Security-aware Program Transformations

Massimo Bartoletti, Pierpaolo Degano, Gian Luigi Ferrari

Dipartimento di Informatica, Università di Pisa

Stack Inspection (1)

- access control mechanism based on the analysis of the execution stack (stack of method frames)
- a security policy maps each class to a *protection domain* (a named set of permissions)
- to check if a permission P is granted:

```
for each frame in the call stack (starting from top)
    if $P$ is not granted to the frame
        throw an AccessControlException
    if the frame is privileged
        return
```

Stack Inspection (2)

- *lazy* evaluation strategy: the one shown above
  - \- slow security checks
  - \- prevents from interprocedural optimizations
  - \+ no update of the security context

Stack Inspection (2)

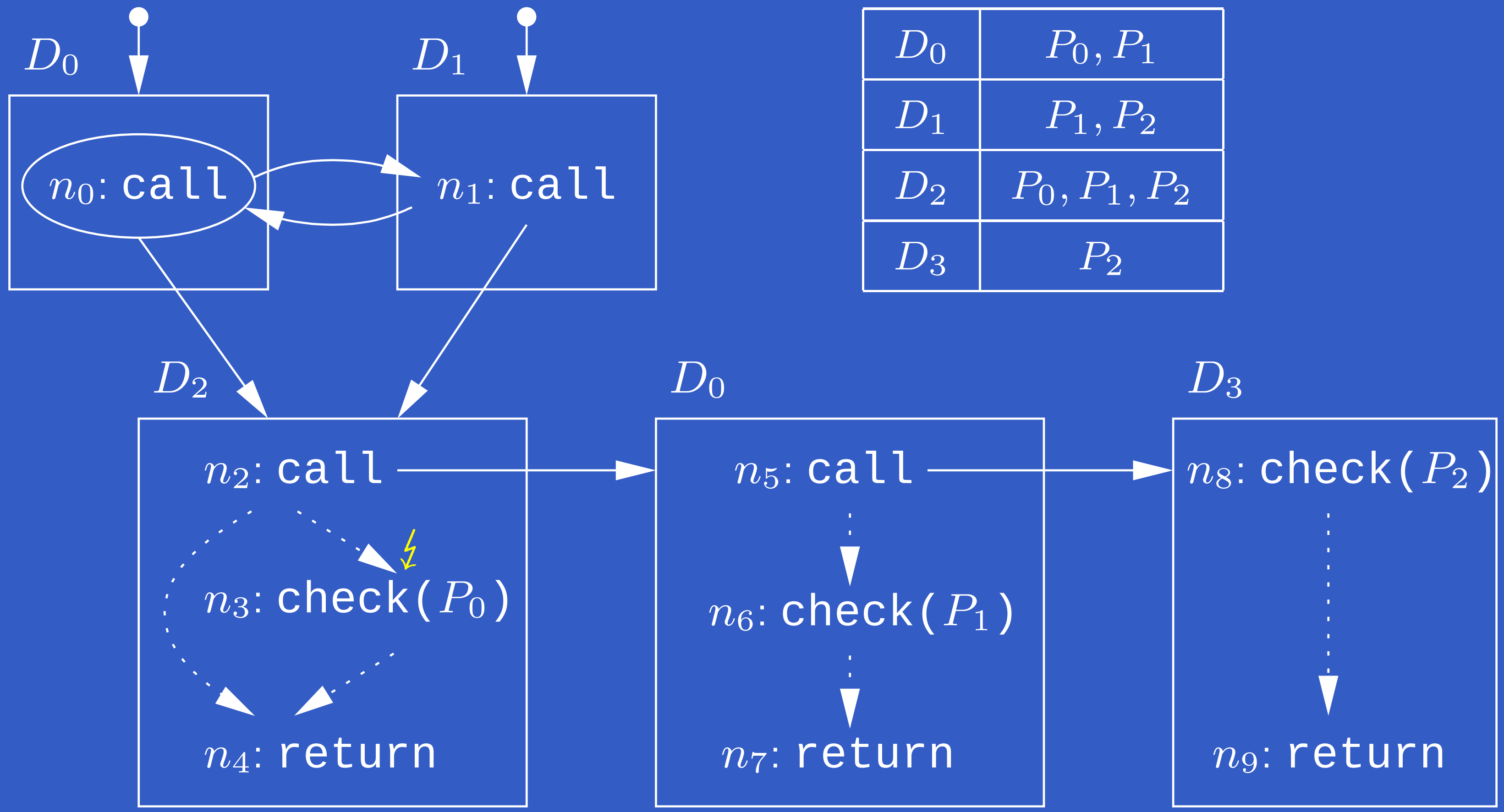
- *lazy* evaluation strategy: the one shown above
  - \- slow security checks
  - \- prevents from interprocedural optimizations
  - \+ no update of the security context
- *eager* evaluation strategy: the set of granted permissions is updated at each method call
  - \+ fast security checks
  - \+ allows for interprocedural optimizations (in combination with *security passing style*)
  - \- update of the security context

Program Model

Java Bytecode $\xrightarrow{CFA}$ Control Flow Graph

- control flow + security checks
- no data flow
- conditional construct ⟶ nondeterminism
- dynamic dispatching ⟶ nondeterminism

Program Model - syntax (1)



D_0

n_0: `call`

D_1

n_1: `call`

D_2

n_2: `call`

↯

n_3: `check(`P_0`)`

n_4: `return`

D_0

n_5: `call`

n_6: `check(`P_1`)`

n_7: `return`

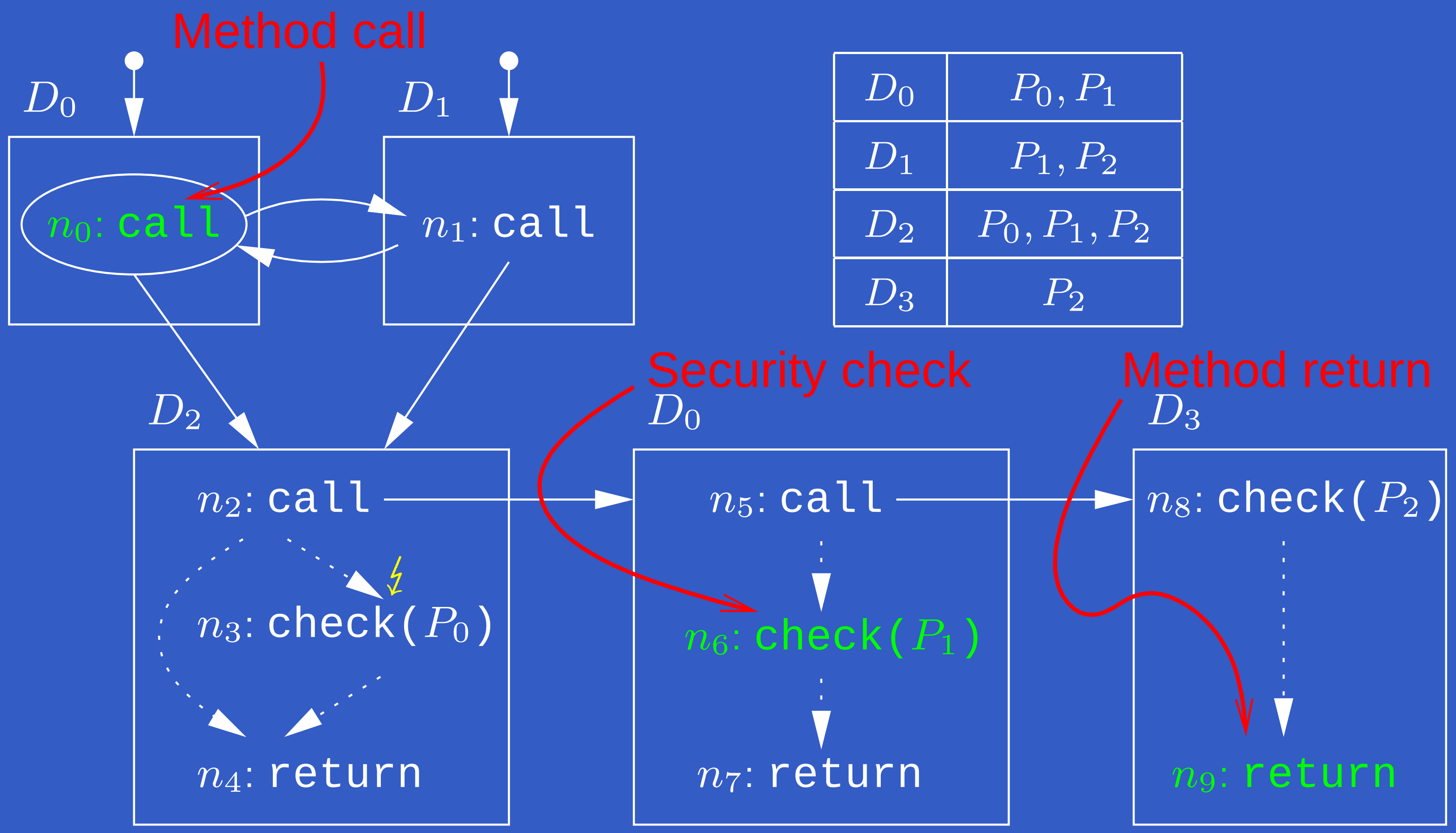
D_3

n_8: `check(`P_2`)`

n_9: `return`

D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

Program Model - syntax (2)



Method call

D_0

n_0: `call`

D_1

n_1: `call`

D_2

n_2: `call`

n_3: `check(`P_0`)`

n_4: `return`

Security check

D_0

n_5: `call`

n_6: `check(`P_1`)`

n_7: `return`

Method return

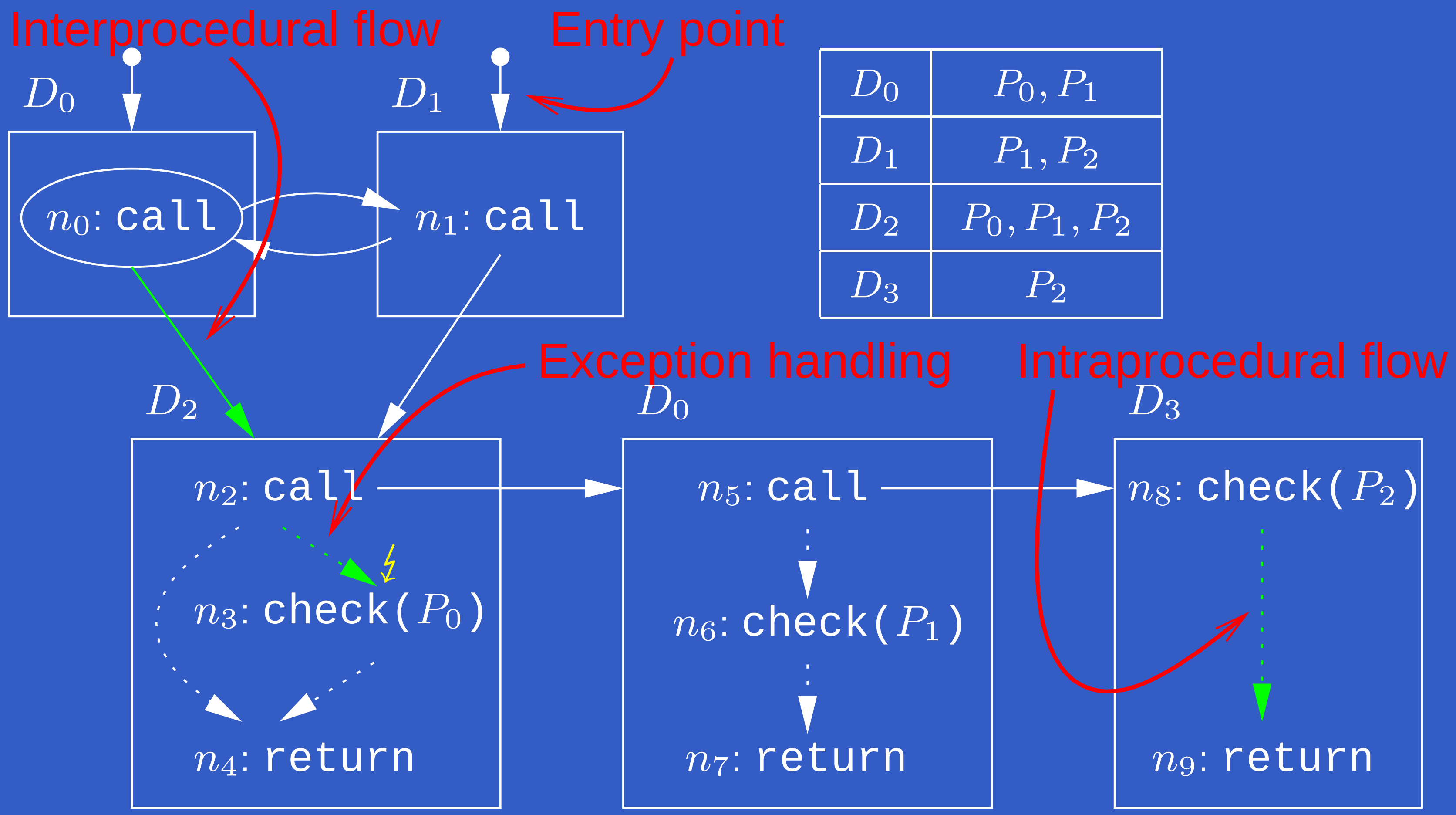
D_3

n_8: `check(`P_2`)`

n_9: `return`

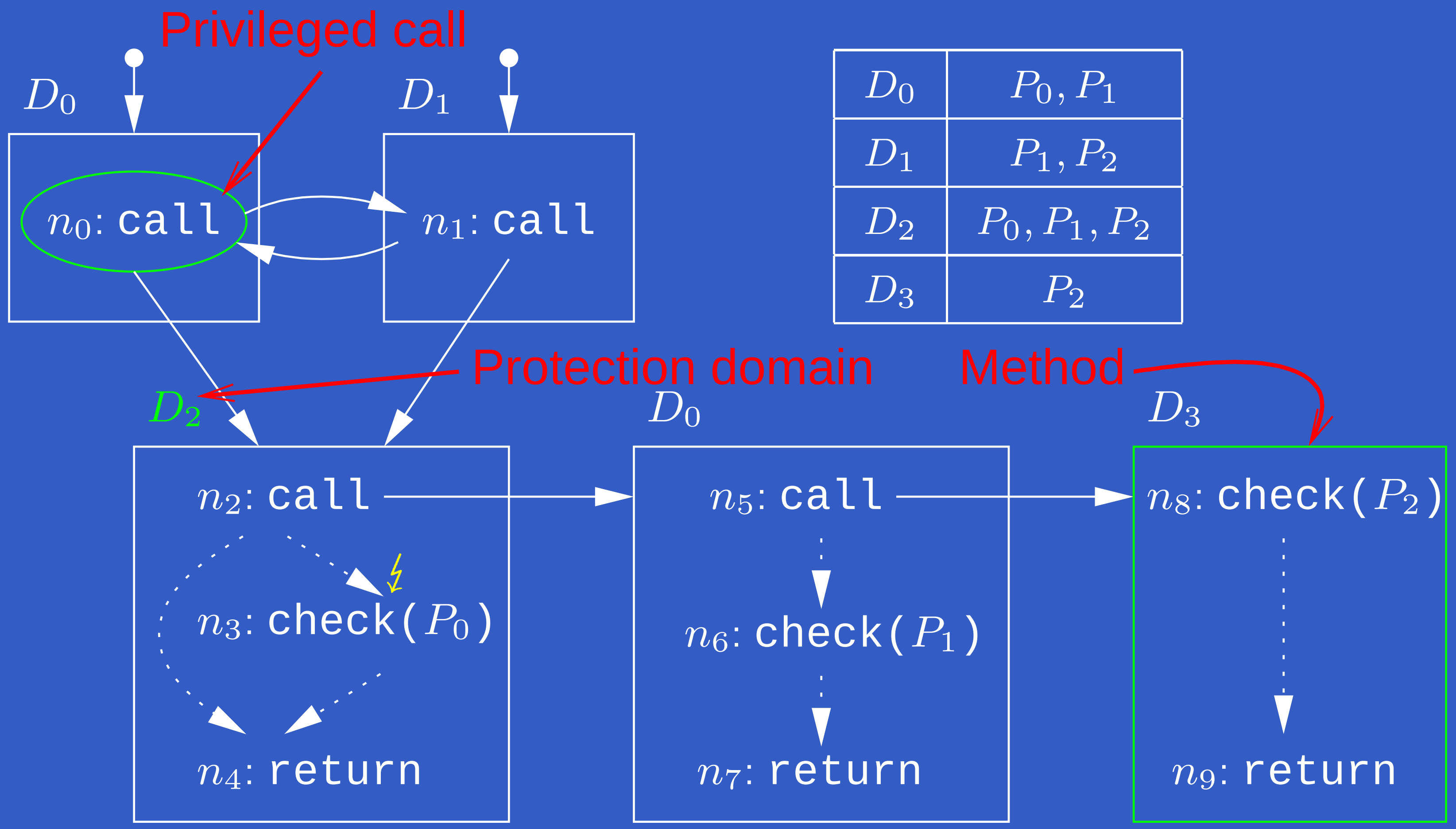
D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

