
J
S
B
SLS
NS


R
J
S
B
SLS
NS
$[\![\cdot]\!]^{f}_{R}\,[\![\cdot]\!]^{\div}_{R}$
$[\![\cdot]\!]^{\div}_{J}$
$[\![\cdot]\!]^{f}_{S}$
$[\![\cdot]\!]^{f}_{B}\,[\![\cdot]\!]^{SSLH}_{B}$
$[\![\cdot]\!]^{f}_{SLS}$



Automatic Detection of Speculative Execution Combinations, Fabian et al., CCS '22



Automatic Detection of Speculative Execution Combinations, Fabian et al., CCS '22

Interaction of Semantics: Trace Shapes

Shape of Traces for program p under $x + y$

$$start_x \cdot \bar{\tau} \cdot$$

Interaction of Semantics: Trace Shapes

Shape of Traces for program p under $x + y$

$$start_x \cdot \bar{\tau} \cdot start_y \cdot \overline{\tau'} \cdot rlb_y$$

Interaction of Semantics: Trace Shapes

Shape of Traces for program p under ghost_{x+y}

$$start_x \cdot \bar{\tau} \cdot start_y \cdot \overline{\tau'} \cdot rlb_y \cdot \overline{\tau''} \cdot rlb_x$$

Interaction of Semantics: Trace Shapes

Shape of Traces for program p under $x + y$

$$start_x \cdot \bar{\tau} \cdot start_y \cdot \overline{\tau'} \cdot rlb_y \cdot \overline{\tau''} \cdot rlb_x$$

Or

$$start_y \cdot \bar{\tau} \cdot start_x \cdot \overline{\tau'} \cdot rlb_x \cdot \overline{\tau''} \cdot rlb_y$$

Interaction of Semantics: Trace Shapes

Shape of Traces for program p under $x + y$

Challenge 1

$$start_x \cdot \bar{\tau} \cdot start_y \cdot \overline{\tau'} \cdot rlb_y \cdot \overline{\tau''} \cdot rlb_x$$

Or

Challenge 3

$$start_y \cdot \bar{\tau} \cdot start_x \cdot \overline{\tau'} \cdot rlb_x \cdot \overline{\tau''} \cdot rlb_y$$

Challenge 2

Security Definition

$\text{👻}_x \vdash p : \mathit{Secure}$

Does not speculative leak more than non-speculative execution

$\text{👻}_x \vdash [\![\cdot]\!] : \mathit{SecP}$ [1]

Preservation of Secure

$:= \quad \text{👻}_{NS} \vdash p : \mathit{Secure}$ **then** $\text{👻}_x \vdash [\![p]\!] : \mathit{Secure}$

[1] Exorcising Spectres with Secure Compilers, Patrignani and Guarnieri, CCS '21

Independence in Extension

Compiler for **x** independent for **version y**

$\text{👻}_y \vdash [\![\cdot]\!] : I \coloneqq \text{👻}_y \vdash p : \mathit{Secure}$ **then** $\text{👻}_y \vdash [\![p]\!] : \mathit{Secure}$

Independence in Extension

Compiler for **x** independent for **version y**

$\text{👻}_y \vdash [\![\cdot]\!] : I \coloneqq \text{👻}_y \vdash p : \mathit{Secure}$ **then** $\text{👻}_y \vdash [\![p]\!] : \mathit{Secure}$

Compiler does not introduce new leaks under the extension semantics

Safe Nesting

$$start_x \cdot \bar{\tau} \cdot start_y \cdot \overline{\tau'} \cdot rlb_y \cdot \overline{\tau''} \cdot rlb_x$$

Or

$$start_y \cdot \bar{\tau} \cdot start_x \cdot \overline{\tau'} \cdot rlb_x \cdot \overline{\tau''} \cdot rlb_y$$

Compiler Criterion?

$\text{👻}_x \vdash [\![\cdot]\!] : SecP \coloneqq \text{👻}_{NS} \vdash p : Secure$ **then** $\text{👻}_x \vdash [\![p]\!] : Secure$

New Compiler Criterion: Conditional SecP

$$\text{👻}_x, \text{👻}_y \vdash [\![\cdot]\!] : CSecP \coloneqq \text{👻}_{NS} \vdash p : Secure \qquad \text{👻}_y \vdash p : Secure$$