Program Model - syntax (3)



D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

Program Model - syntax (4)

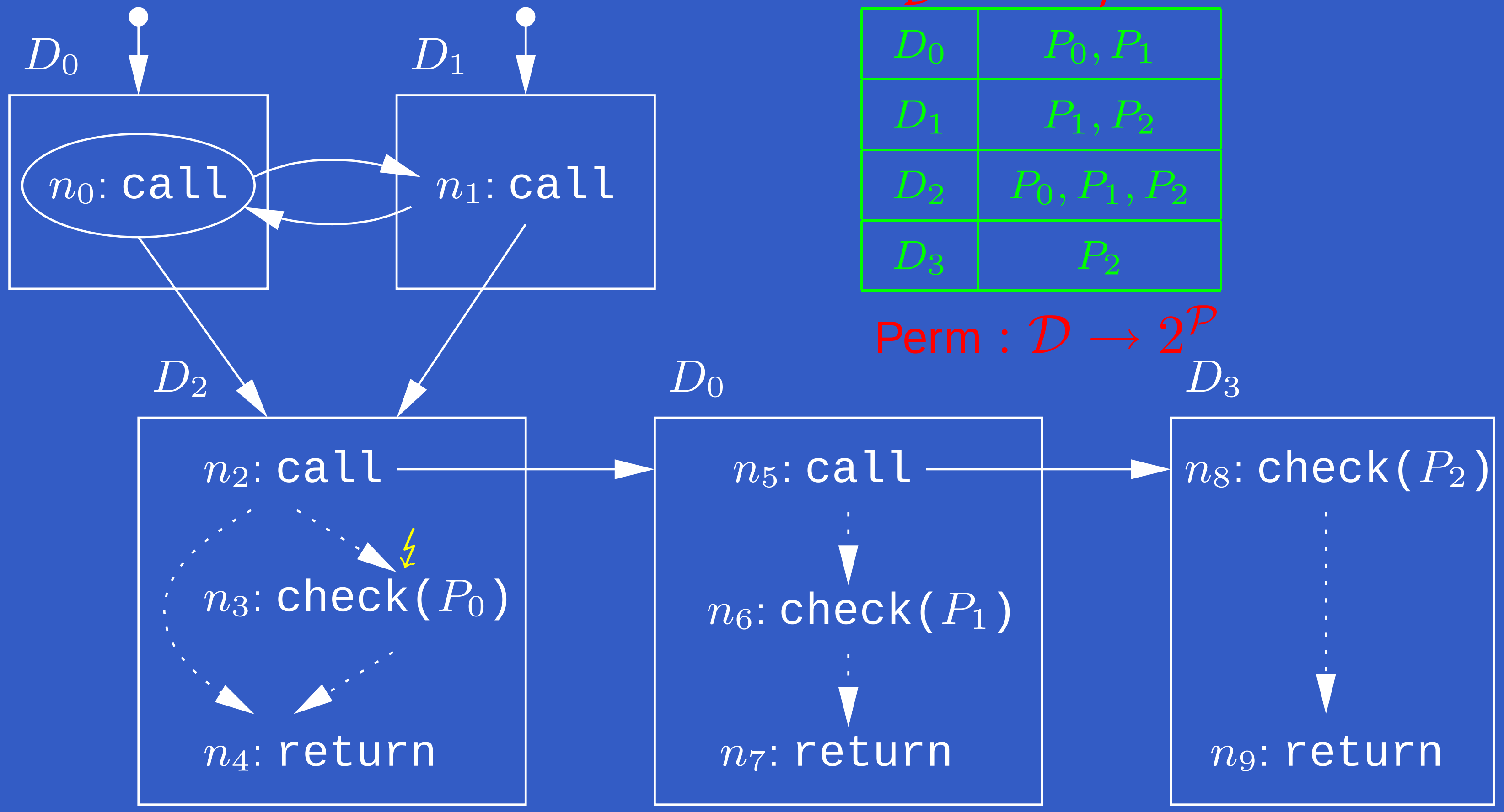


D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

Program Model - syntax (5)

$\mathcal{D}$	$\mathcal{P}$
D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

Perm : $\mathcal{D} \rightarrow 2^{\mathcal{P}}$



D_0

n_0: `call`

D_1

n_1: `call`

D_2

n_2: `call`

n_3: `check(`P_0`)`

n_4: `return`

D_0

n_5: `call`

n_6: `check(`P_1`)`

n_7: `return`

D_3

n_8: `check(`P_2`)`

n_9: `return`

Program Model - semantics (1)

- state = call stack + exception flag

  Example:

$$\langle [n_0, \ldots, n_k], \mathit{true} \rangle \quad = \quad [n_0, \ldots, n_k]\lightning$$

Program Model - semantics (1)

- state = call stack + exception flag

  Example:

$$\langle [n_0, \ldots, n_k], \mathit{true} \rangle \quad = \quad [n_0, \ldots, n_k]\lightning$$

top node

Program Model - semantics (1)

- state = call stack + exception flag

  Example:

$$\langle [n_0, \ldots, n_k], \mathit{true} \rangle \quad = \quad [n_0, \ldots, n_k]\lightning$$

an exception is active!

Program Model - semantics (1)

- state = call stack + exception flag

  Example:

$$\langle [n_0, \ldots, n_k], \mathit{true} \rangle \quad = \quad [n_0, \ldots, n_k]\lightning$$

- stack inspection $\sigma \vdash P$
- transition relation $\langle \sigma, x \rangle \ \triangleright \ \langle \sigma', x' \rangle$
- reachability relation $G \triangleright \langle \sigma, x \rangle$ when there is a trace from $\langle [], \mathit{false} \rangle$ to $\langle \sigma, x \rangle$

Program Model - semantics (2)

Stack inspection

$$\frac{}{[\,] \vdash P} \qquad [\vdash_1]$$

$$\frac{P \in \mathsf{Perm}(n) \quad \sigma \vdash P}{\sigma : n \vdash P} \qquad [\vdash_2]$$

$$\frac{P \in \mathsf{Perm}(n) \quad \mathsf{Priv}(n)}{\sigma : n \vdash P} \qquad [\vdash_3]$$

Program Model - semantics (3)

Method call/return

$$\frac{\bullet \longrightarrow n}{[\,] \rhd [n]} \qquad [\rhd_{entry}]$$

$$\frac{\ell(n) = \texttt{call} \quad n \longrightarrow n'}{\sigma : n \rhd \sigma : n : n'} \qquad [\rhd_{call}]$$

$$\frac{\ell(m) = \texttt{return} \quad n \dashrightarrow n'}{\sigma : n : m \rhd \sigma : n'} \qquad [\rhd_{ret}]$$

Program Model - semantics (4)

Security checks

$$\frac{\ell(n) = \mathsf{check}(P) \quad \sigma : n \vdash P \quad n \dashrightarrow n'}{\sigma : n \rhd \sigma : n'} \quad [\rhd_{pass}]$$

$$\frac{\ell(n) = \mathsf{check}(P) \quad \sigma : n \nvdash P}{\sigma : n \rhd \sigma : n\lightning} \quad [\rhd_{fail}]$$

Program Model - semantics (5)

Exception handling

$$\frac{n \dashrightarrow_{\lightning} n'}{\sigma : n\lightning \rhd \sigma : n'} \qquad [\rhd_{catch}]$$

$$\frac{n \not\dashrightarrow_{\lightning}}{\sigma : n\lightning \rhd \sigma\lightning} \qquad [\rhd_{propagate}]$$

The Trace Permissions Analysis (1)

- for each node n, it computes the *security contexts* $\tau(n)$ of all σ such that $G \triangleright \sigma$
- Security context of a state σ:

$$\Gamma([]) = \emptyset \qquad \Gamma(\sigma : n) = \begin{cases} \{\mathsf{Dom}(n)\} & \text{if } \mathsf{Priv}(n) \\ \Gamma(\sigma) \cup \{\mathsf{Dom}(n)\} & \text{otherwise} \end{cases}$$

- Set of permissions granted to a security context γ:

$$\Pi(\gamma) \quad = \quad \bigcap_{D \in \gamma} \mathsf{Perm}(D)$$

The Trace Permissions Analysis (2)

- For each state σ and permission P:

$$\sigma \vdash P \qquad \Longleftrightarrow \qquad P \in \Pi(\Gamma(\sigma))$$

- For each solution τ and state $\sigma : n$

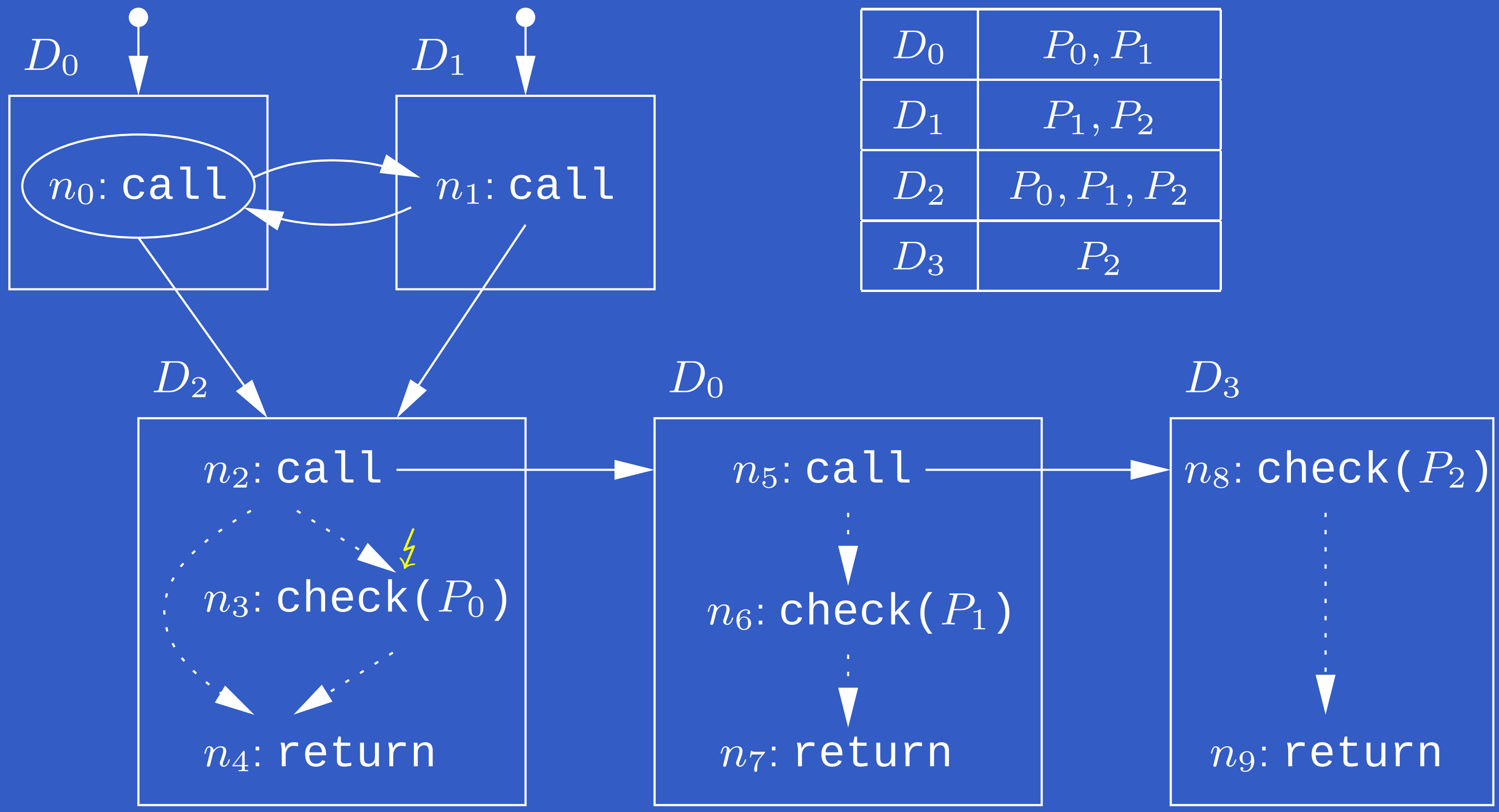
$$G \rhd \sigma : n \qquad \Longrightarrow \qquad \exists \gamma \in \tau(n).\ \gamma = \Gamma(\sigma : n)$$

- For the *minimal* solution τ and each state $\sigma : n$

$$\gamma \in \tau(n) \qquad \Longrightarrow \qquad \exists \sigma.\ G \rhd \sigma : n \ \ \wedge \ \ \gamma = \Gamma(\sigma : n)$$

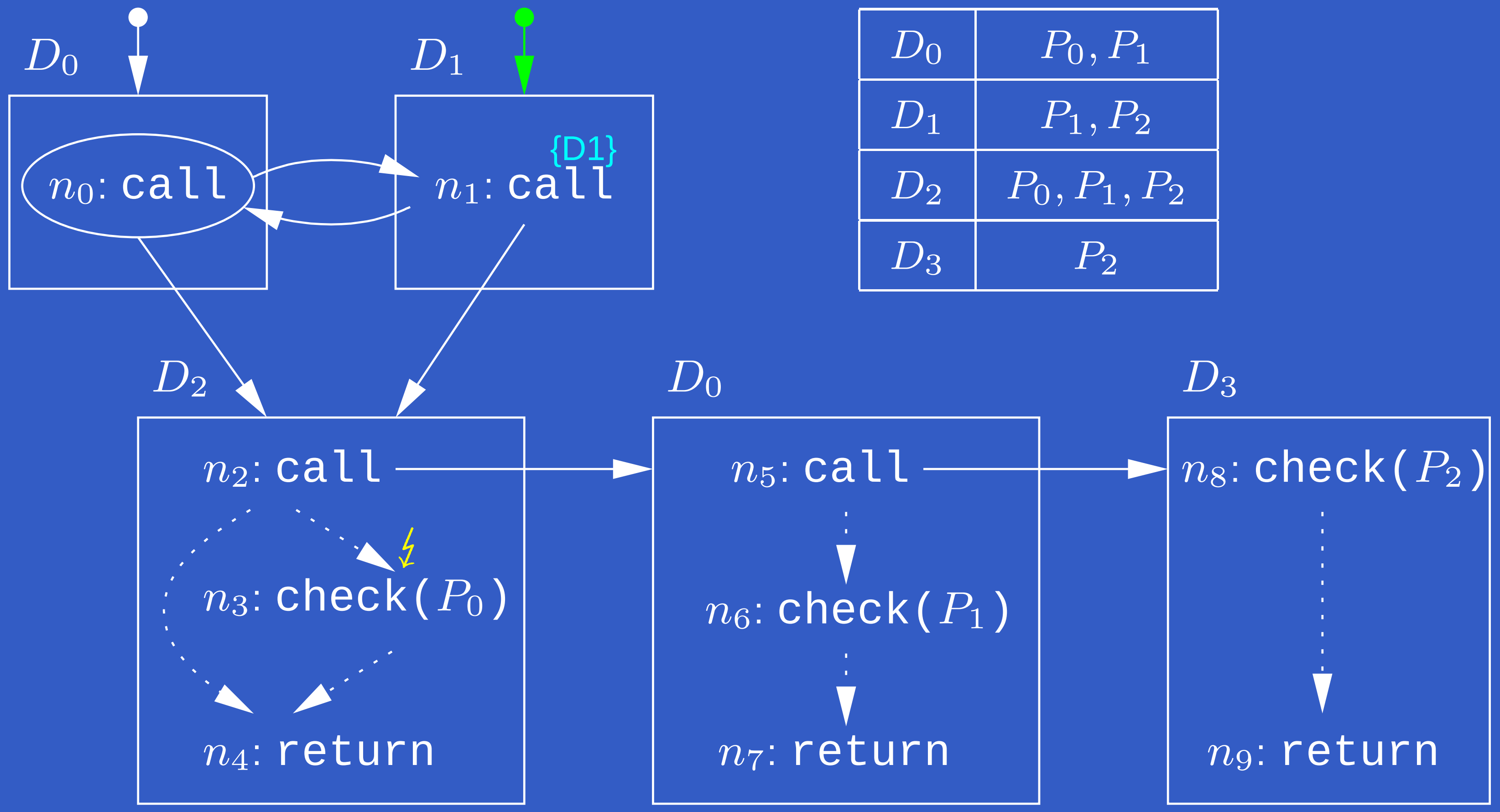
- the minimal solution is computed in $\mathcal{O}(N)$ by our worklist algorithm (N is the number of nodes)

The Trace Permissions Analysis (3)



D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

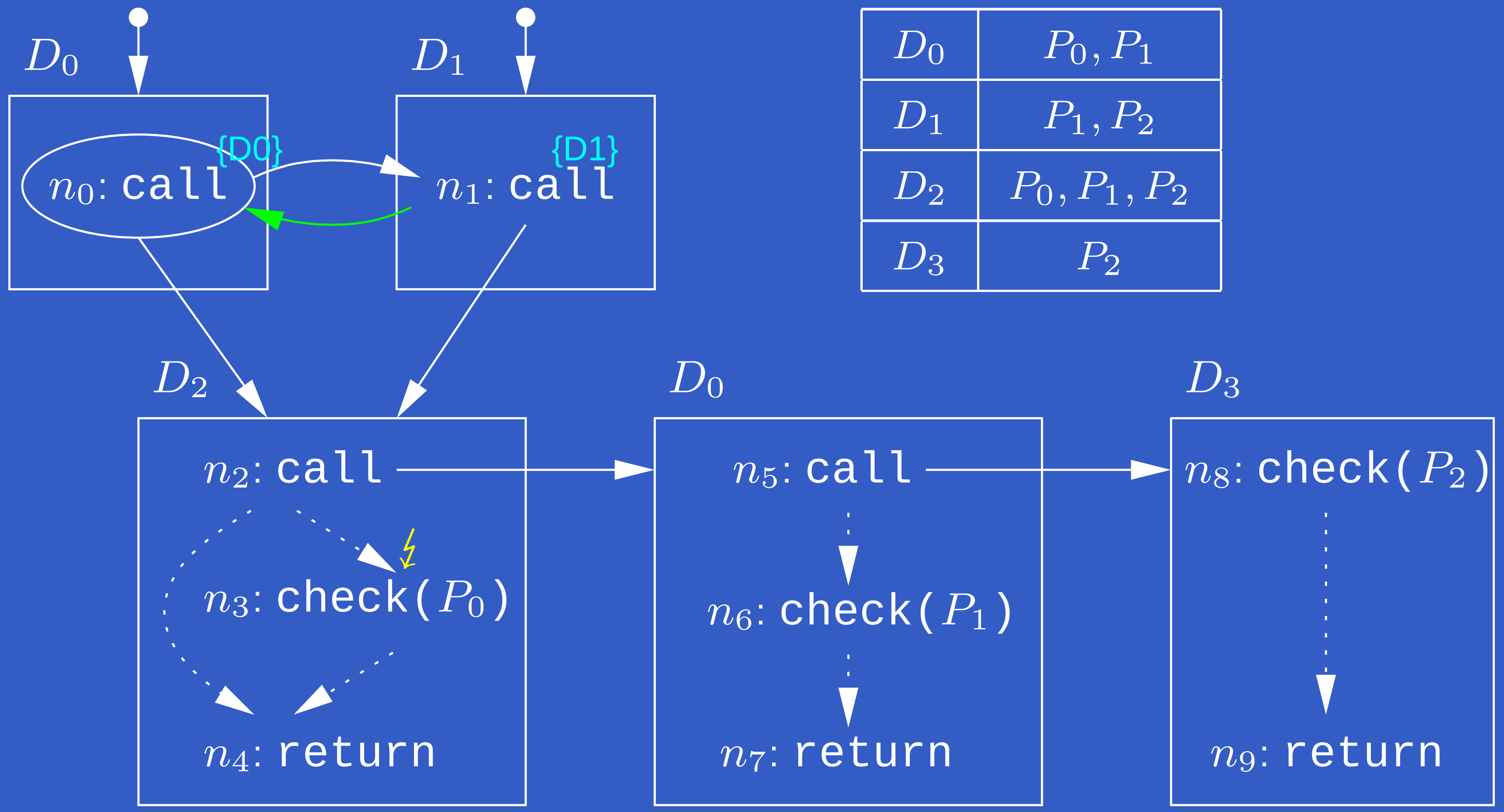
The Trace Permissions Analysis (3.1)



D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

$[] \triangleright [n_1]$

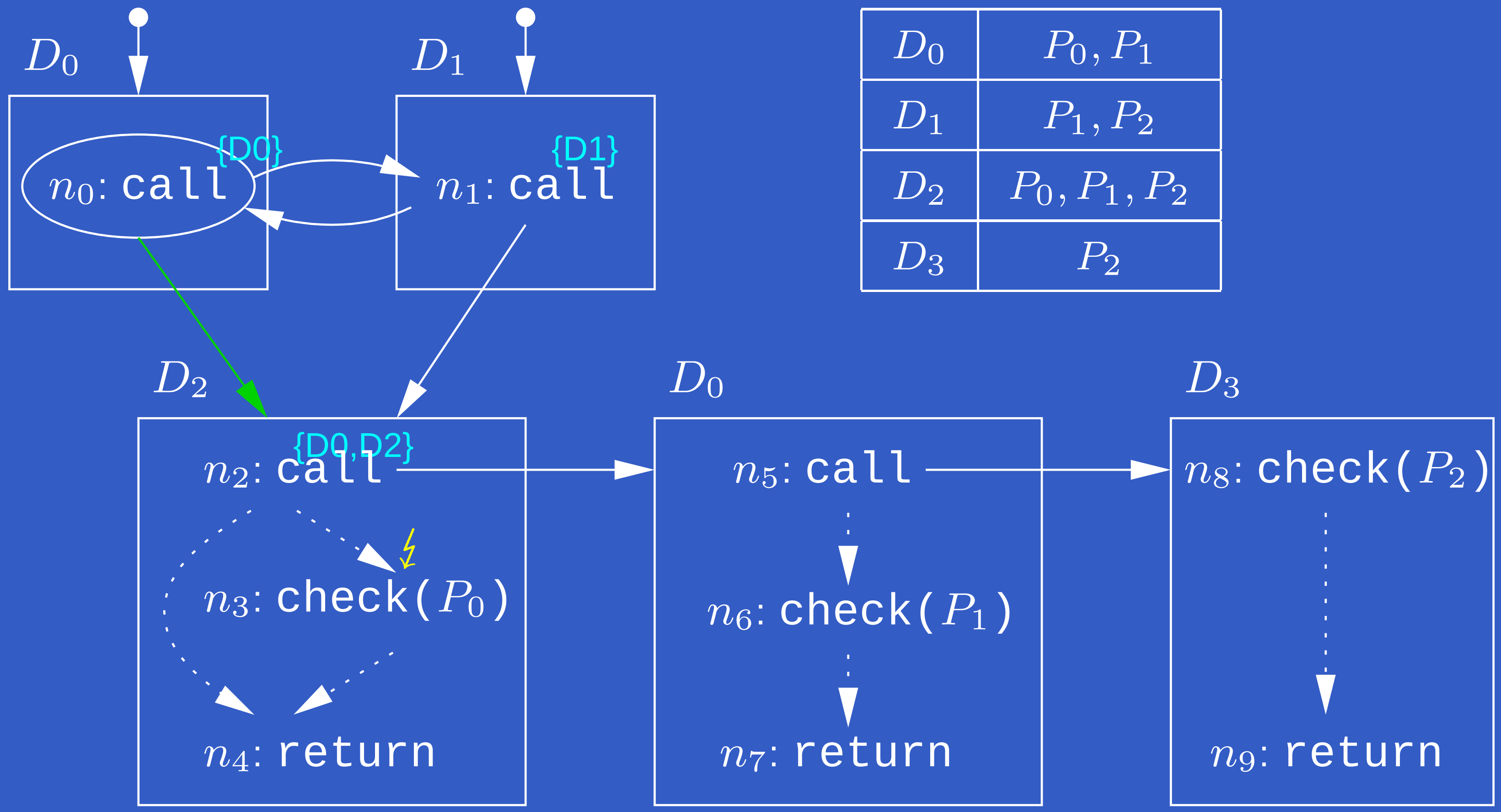
The Trace Permissions Analysis (3.2)



D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

$$[] \triangleright [n_1] \triangleright [n_1, n_0]$$

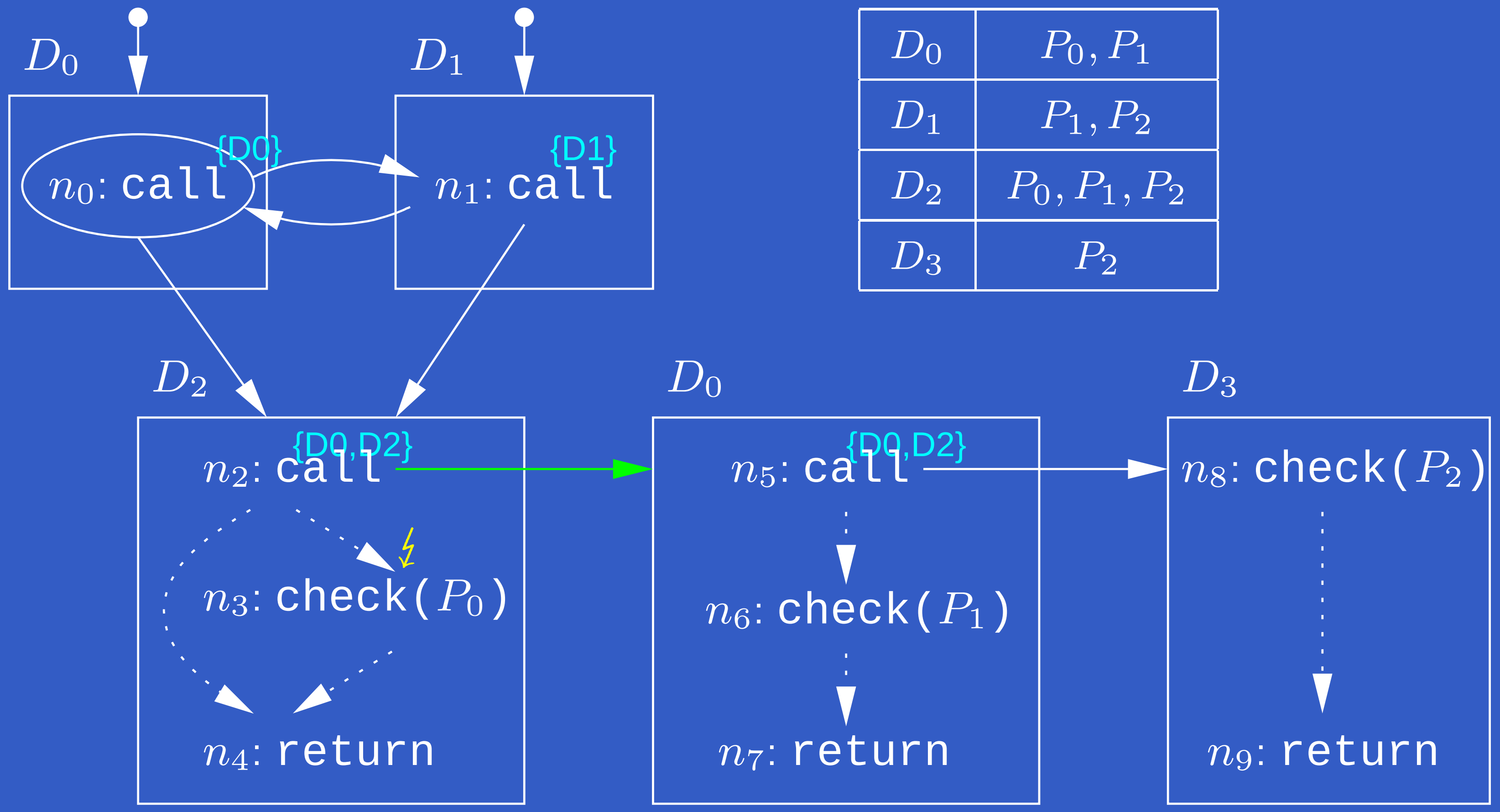
The Trace Permissions Analysis (3.3)



D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

$[] \triangleright [n_1] \triangleright [n_1, n_0] \triangleright [n_1, n_0, n_2]$