then $\text{👻}_x +_y \vdash [\![p]\!] : Secure$

Lifting Theorem

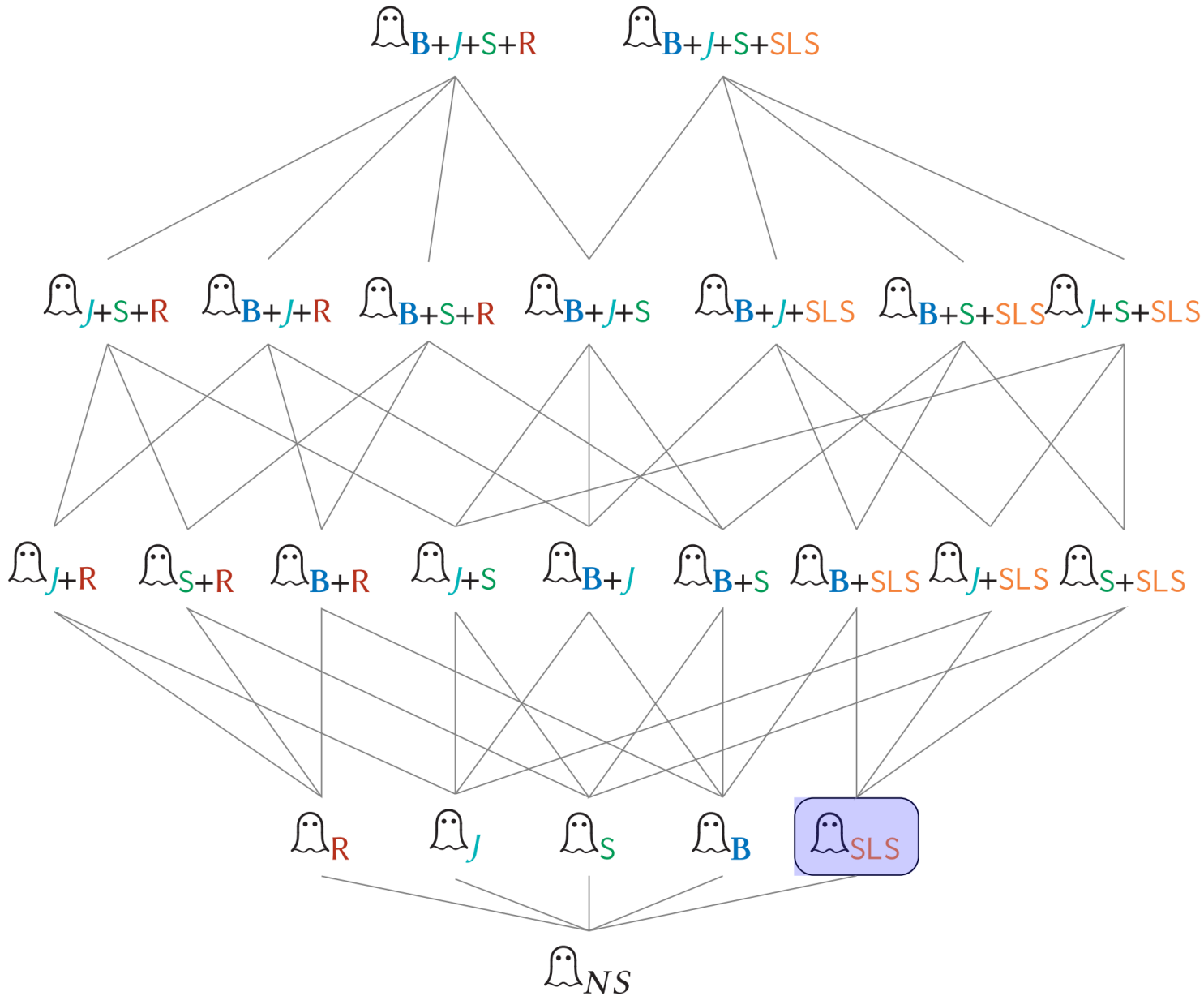
$start_x \cdot \bar{\tau} \cdot start_y \cdot \overline{\tau'} \cdot rlb_y \cdot \overline{\tau''} \cdot rlb_x$

$start_y \cdot \bar{\tau} \cdot start_x \cdot \overline{\tau'} \cdot rlb_x \cdot \overline{\tau''} \cdot rlb_y$

✓ ✓ ✓ **then** $\text{👻}_x, \text{👻}_y \vdash [\![\cdot]\!] : CSecP$

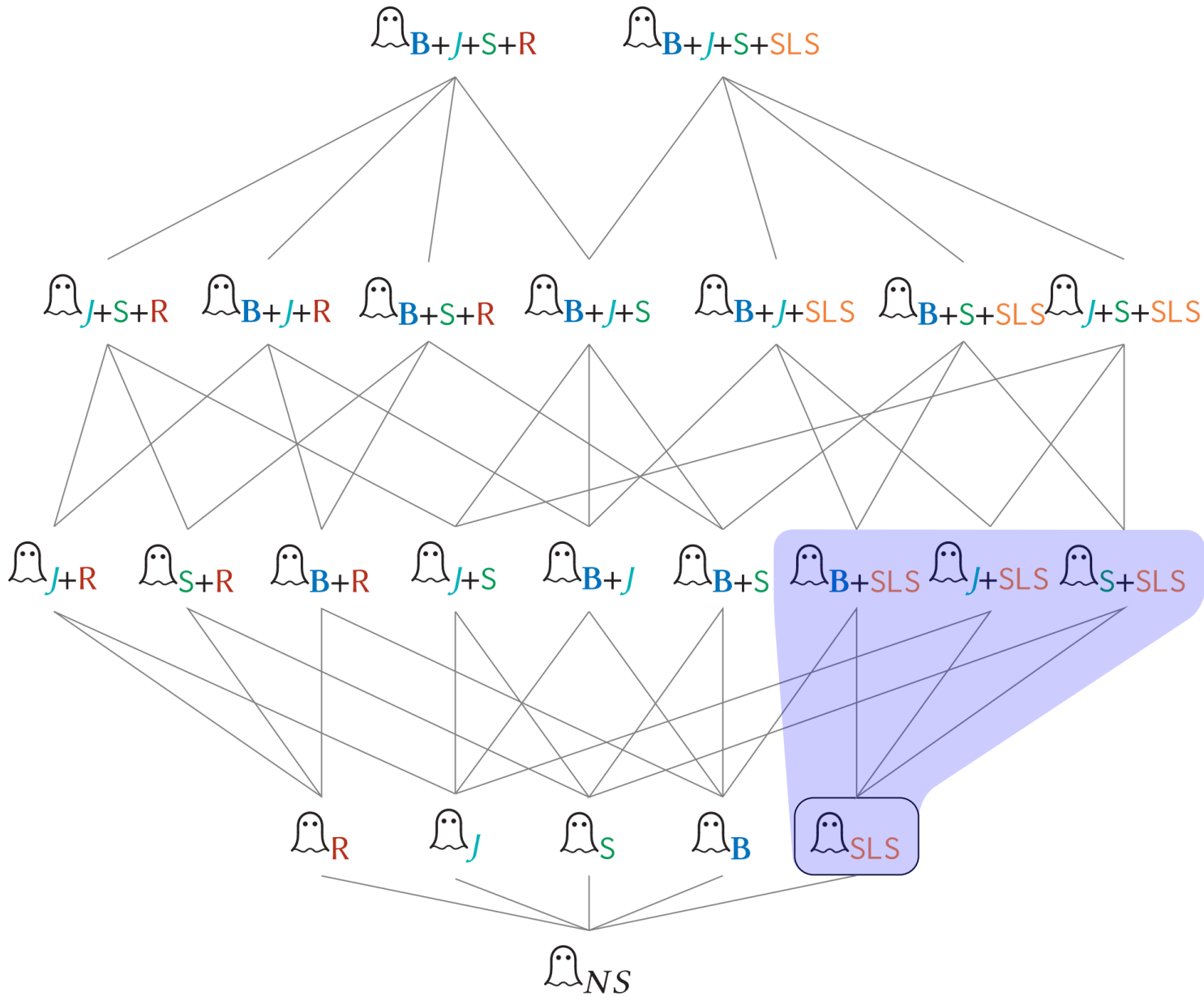
Compilers

$[\![\cdot]\!]^{f}_{\mathrm{SLS}}$



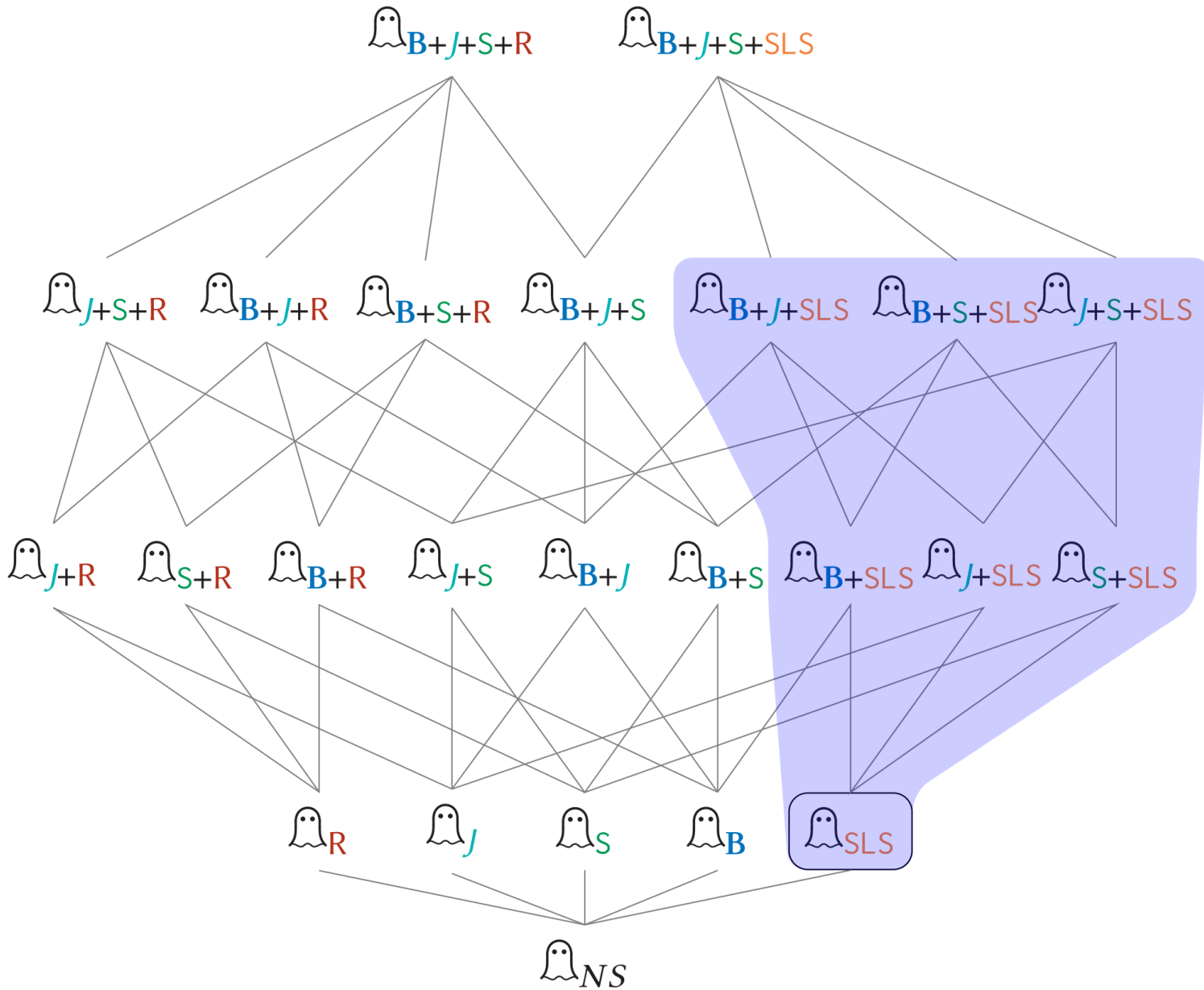
Compilers