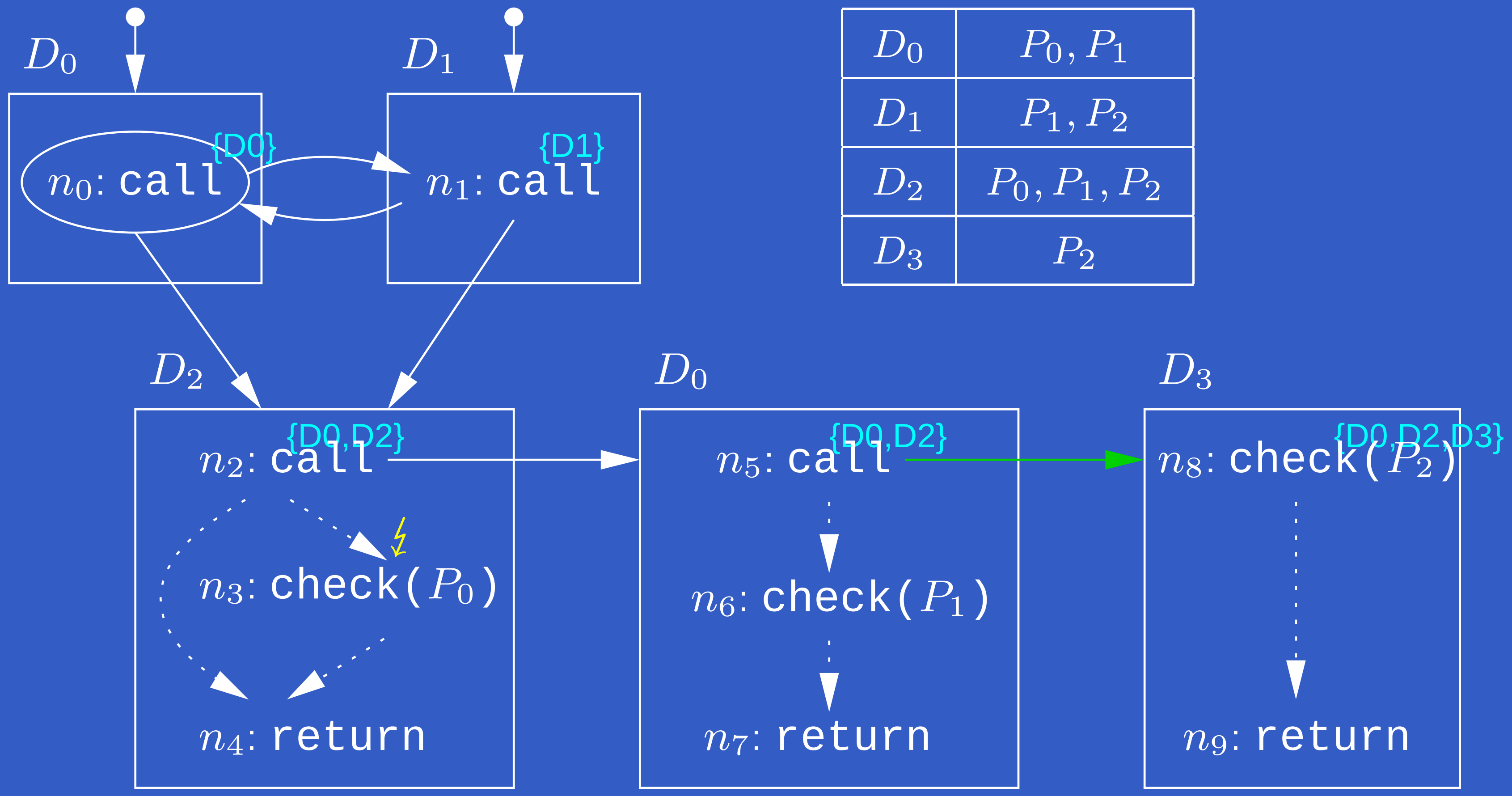
The Trace Permissions Analysis (3.4)



D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

$$[] \triangleright [n_1] \triangleright [n_1, n_0] \triangleright [n_1, n_0, n_2] \triangleright [n_1, n_0, n_2, n_5]$$

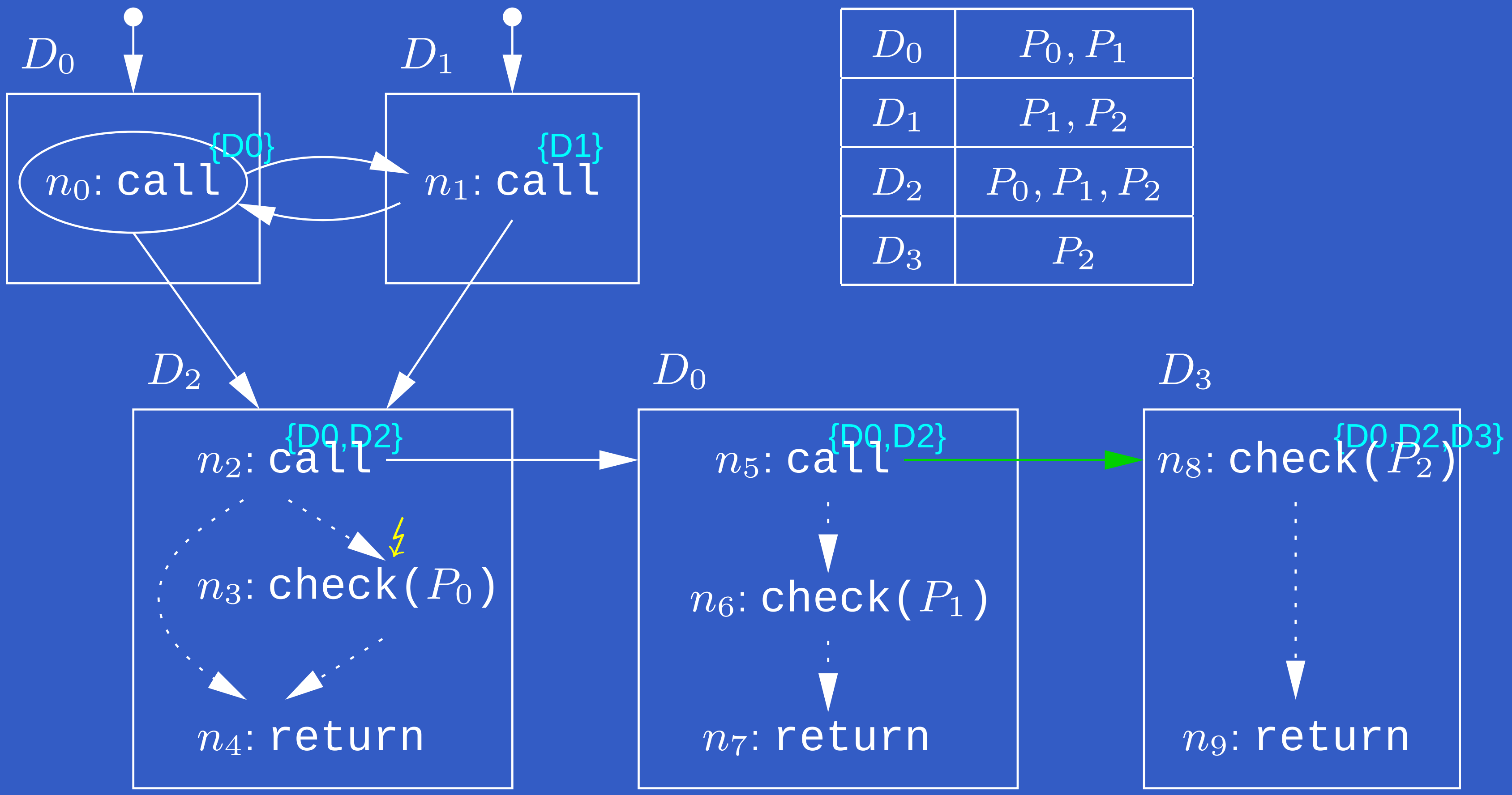
The Trace Permissions Analysis (3.5)



D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

$$\cdots \rhd [n_1, n_0, n_2, n_5] \rhd [n_1, n_0, n_2, n_5, n_8]$$

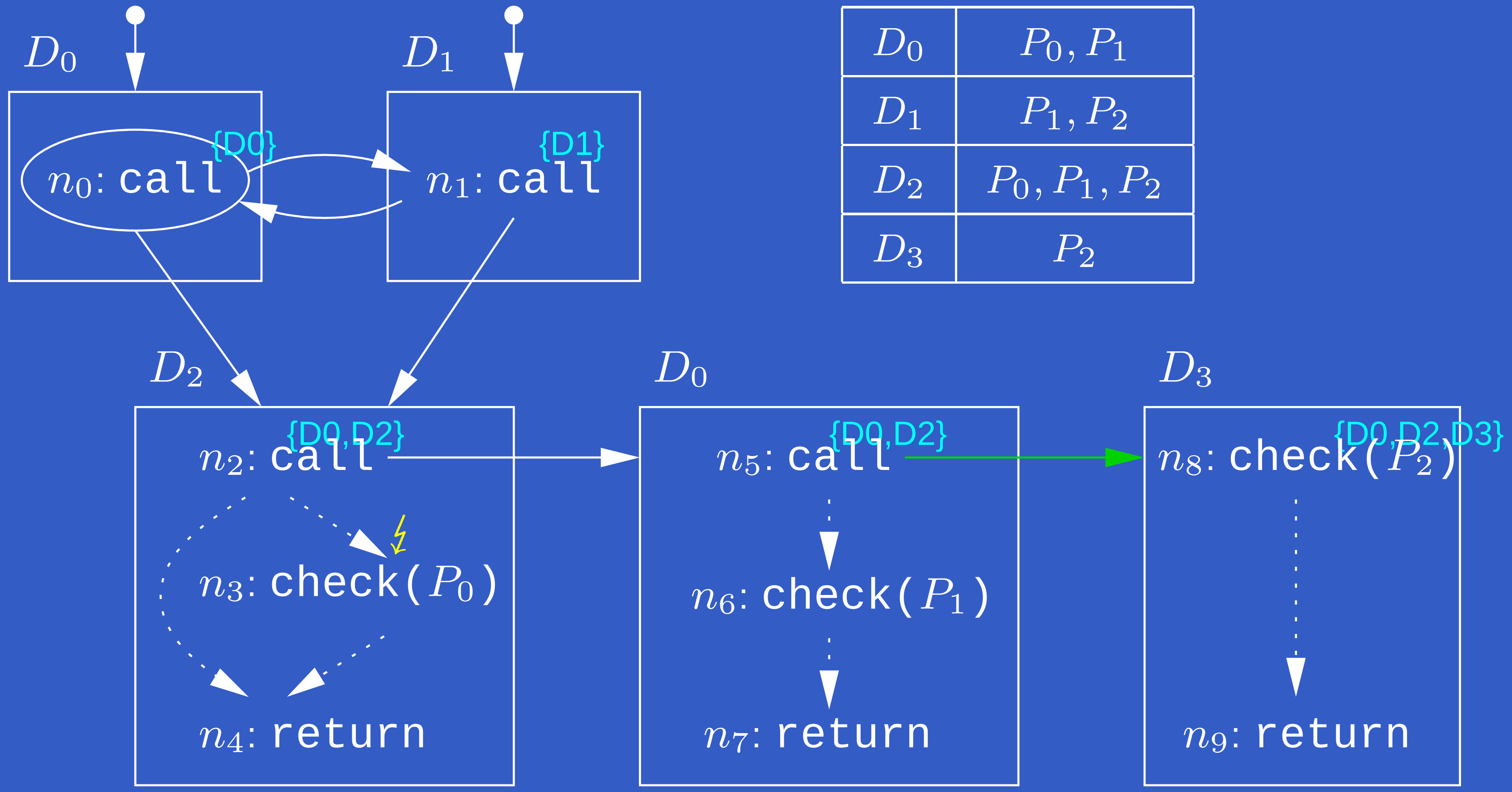
The Trace Permissions Analysis (3.5)



D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

$$\cdots \rhd [n_1, n_0, n_2, n_5] \rhd [n_1, n_0, n_2, n_5, n_8] \not\vdash P_2$$

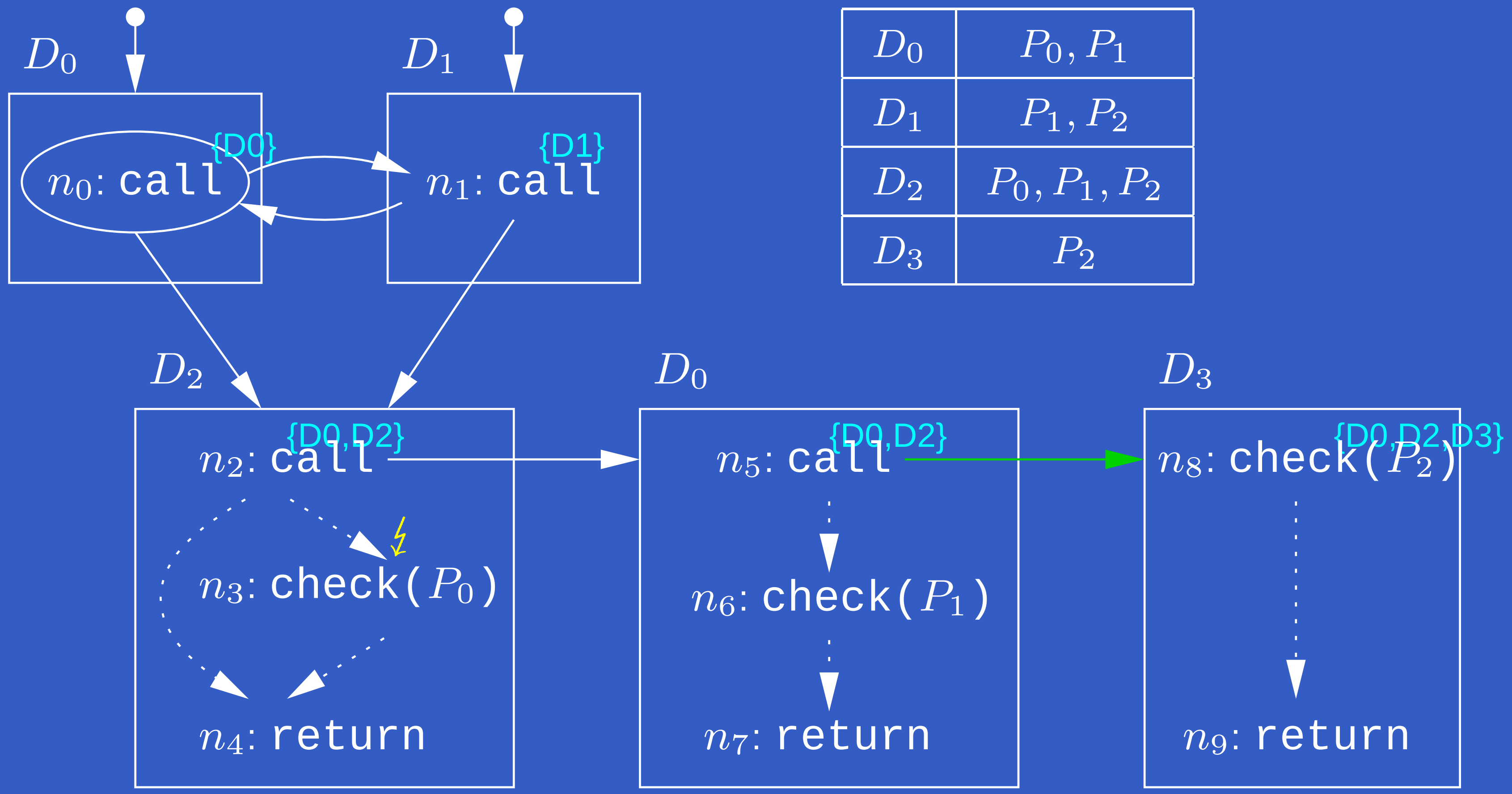
The Trace Permissions Analysis (3.5)