$[\![\cdot]\!]^{f}_{\mathrm{SLS}}$



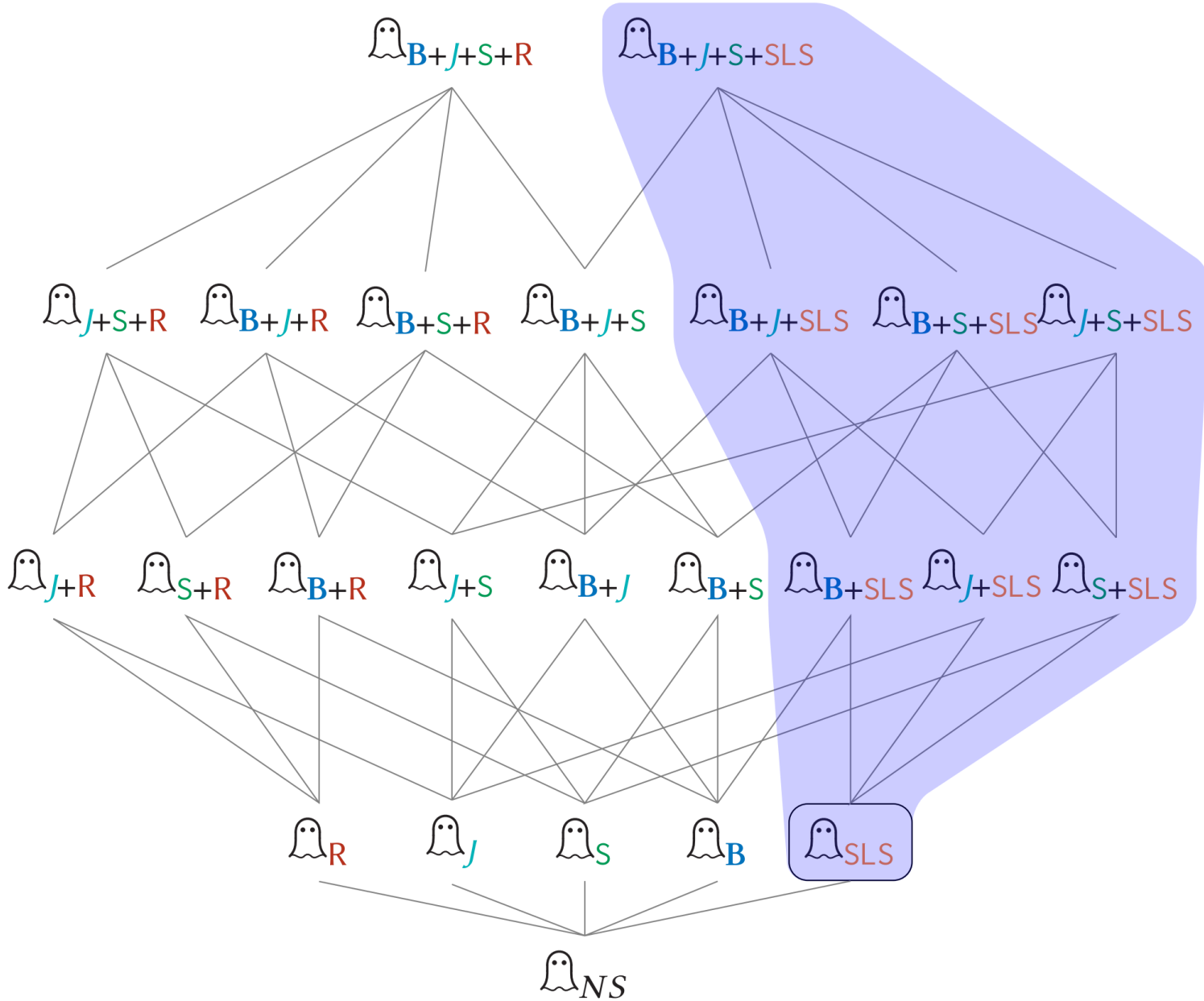
Compilers

$[\![\cdot]\!]^{f}_{\mathrm{SLS}}$

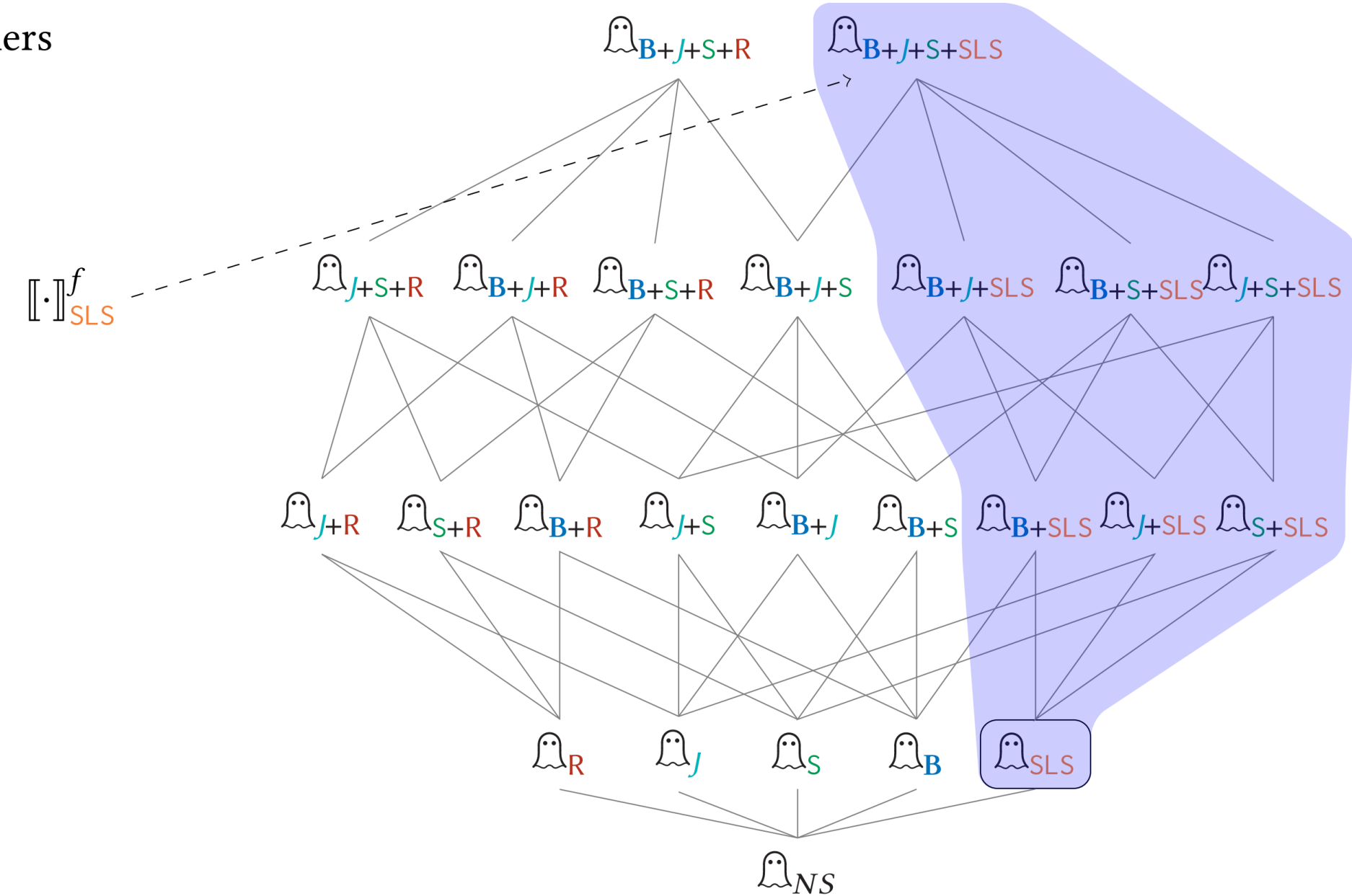


Compilers

$[\![\cdot]\!]^{f}_{\mathrm{SLS}}$