D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

$\cdots \triangleright [n_1, n_0, n_2, n_5, n_8]$ ↯

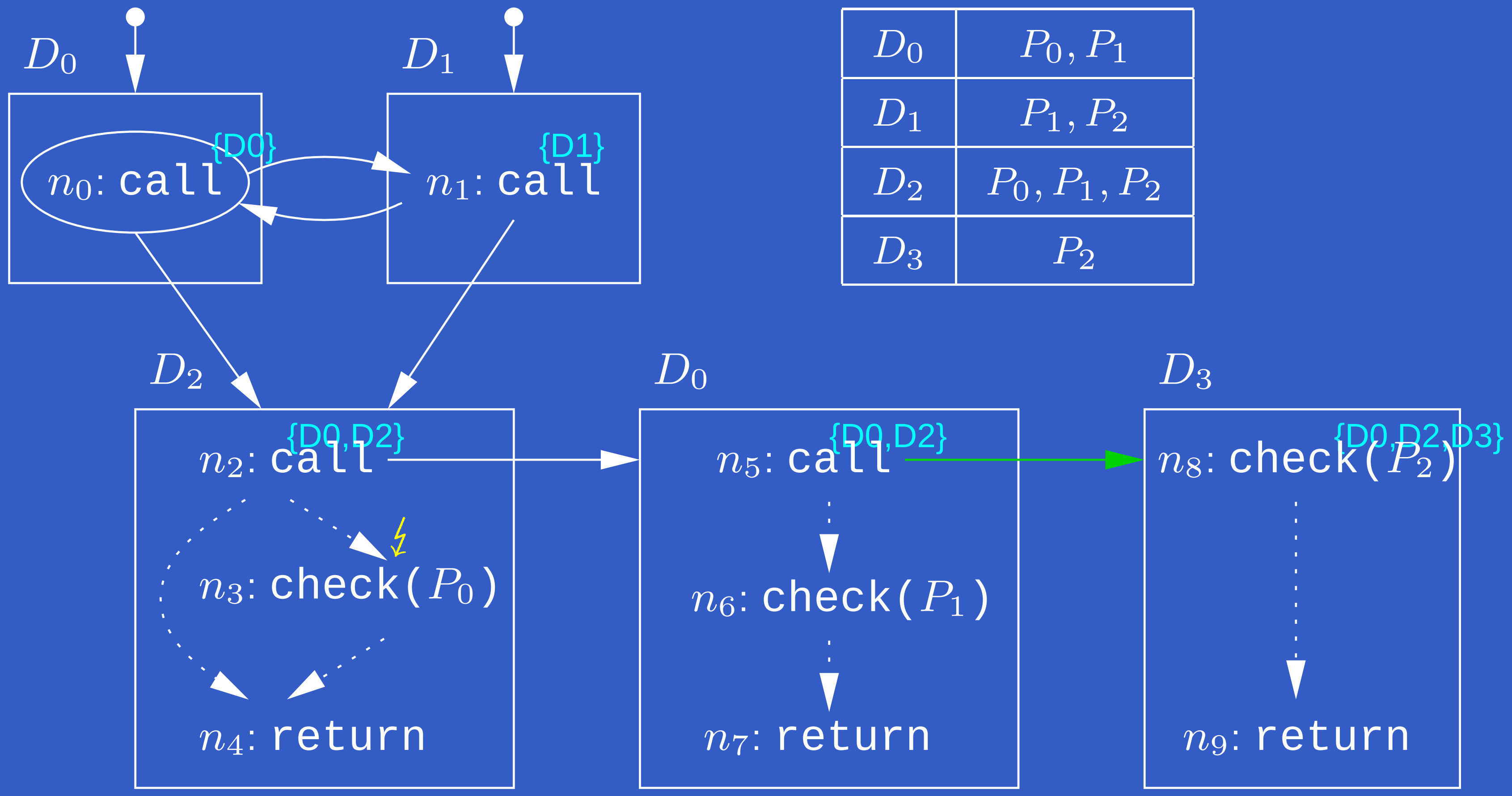
The Trace Permissions Analysis (3.5)



D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

$$\cdots \rhd [n_1, n_0, n_2, n_5, n_8]\lightning \rhd [n_1, n_0, n_2, n_5]\lightning$$

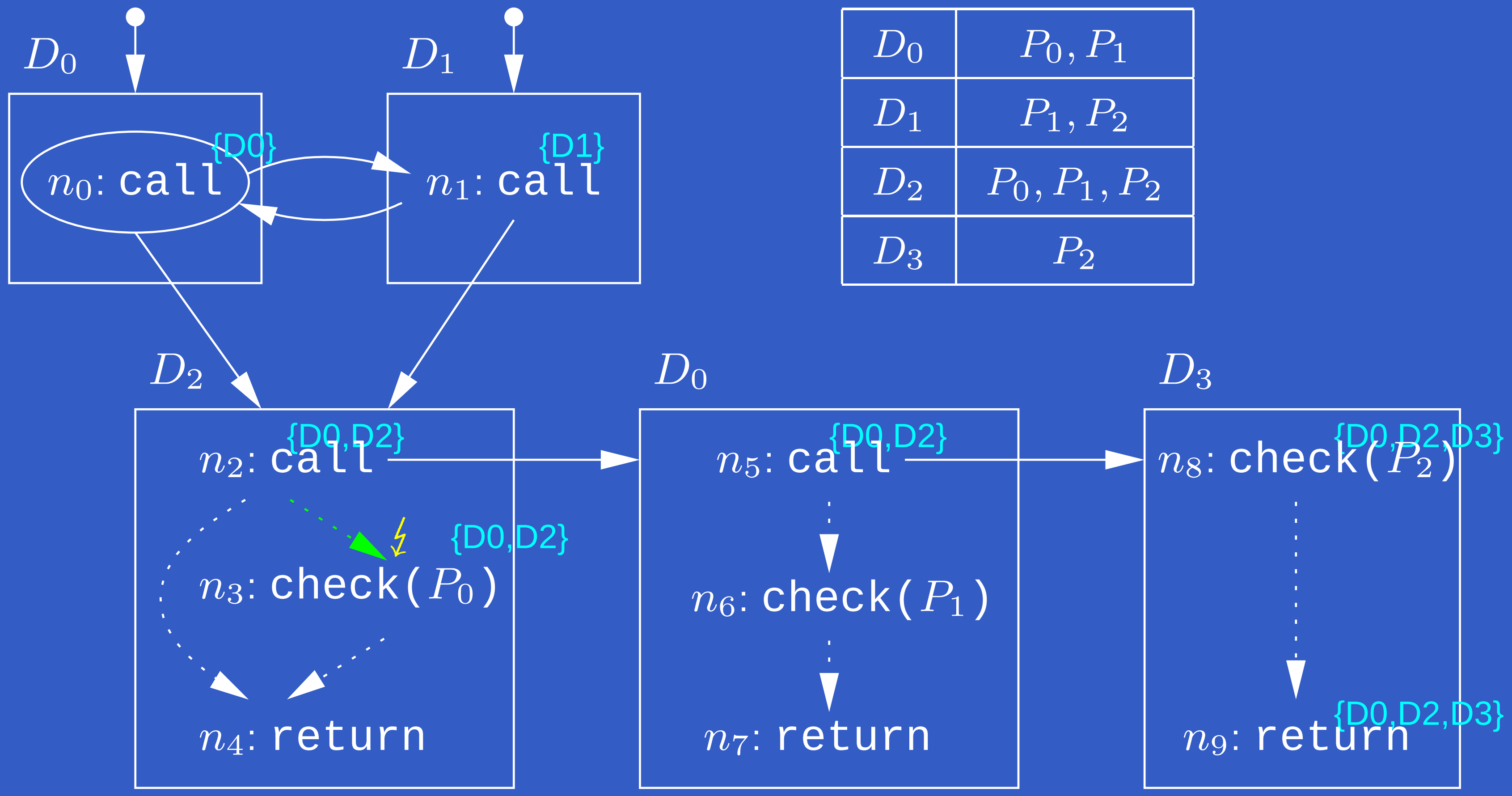
The Trace Permissions Analysis (3.5)



D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

$\cdots \triangleright [n_1, n_0, n_2, n_5, n_8]\lightning \triangleright [n_1, n_0, n_2, n_5]\lightning \triangleright [n_1, n_0, n_2]\lightning$

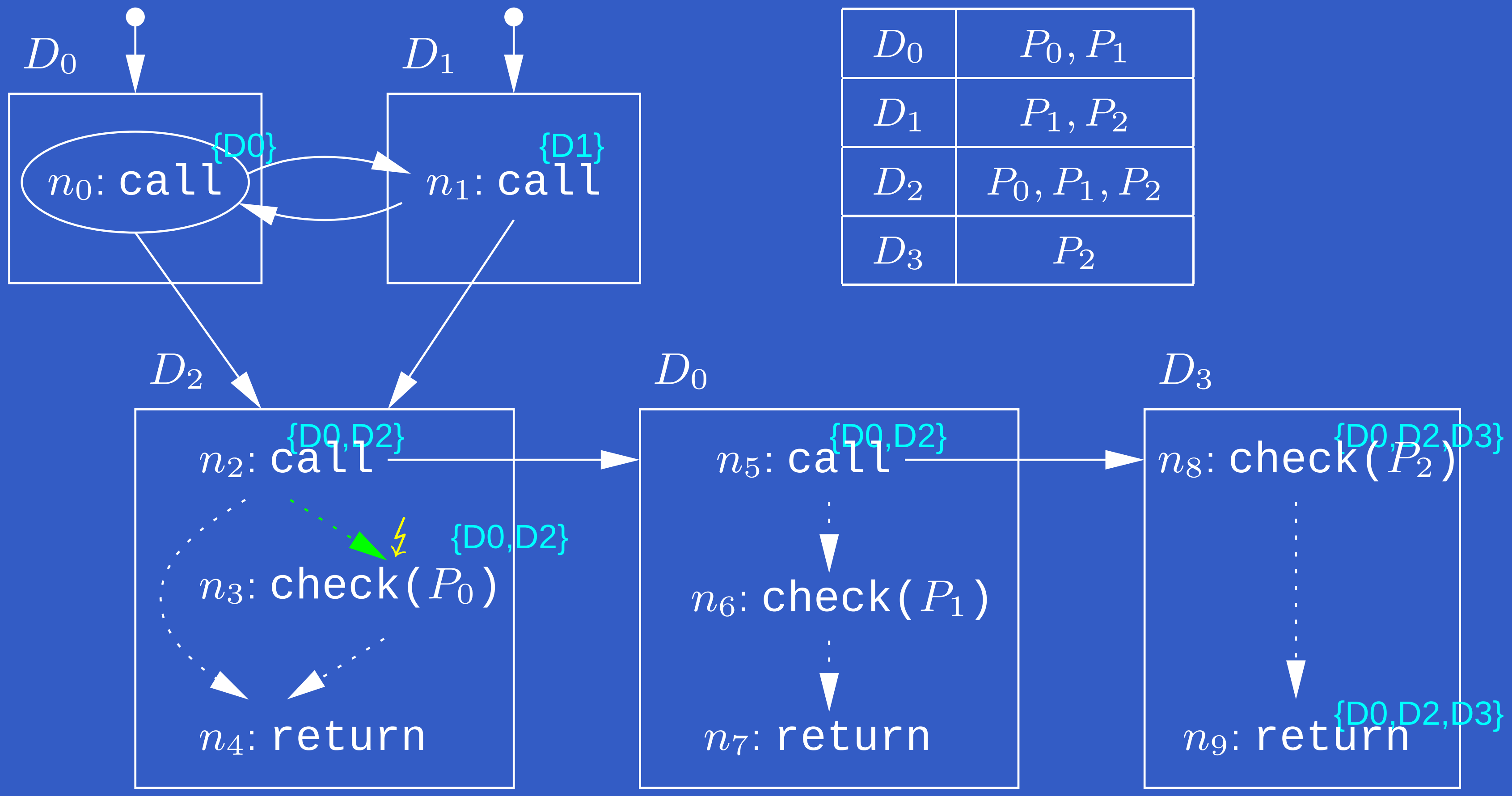
The Trace Permissions Analysis (3.6)



D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

$\cdots \triangleright [n_1, n_0, n_3]$

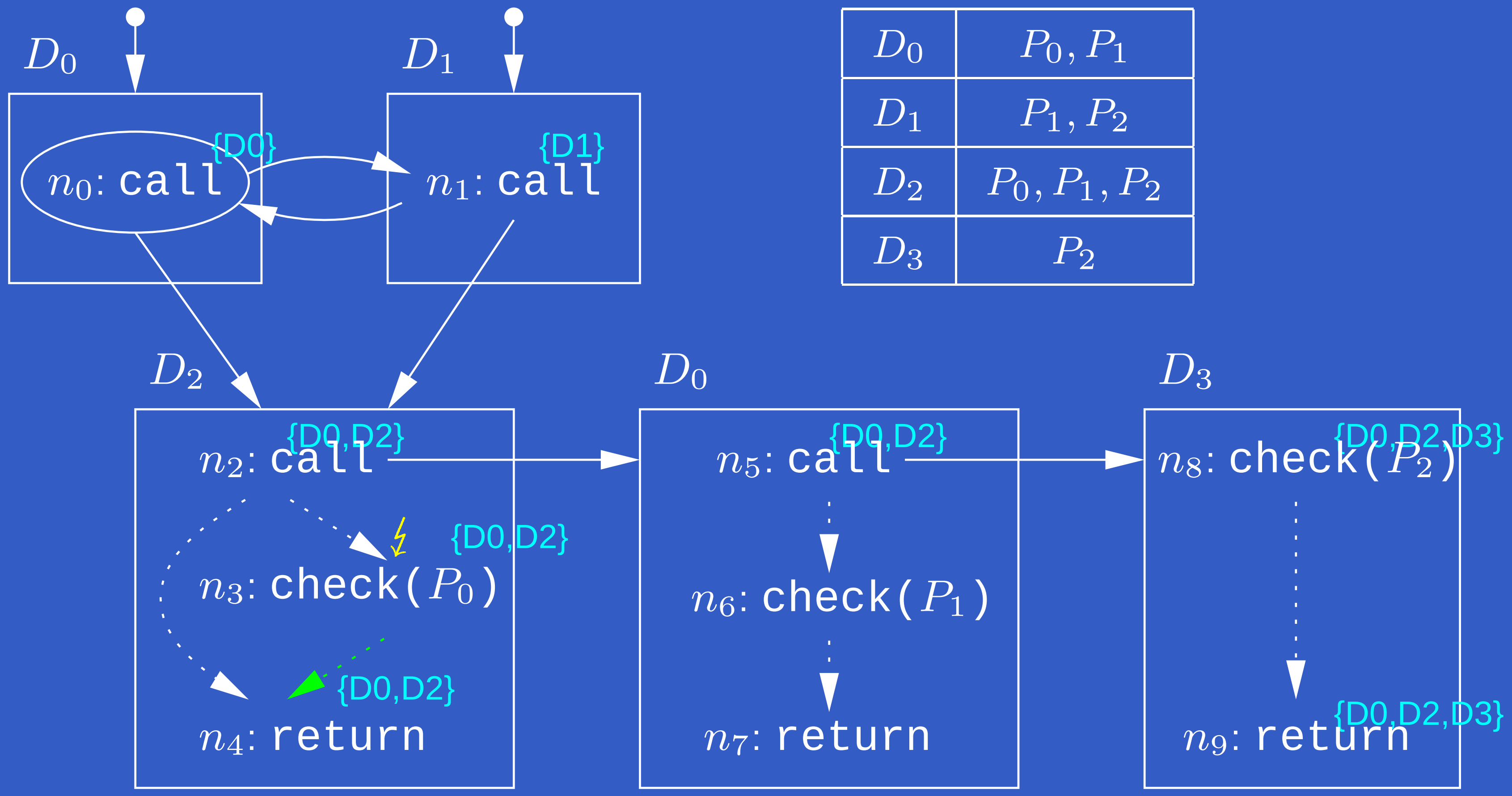
The Trace Permissions Analysis (3.6)



D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

$$\cdots \triangleright [n_1, n_0, n_3] \vdash P_0$$

The Trace Permissions Analysis (3.7)



D_0

n_0: `call` {D0}

D_1

n_1: `call` {D1}

D_2

n_2: `call` {D0,D2}

n_3: `check(`P_0`)` {D0,D2}

n_4: `return` {D0,D2}

D_0

n_5: `call` {D0,D2}

n_6: `check(`P_1`)`

n_7: `return`

D_3

n_8: `check(`P_2`)` {D0,D2,D3}

n_9: `return` {D0,D2,D3}

D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

$$\cdots \rhd [n_1, n_0, n_3] \rhd [n_1, n_0, n_4]$$

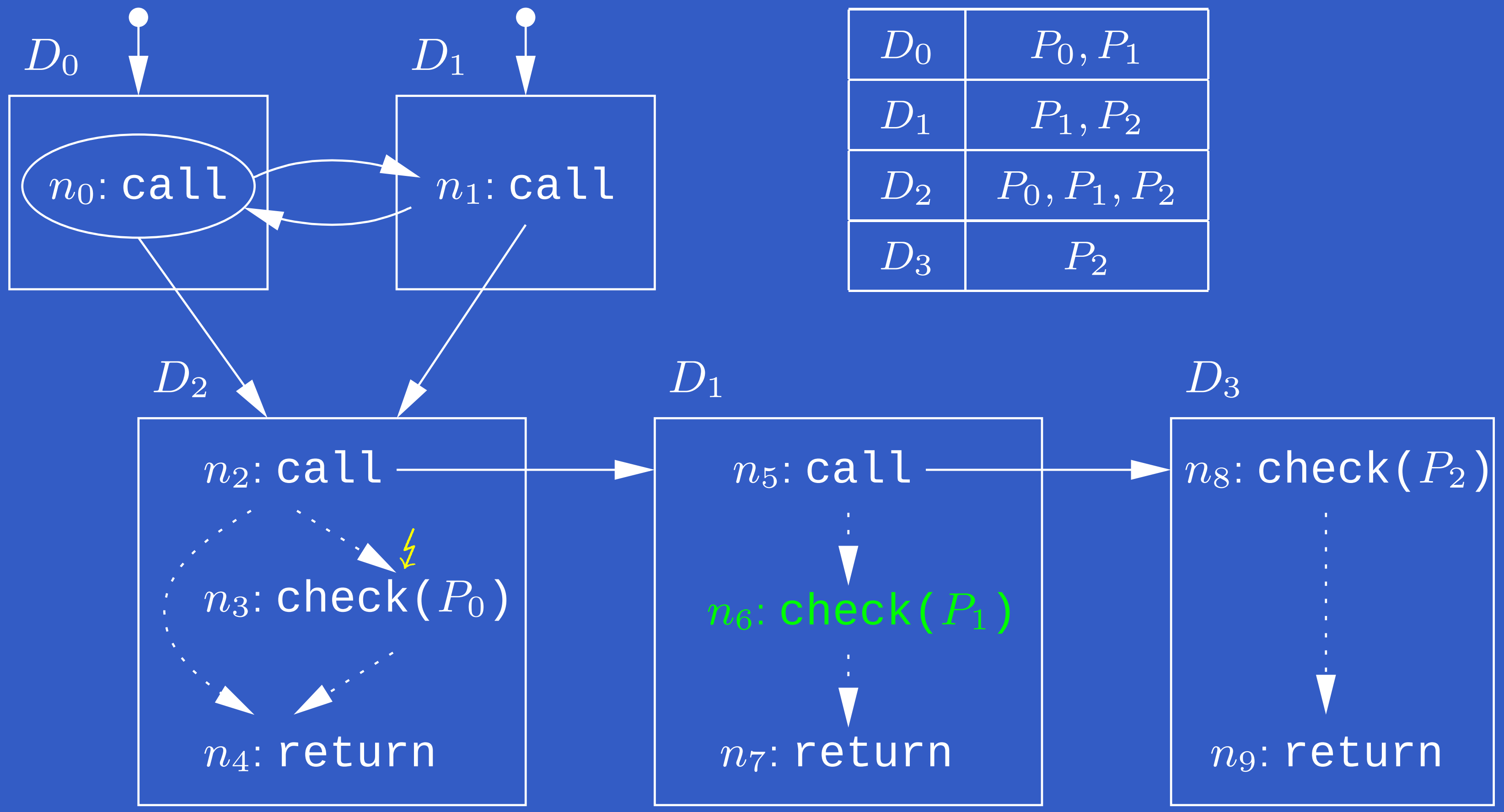
Optimizations

- Elimination of the redundant checks
- Dead code elimination
- Method inlining
- Tail call elimination
- Fast implementation of eager stack inspection

Optimizations

- Elimination of the redundant checks
- Dead code elimination
- Method inlining
- Tail call elimination
- Fast implementation of eager stack inspection

Redundant Checks Elimination (1)



D_0 D_1

n_0: `call` n_1: `call`

D_2 D_1 D_3

n_2: `call` n_5: `call` n_8: `check(`P_2`)`

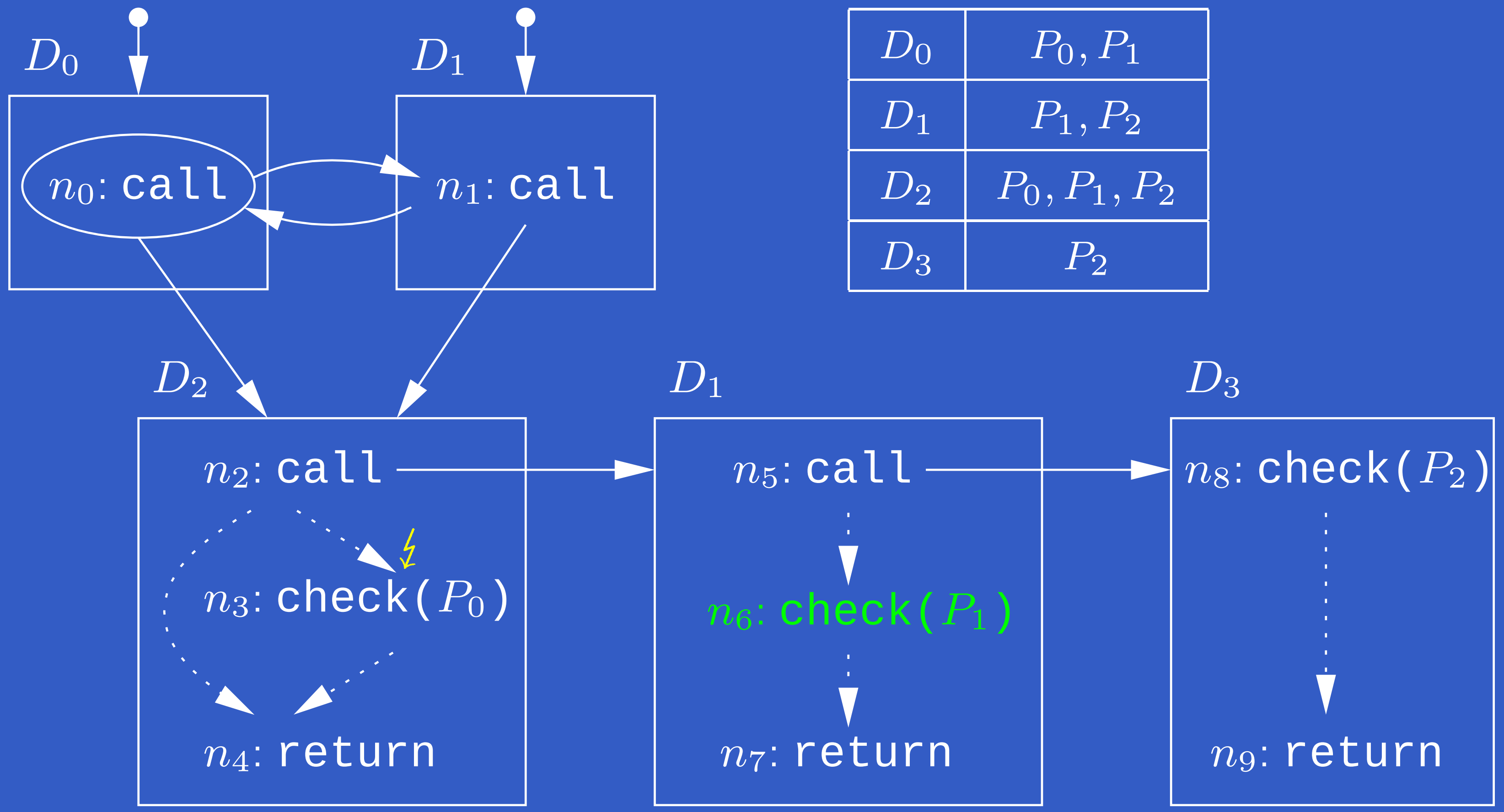
↯

n_3: `check(`P_0`)` n_6: `check(`P_1`)`

n_4: `return` n_7: `return` n_9: `return`

D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

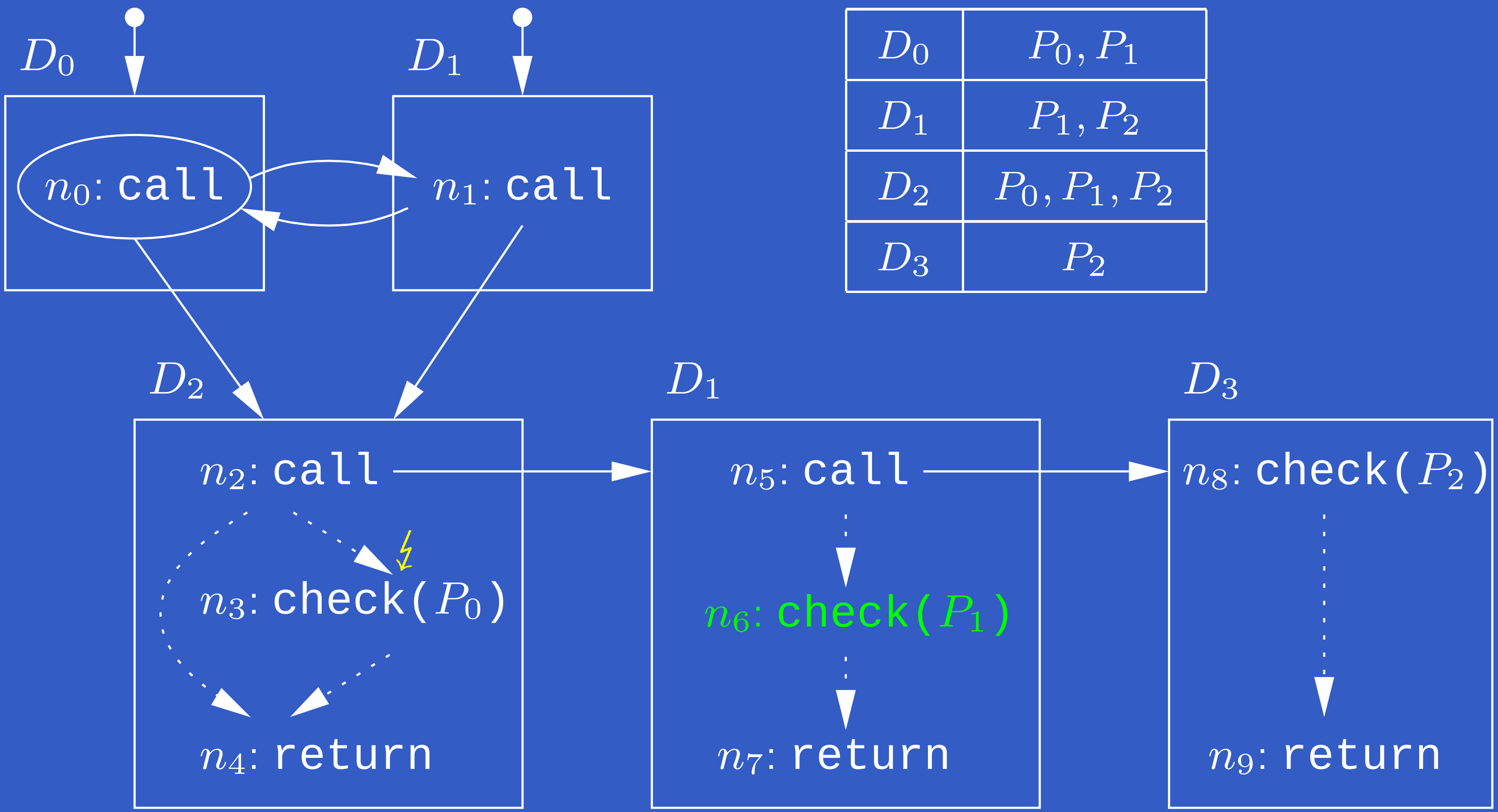
Redundant Checks Elimination (1)



D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

$$\tau(n_6) = \{\{D_1, D_2\}\}$$

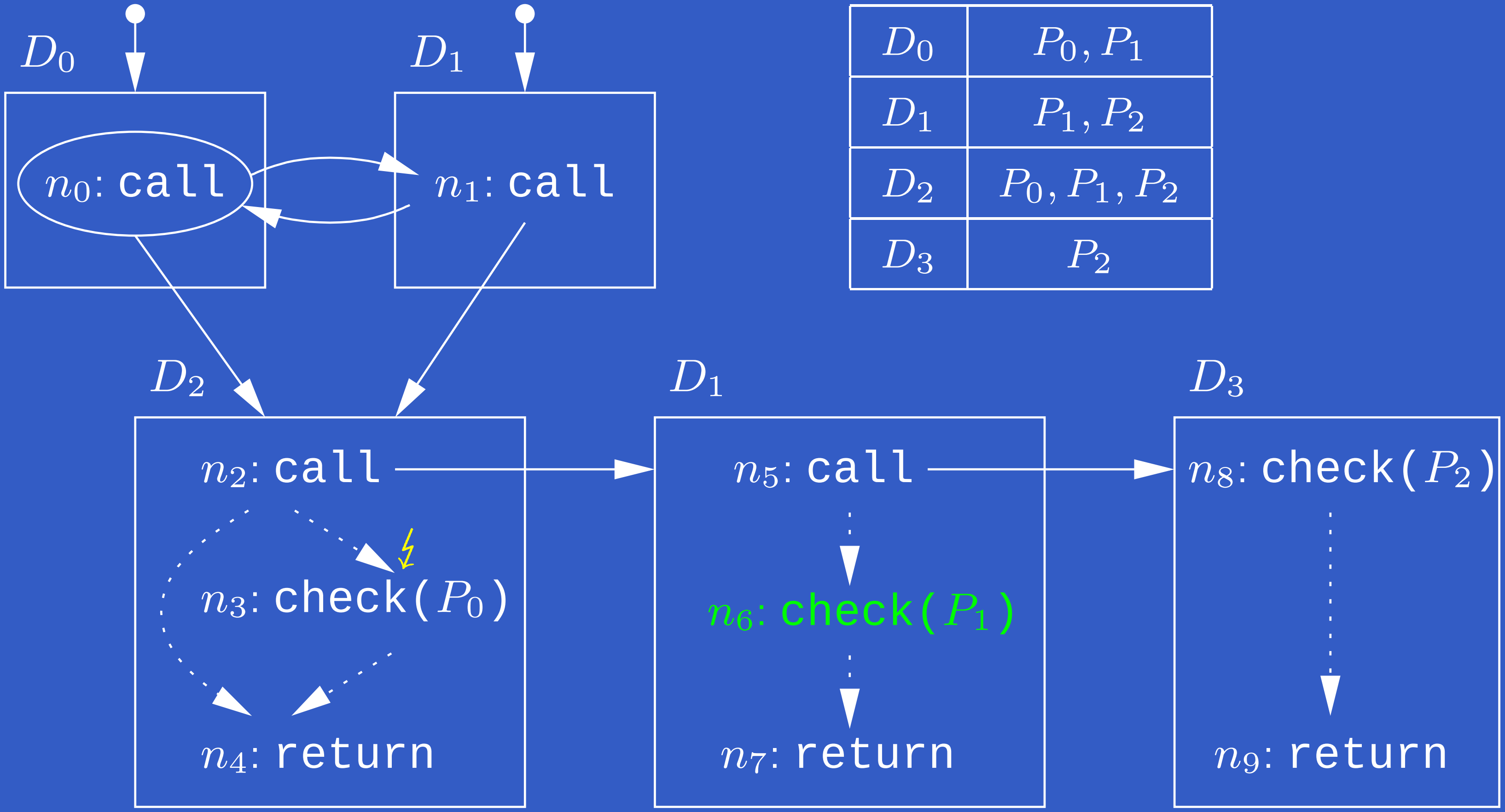
Redundant Checks Elimination (1)