Compilers

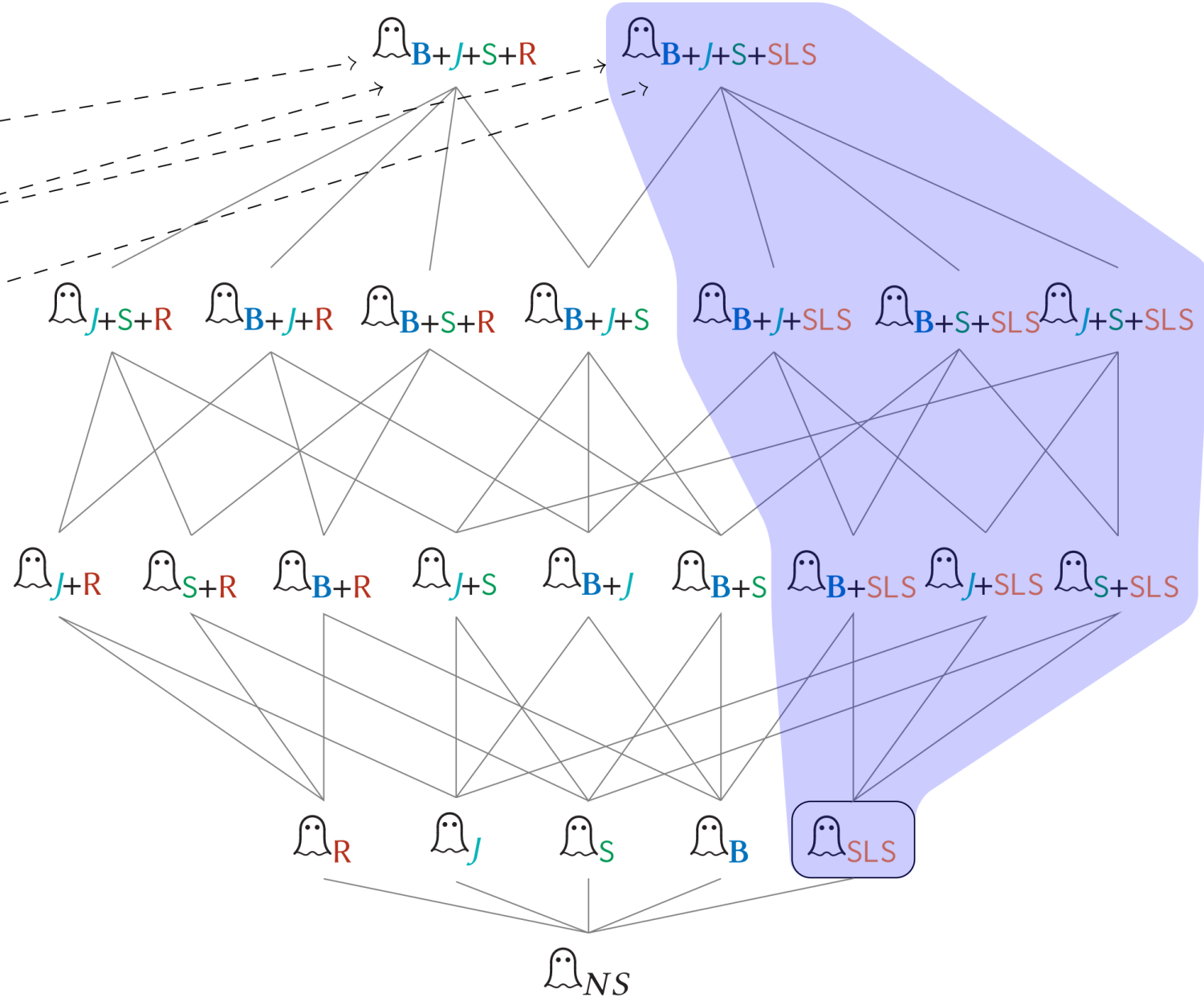

B+J+S+R
B+J+S+SLS
$[\![\cdot]\!]^{f}_{\mathrm{SLS}}$
J+S+R
B+J+R
B+S+R
B+J+S
B+J+SLS
B+S+SLS
J+S+SLS
J+R
S+R
B+R
J+S
B+J
B+S
B+SLS
J+SLS
S+SLS
R
J
S
B
SLS
NS

Compilers

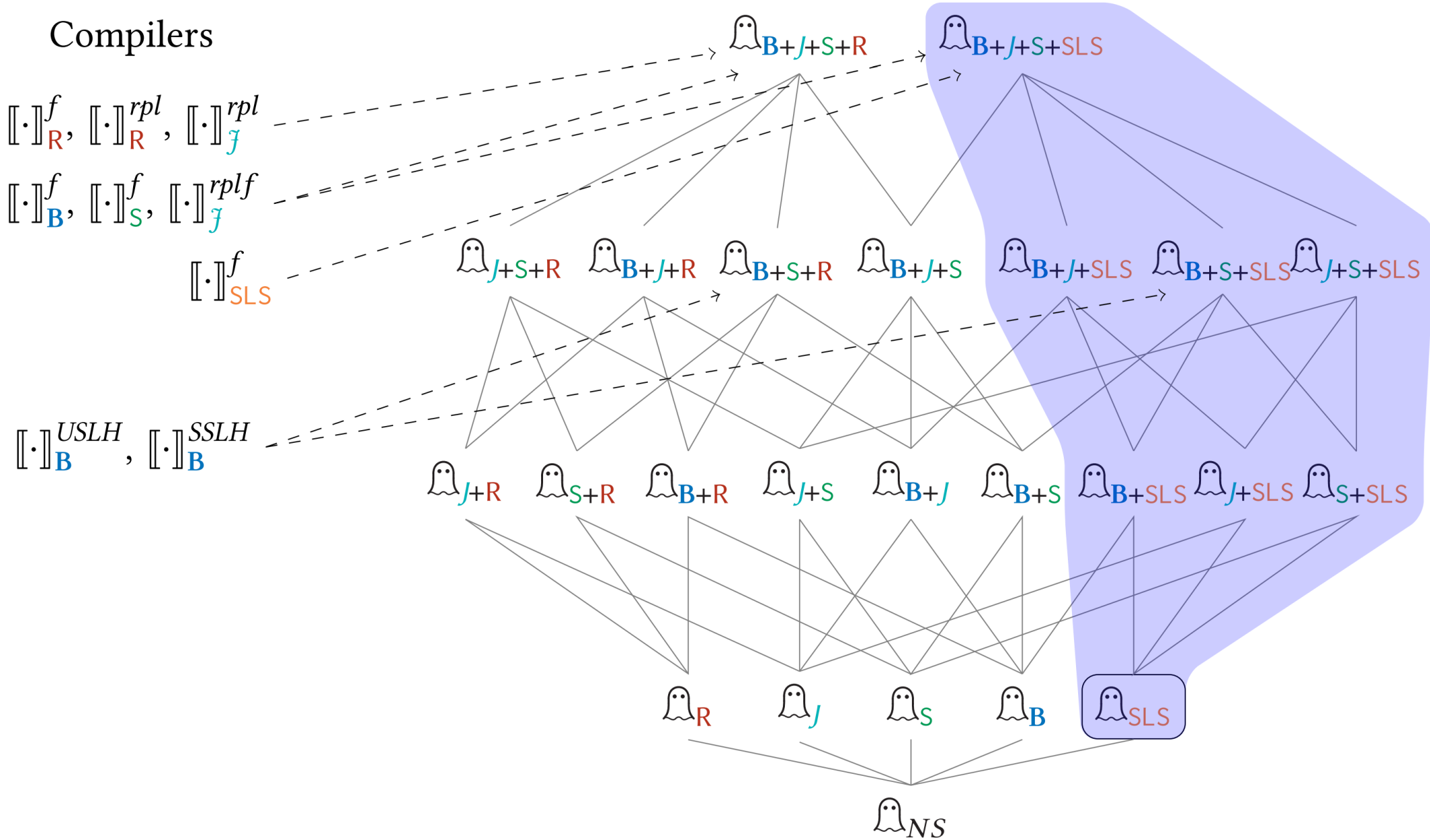
$[\![\cdot]\!]^{f}_{\mathsf{R}}$, $[\![\cdot]\!]^{rpl}_{\mathsf{R}}$, $[\![\cdot]\!]^{rpl}_{\mathcal{J}}$

$[\![\cdot]\!]^{f}_{\mathbf{B}}$, $[\![\cdot]\!]^{f}_{\mathsf{S}}$, $[\![\cdot]\!]^{rplf}_{\mathcal{J}}$

$[\![\cdot]\!]^{f}_{\mathsf{SLS}}$



Compilers


$[\![\cdot]\!]^{f}_{\mathsf{R}}$, $[\![\cdot]\!]^{rpl}_{\mathsf{R}}$, $[\![\cdot]\!]^{rpl}_{\mathcal{J}}$
$[\![\cdot]\!]^{f}_{\mathbf{B}}$, $[\![\cdot]\!]^{f}_{\mathsf{S}}$, $[\![\cdot]\!]^{rplf}_{\mathcal{J}}$
$[\![\cdot]\!]^{f}_{\mathsf{SLS}}$
$[\![\cdot]\!]^{USLH}_{\mathbf{B}}$, $[\![\cdot]\!]^{SSLH}_{\mathbf{B}}$
B+J+S+R
B+J+S+SLS
J+S+R
B+J+R
B+S+R
B+J+S
B+J+SLS
B+S+SLS
J+S+SLS
J+R
S+R
B+R
J+S
B+J
B+S
B+SLS
J+SLS
S+SLS
R
J
S
B
SLS
NS