D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

$$\Pi(\{D_1, D_2\}) = \text{Perm}(D_1) \cap \text{Perm}(D_2) = \{P_1, P_2\}$$

Redundant Checks Elimination (1)



D_0

n_0: `call`

D_1

n_1: `call`

D_2

n_2: `call`

n_3: `check(`P_0`)` ↯

n_4: `return`

D_1

n_5: `call`

n_6: `check(`P_1`)`

n_7: `return`

D_3

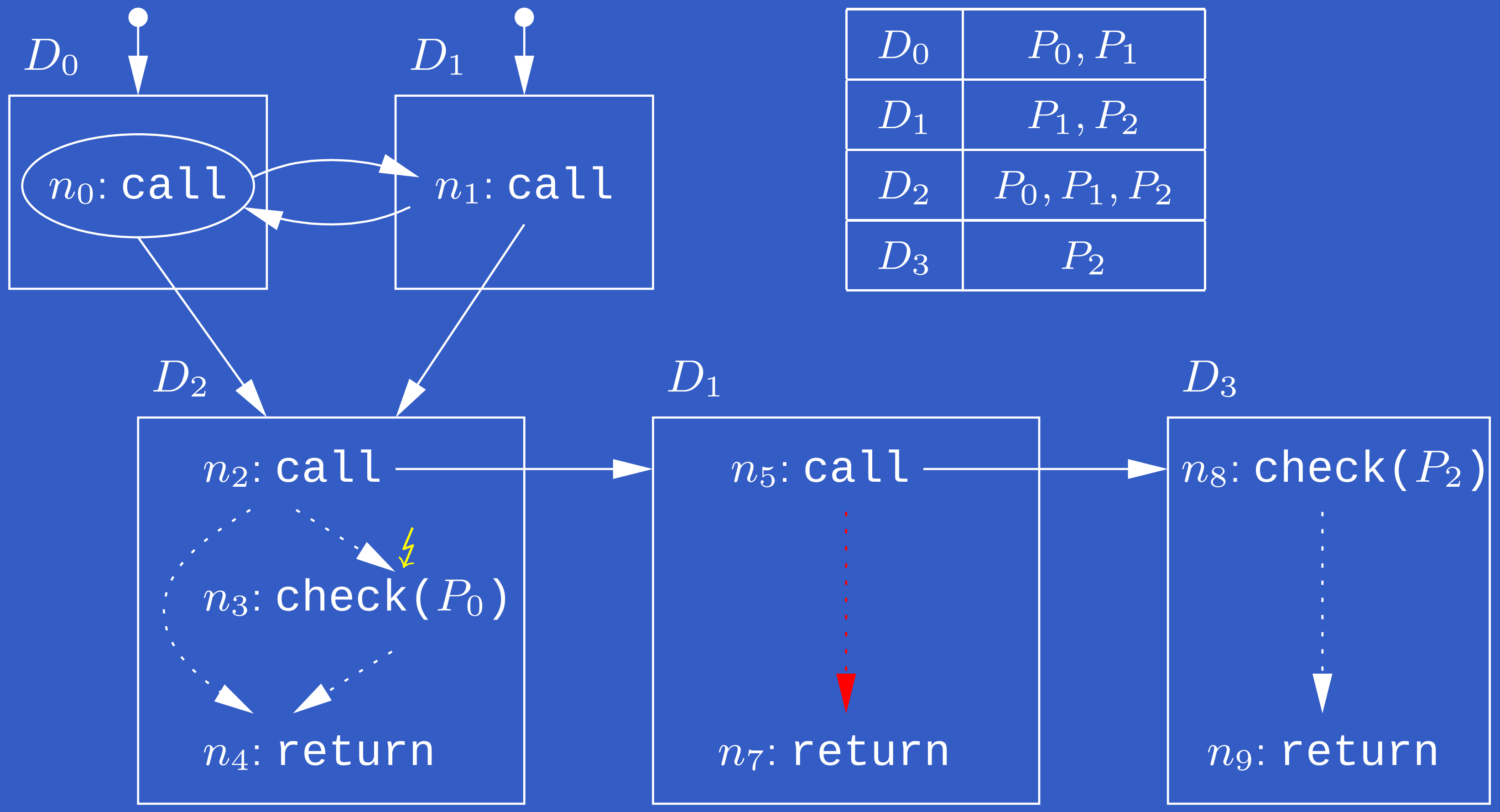
n_8: `check(`P_2`)`

n_9: `return`

D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

$P_1 \in \Pi(\{D_1, D_2\}) \implies n_6$ is redundant

Redundant Checks Elimination (1)



D_0	P_0, P_1
D_1	P_1, P_2
D_2	P_0, P_1, P_2
D_3	P_2

Redundant Checks Elimination (2)

- a check node n for permission P is *redundant* when:

$$\forall \sigma \in N^*. \quad G \triangleright \sigma : n \implies \sigma : n \vdash P$$

Redundant Checks Elimination (2)

- a check node n for permission P is *redundant* when:

$$\forall \sigma \in N^*. \quad G \rhd \sigma : n \implies \sigma : n \vdash P$$

- Let $\tau \models TP^{=}(G, \textit{Perm})$. For each check node n, define:

$$\Pi(n) \quad = \quad \bigcap \{ \Pi(\gamma) \mid \gamma \in \tau(n) \}$$

$\Pi(n)$ is the set of permissions (statically) granted to n.

Redundant Checks Elimination (2)

- a check node n for permission P is *redundant* when:

$$\forall \sigma \in N^*. \quad G \triangleright \sigma : n \implies \sigma : n \vdash P$$

- Let $\tau \models TP^{=}(G, \textit{Perm})$. For each check node n, define:

$$\Pi(n) \quad = \quad \bigcap \{\, \Pi(\gamma) \mid \gamma \in \tau(n) \,\}$$

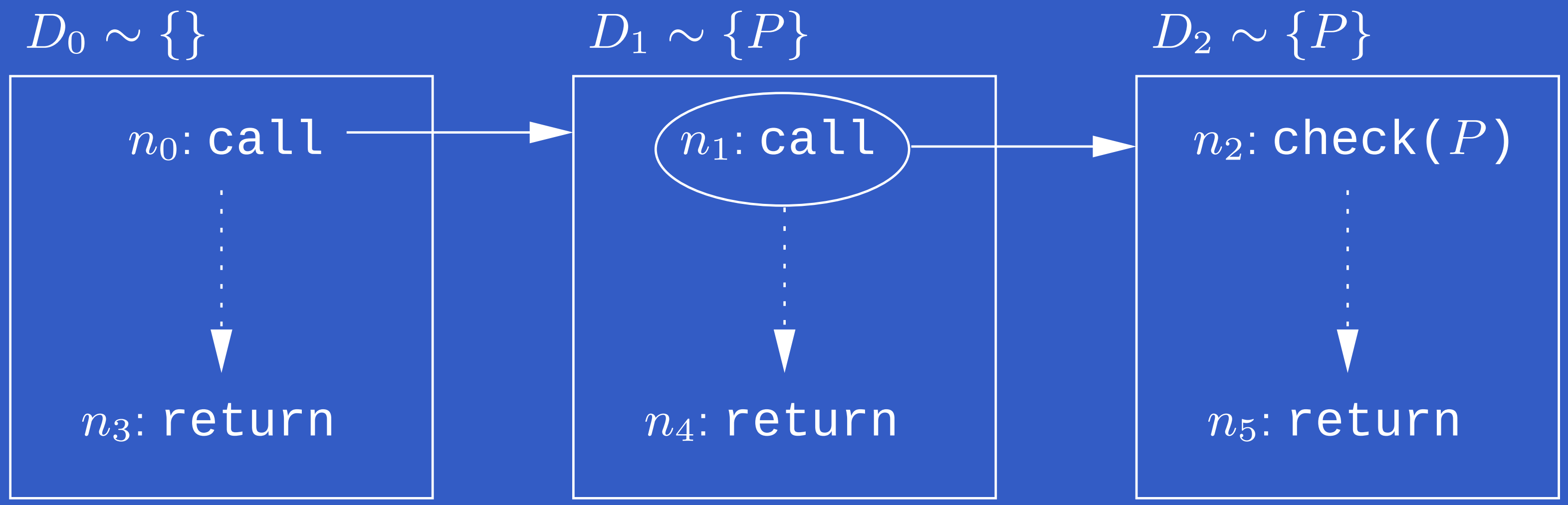
  $\Pi(n)$ is the set of permissions (statically) granted to n.

- Correctness of the optimization:

$$n \ \text{ is redundant} \qquad \iff \qquad P \in \Pi(n)$$

Method inlining (1)

Example 1 (before inlining)



$D_0 \sim \{\}$

n_0: `call`

n_3: `return`

$D_1 \sim \{P\}$

n_1: `call`

n_4: `return`

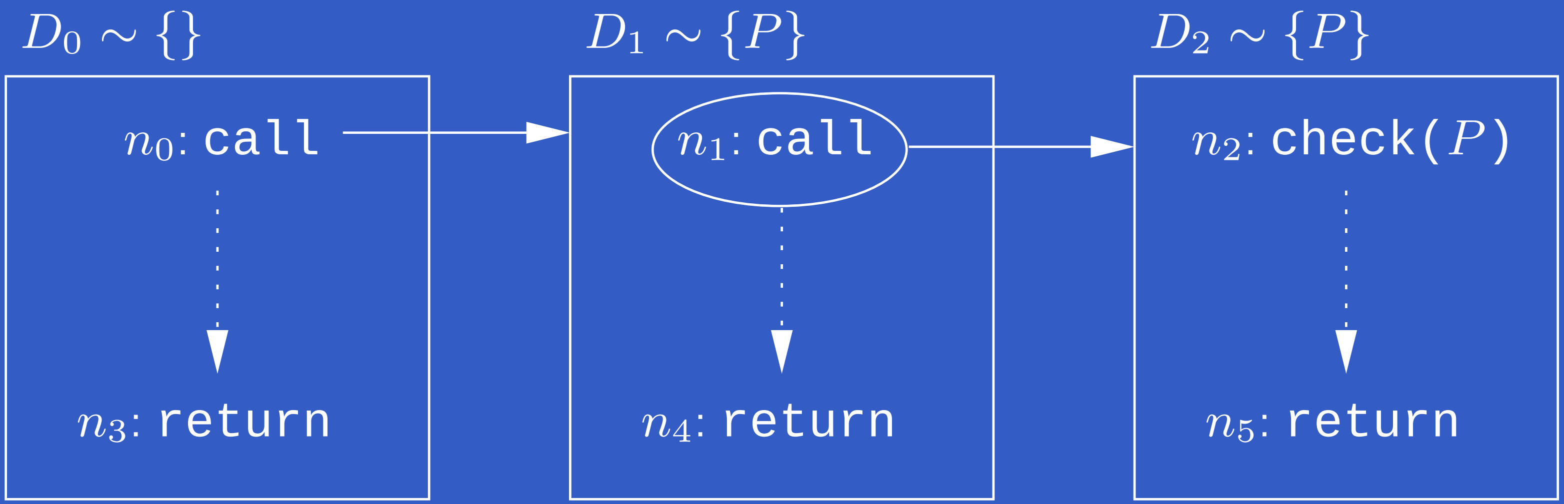
$D_2 \sim \{P\}$

n_2: `check(`P`)`

n_5: `return`

Method inlining (1)

Example 1 (before inlining)



$D_0 \sim \{\}$

$D_1 \sim \{P\}$

$D_2 \sim \{P\}$

n_0: `call`

n_3: `return`

n_1: `call`

n_4: `return`

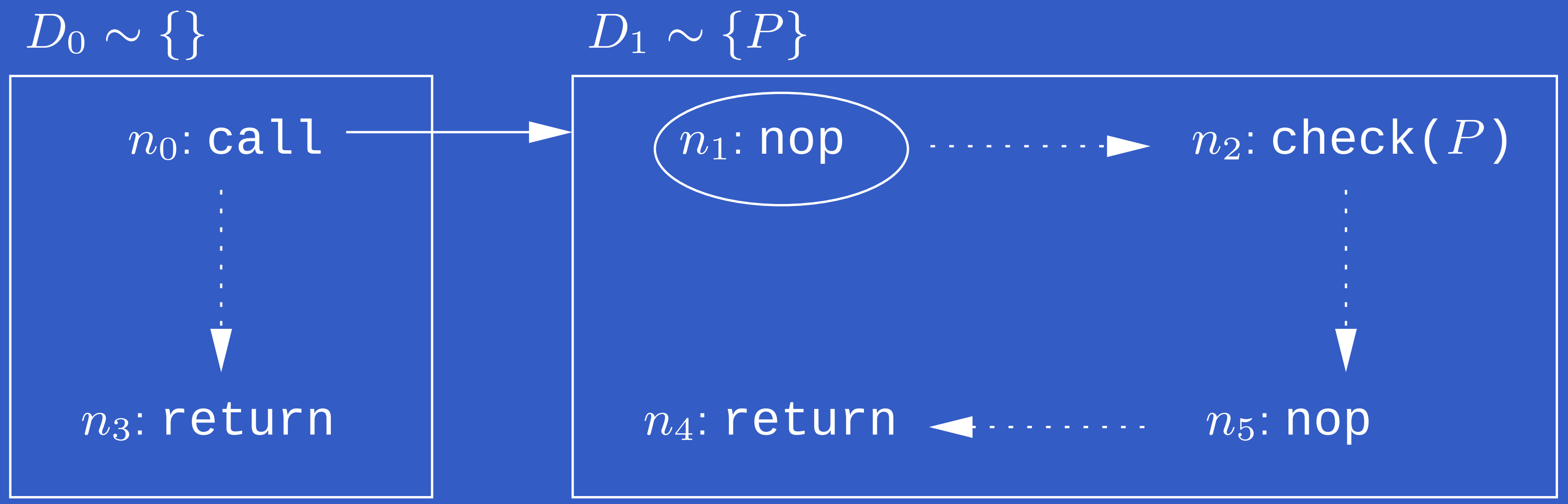
n_2: `check(`P`)`

n_5: `return`

$$G \triangleright [n_0] \triangleright [n_0, n_1] \triangleright [n_0, n_1, n_2] \vdash P$$

Method inlining (2)

Example 1 (after inlining of n_1)



$D_0 \sim \{\}$

$D_1 \sim \{P\}$

n_0: `call`

n_3: `return`

n_1: `nop`

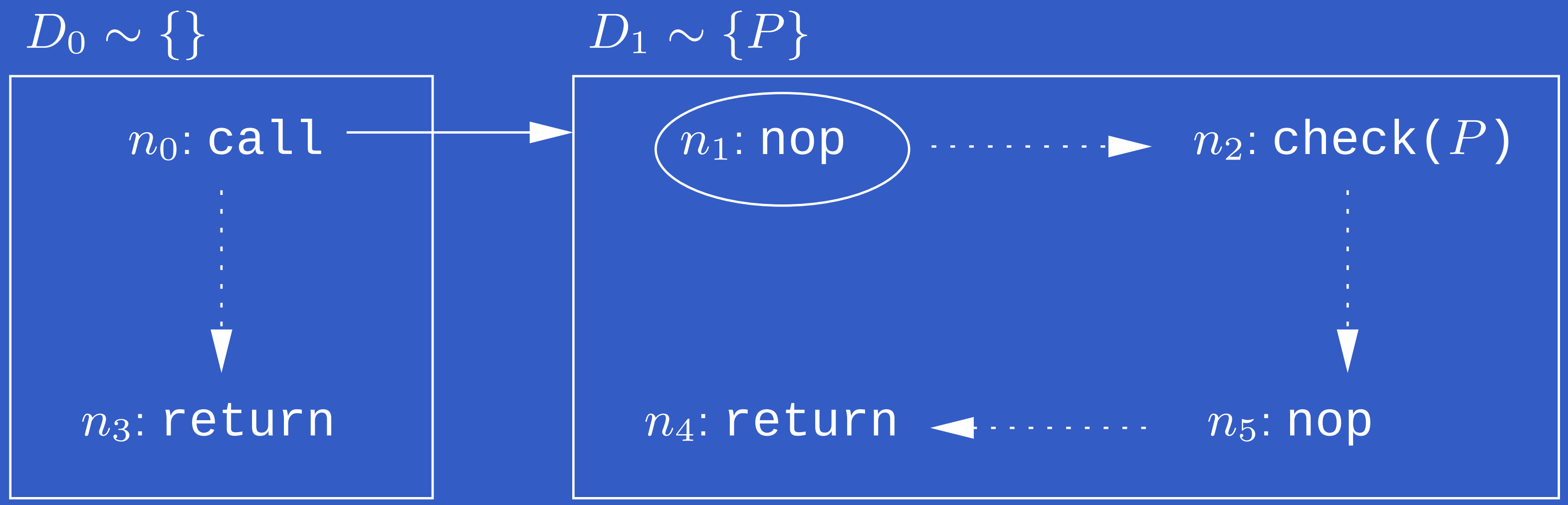
n_2: `check(`P`)`

n_5: `nop`

n_4: `return`

Method inlining (2)