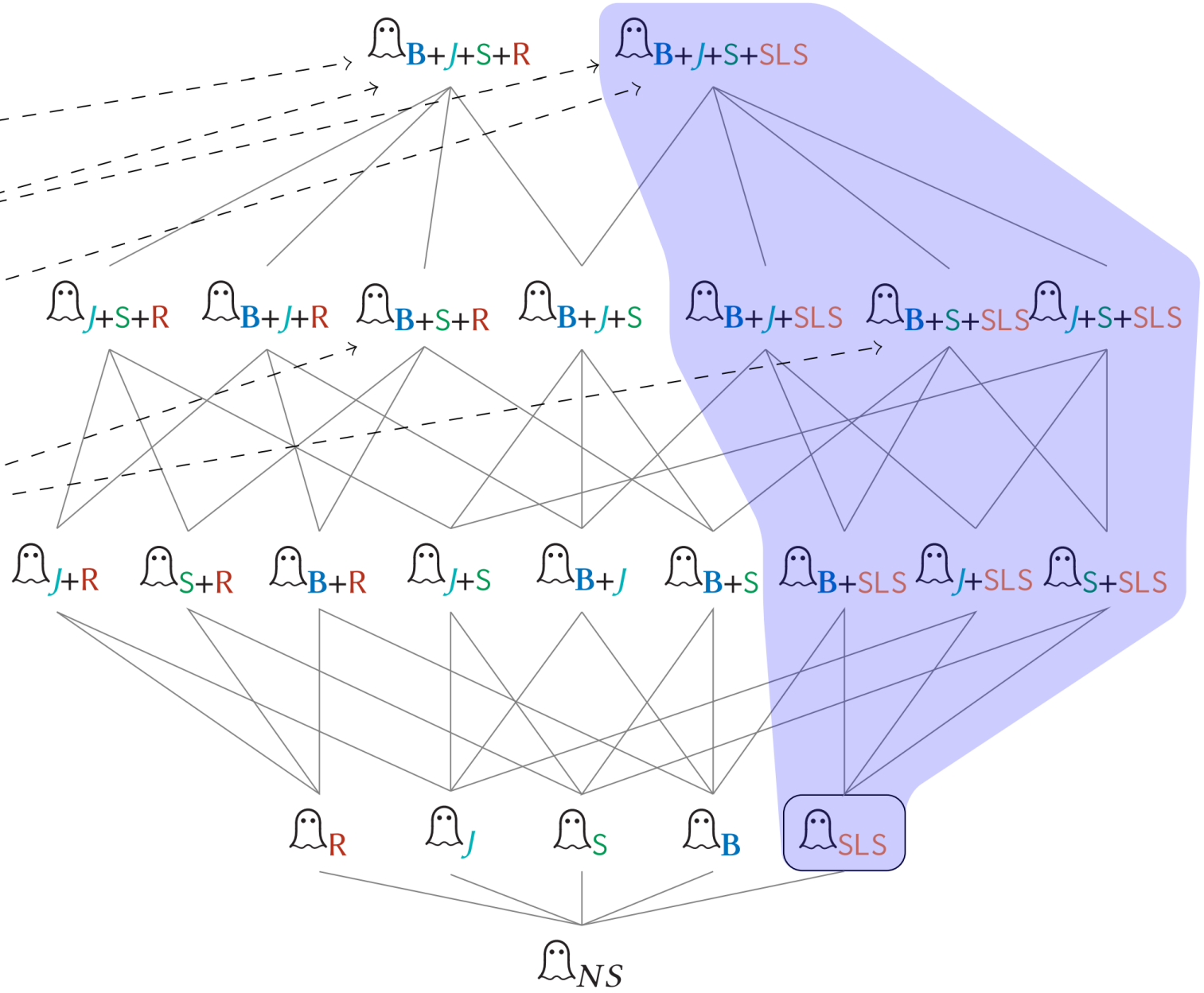
Compilers

$[\![\cdot]\!]^{f}_{\mathsf{R}}$, $[\![\cdot]\!]^{rpl}_{\mathsf{R}}$, $[\![\cdot]\!]^{rpl}_{\mathcal{J}}$

$[\![\cdot]\!]^{f}_{\mathbf{B}}$, $[\![\cdot]\!]^{f}_{\mathsf{S}}$, $[\![\cdot]\!]^{rplf}_{\mathcal{J}}$

$[\![\cdot]\!]^{f}_{\mathsf{SLS}}$

$[\![\cdot]\!]^{USLH}_{\mathbf{B}}$, $[\![\cdot]\!]^{SSLH}_{\mathbf{B}}$



Reducing the Proof Burden

Independence

Safe Nesting

Reducing the Proof Burden

Syntactic Independence

Independence

Safe Nesting

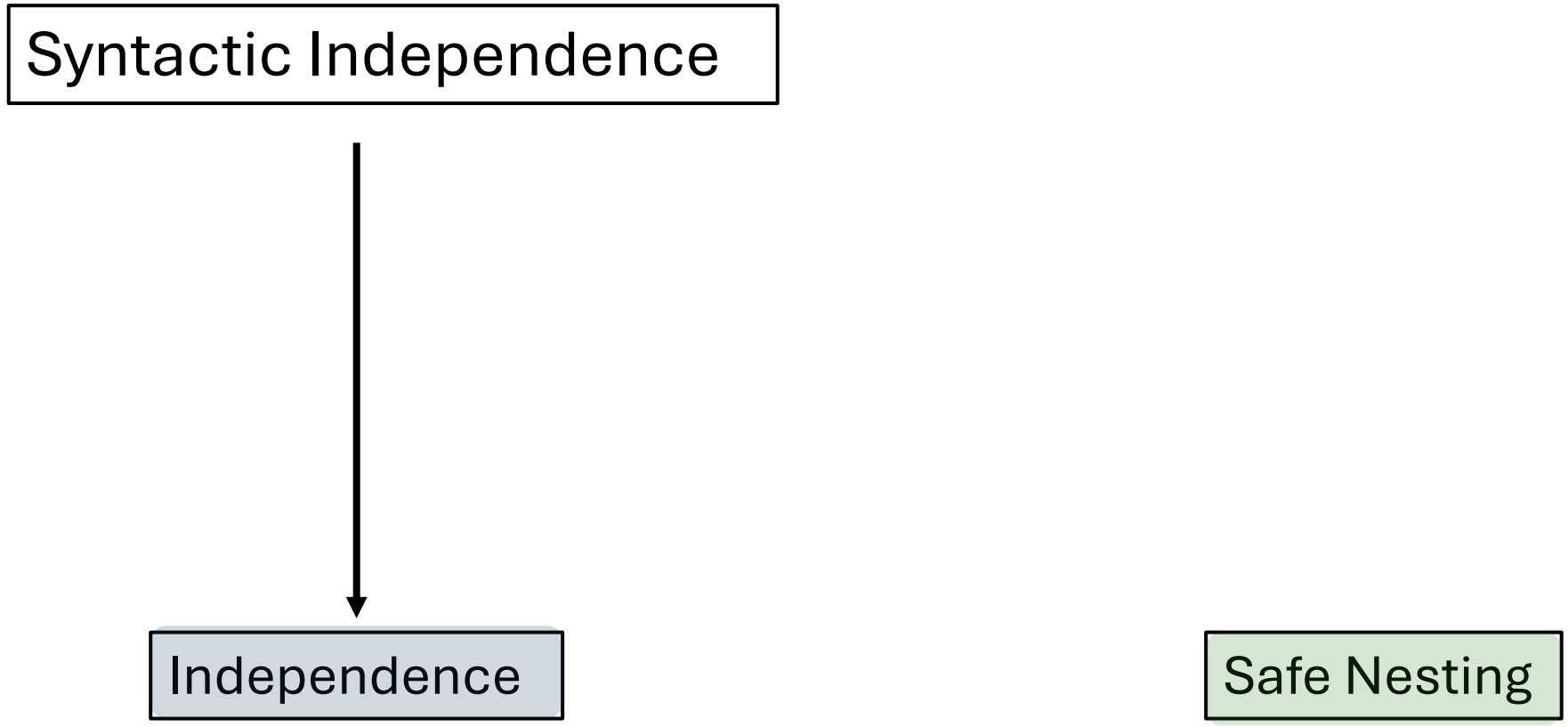
Reducing the Proof Burden

Syntactic Independence

Independence

Safe Nesting

Reducing the Proof Burden



Reducing the Proof Burden

Syntactic Independence

Trapped Speculation

Independence

Safe Nesting

Reducing the Proof Burden

Syntactic Independence

Trapped Speculation

Independence

Safe Nesting

Reducing the Proof Burden

Syntactic Independence

Trapped Speculation

Independence

Safe Nesting

$[\![\cdot]\!]^{f}_{SLS}$

$[\![\cdot]\!]^{f}_{B}$

$[\![\cdot]\!]^{f}_{S}$

$[\![\cdot]\!]^{\dot{-}}_{R}$

$[\![\cdot]\!]^{f}_{R}$

$[\![\cdot]\!]^{\dot{-}}_{J}$

Thank you!

Paper

Backup Slides

Compiler Composition ?!

Applicabiltiy to other Areas

Question: How to fulfil the side condition in CRSSP?

THEOREM 4 (LIFTED COMPILER PRESERVATION, 🐓). *If* $\text{👻}_x \vdash [\![\cdot]\!] : RSSP$ *and* $\text{👻}_y \vdash [\![\cdot]\!] : I$ *and* $\text{👻}_{x+y} \vdash [\![\cdot]\!] : safeN$ *and* $\vdash \text{👻}_{x+y} : WFC$ *, then* $\text{👻}_x, \text{👻}_y \vdash [\![\cdot]\!] : CRSSP$.

Definition 9 (Conditional Robust Speculative Safety Preservation (*CRSSP*)).

$$\text{👻}_x, \text{👻}_y \vdash [\![\cdot]\!] : CRSSP \stackrel{\text{def}}{=} \forall P \in L.\ \textit{if}\ \text{👻}_{NS} \vdash P : RSS\ \textit{and}\ \text{👻}_y \vdash P : RSS,\ \textit{then}\ \text{👻}_{x+y} \vdash [\![P]\!] : RSS.$$

Independence Example: Retpoline

```
1. jmp rax
```

$[\![\cdot]\!]^{\dot{-}}$

```
call thunk

thunk:
call LIND1

LIND0:
pause
lfence
jmp
LIND1:
mov rsp, rax
ret
```

Independence Example: Retpoline

```
1. jmp rax
```

$[\![\cdot]\!]^{\dot{-}}$

```
call thunk

thunk:
call LIND1

LIND0:
pause
lfence
jmp
LIND1:
mov rsp, rax
ret
```

$\mathcal{C}_J \vdash [\![\cdot]\!]^{\dot{-}} : RSNIP$

Independence Example: Retpoline

```
1. jmp rax
```

$[\![\cdot]\!]^{\dot{-}}$

```
call thunk

thunk:
call LIND1

LIND0:
pause
lfence
jmp
LIND1:
mov rsp, rax
ret
```

$\mathcal{C}_J \vdash [\![\cdot]\!]^{\dot{-}}$: Secure

$\mathcal{C}_R$?

Independence Example: Retpoline

```
1. jmp rax
```

$[\![\cdot]\!]^{\dot{-}}$

```
call thunk

thunk:
call LIND1

LIND0:
pause
lfence
jmp
LIND1:
mov rsp, rax
ret
```

$\mathcal{C}_J \vdash [\![\cdot]\!]^{\dot{-}} : Secure$

$\mathcal{C}_R$?

Syntactic Independence

Instructions related to speculation

Instructions inserted by the compiler

$$\mathcal{C}_J \quad \{jmp\} \quad \cap \quad \{lfence\} \quad [\![\cdot]\!]^f$$

$$\{\ \}$$

$$\mathcal{C}_B \vdash [\![\cdot]\!]^f : SI_{\mathcal{C}_J}$$

Interactions of Semantics

Shape of Traces for program p under $x + y$

Challenge 1

$$start_x \cdot \bar{\tau} \cdot start_y \cdot \bar{\tau'} \cdot rlb_y \cdot \overline{\tau''} \cdot rlb_x$$

Or

Challenge 3

$$start_y \cdot \bar{\tau} \cdot start_x \cdot \bar{\tau'} \cdot rlb_x \cdot \overline{\tau''} \cdot rlb_y$$

Challenge 2

Interactions of Semantics

Shape of Traces for program p under $x + y$

Challenge 1

$$start_x \cdot \bar{\tau} \cdot start_y \cdot \bar{\tau'} \cdot rlb_y \cdot \overline{\tau''} \cdot rlb_x$$

Or

Safe Nesting

$$start_y \cdot \bar{\tau} \cdot start_x \cdot \bar{\tau'} \cdot rlb_x \cdot \overline{\tau''} \cdot rlb_y$$

Challenge 2