Example 1 (after inlining of n_1)



$D_0 \sim \{\}$

n_0: `call`

n_3: `return`

$D_1 \sim \{P\}$

n_1: `nop`

n_2: `check(`P`)`

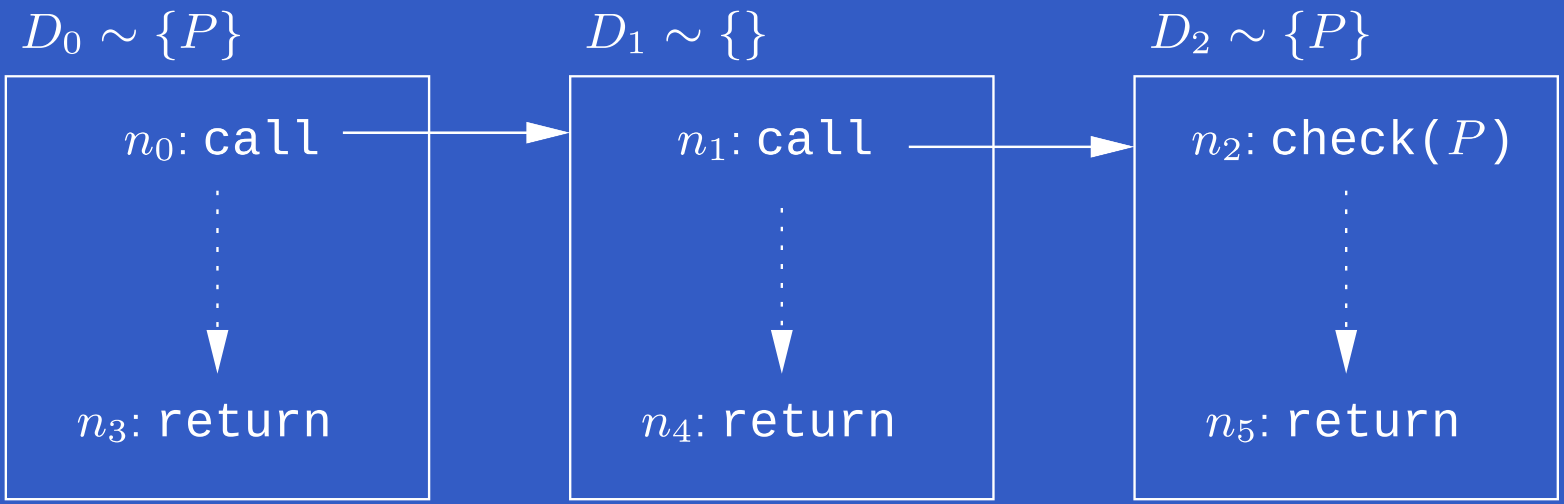
n_5: `nop`

n_4: `return`

$$\dot{G} \rhd [n_0] \rhd [n_0, n_1] \rhd [n_0, n_2] \not\vdash P$$

Method inlining (3)

Example 2 (before inlining)



$D_0 \sim \{P\}$

$D_1 \sim \{\}$

$D_2 \sim \{P\}$

n_0: `call`

n_3: `return`

n_1: `call`

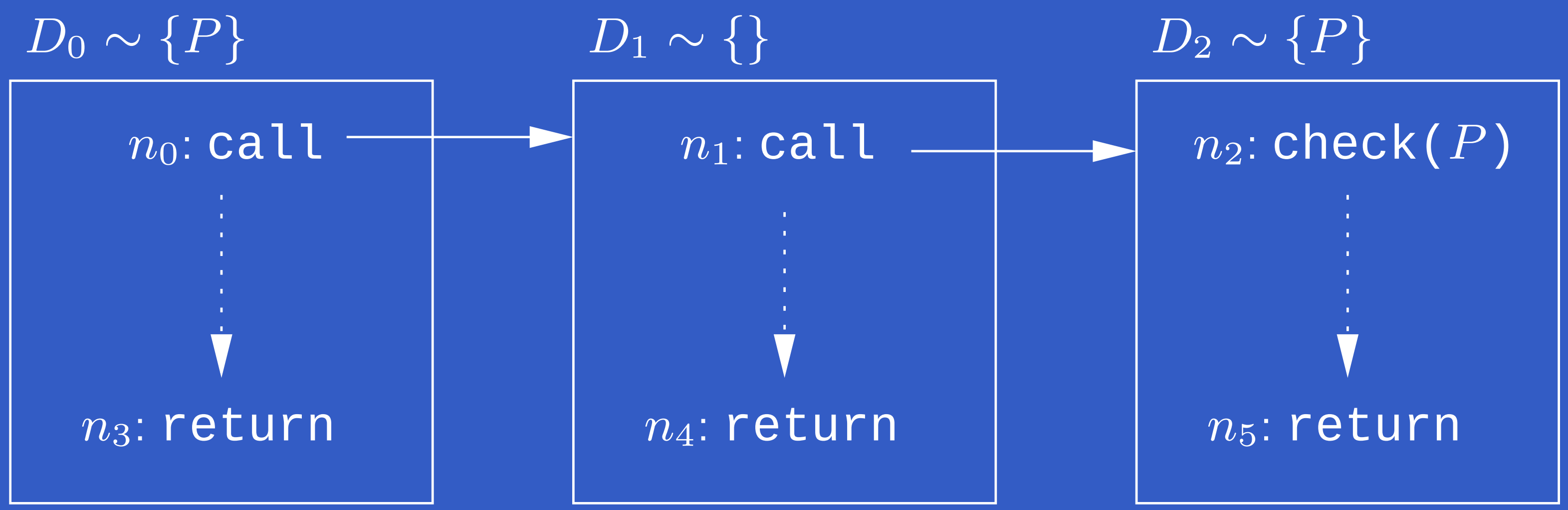
n_4: `return`

n_2: `check(`P`)`

n_5: `return`

Method inlining (3)

Example 2 (before inlining)



$D_0 \sim \{P\}$

n_0: `call`

n_3: `return`

$D_1 \sim \{\}$

n_1: `call`

n_4: `return`

$D_2 \sim \{P\}$

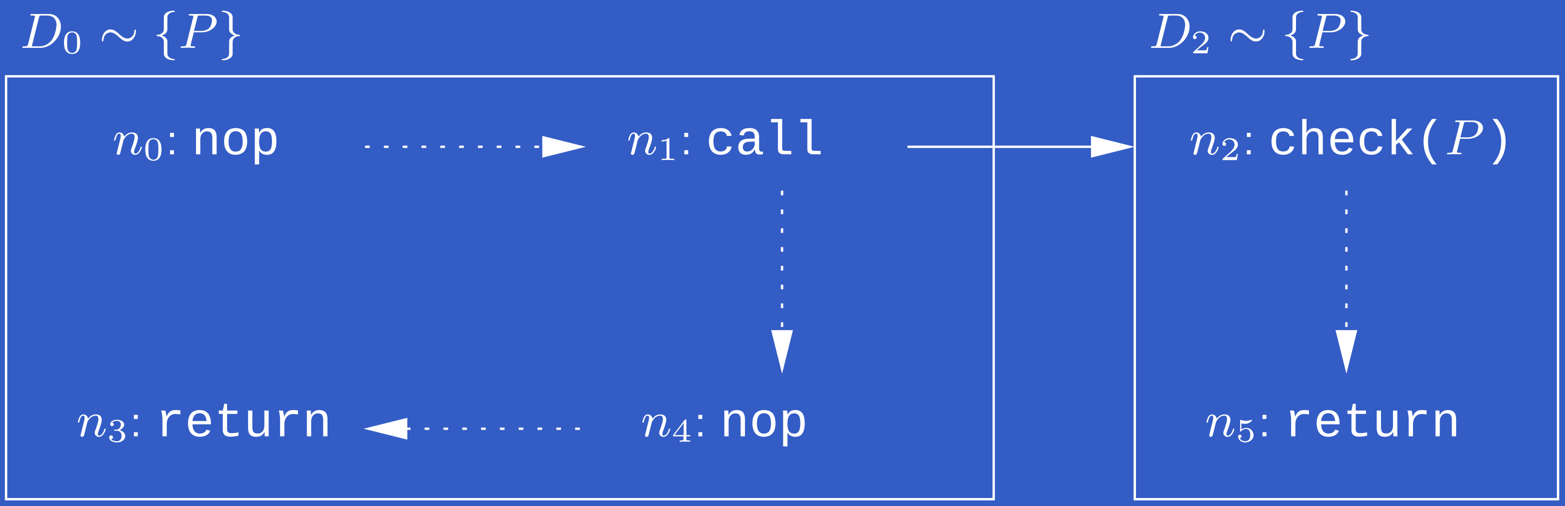
n_2: `check(`P`)`

n_5: `return`

$$G \rhd [n_0] \rhd [n_0, n_1] \rhd [n_0, n_1, n_2] \not\vdash P$$

Method inlining (4)

Example 2 (after inlining of n_0)



$D_0 \sim \{P\}$

n_0: `nop`

n_1: `call`

n_3: `return`

n_4: `nop`

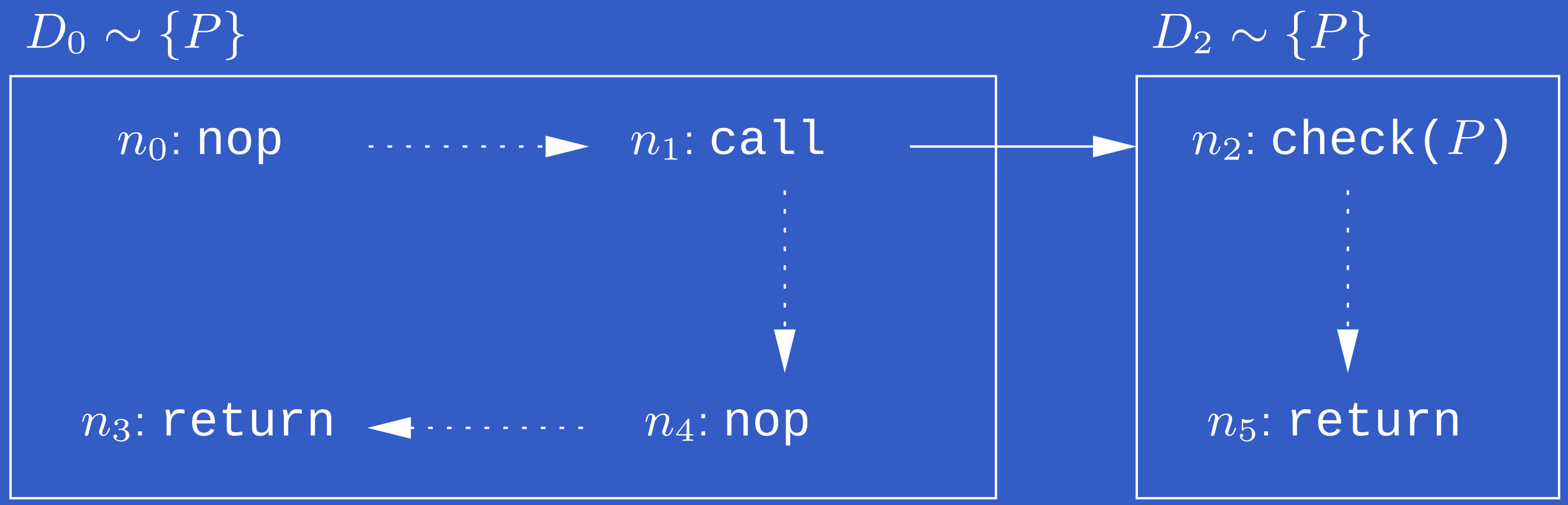
$D_2 \sim \{P\}$

n_2: `check(`P`)`

n_5: `return`

Method inlining (4)

Example 2 (after inlining of n_0)



$$\dot{G} \rhd [n_0] \rhd [n_1] \rhd [n_1, n_2] \vdash P$$

Method inlining (5)

Let $\dot{n} \longrightarrow n'$ be the call candidate for inlining. We require:

- static dispatching, non-recursiveness:

$$\forall m' \in N.\, \dot{n} \longrightarrow m' \implies m' = n' \;\wedge\; m' \notin \mu(\dot{n})$$

Method inlining (5)

Let $\dot{n} \longrightarrow n'$ be the call candidate for inlining. We require:

- static dispatching, non-recursiveness:

$$\forall m' \in N.\ \dot{n} \longrightarrow m' \implies m' = n' \ \wedge \ m' \notin \mu(\dot{n})$$

- original version inlining:

$$\forall m \in N.\ m \longrightarrow n' \implies m = \dot{n}$$

Method inlining (5)

Let $\dot{n} \longrightarrow n'$ be the call candidate for inlining. We require:

- static dispatching, non-recursiveness:

$$\forall m' \in N.\ \dot{n} \longrightarrow m' \implies m' = n' \ \wedge \ m' \notin \mu(\dot{n})$$

- original version inlining:

$$\forall m \in N.\ m \longrightarrow n' \implies m = \dot{n}$$

- isolation of the protection domain of n':

$$\forall m \notin \mu(n').\ \textsf{Dom}(m) \neq \textsf{Dom}(n')$$

Method inlining (6)

The key idea is the following:

- method inlining is safe iff the outcome of the security checks is preserved.

Method inlining (6)

The key idea is the following:

- method inlining is safe iff the outcome of the security checks is preserved.
- let Dom$(\dot{n}) = D$, Dom$(n') = D'$. We define:

$$Inl_{\dot{n}}(\gamma) = \begin{cases} \gamma & \text{if } D' \notin \gamma \\ (\gamma \setminus \{D'\}) \cup \{D\} & \text{otherwise} \end{cases}$$

Method inlining (6)

The key idea is the following:

- method inlining is safe iff the outcome of the security checks is preserved.
- let Dom$(\dot{n}) = D$, Dom$(n') = D'$. We define:

$$Inl_{\dot{n}}(\gamma) = \begin{cases} \gamma & \text{if } D' \notin \gamma \\ (\gamma \setminus \{D'\}) \cup \{D\} & \text{otherwise} \end{cases}$$

- the three conditions above guarantee that $Inl_{\dot{n}}(\gamma)$ is the context *after* the inlining of $\dot{n}$.

Method inlining (7)

The correctness of method inlining is decided as follows:

- assume that a solution τ to the TP analysis is available. We assign a fresh name to Dom(n'), then we restart the worklist algorithm from $\dot{n}$.

Method inlining (7)

The correctness of method inlining is decided as follows:

- assume that a solution τ to the TP analysis is available. We assign a fresh name to Dom(n'), then we restart the worklist algorithm from $\dot{n}$.
- each time we reach a node $\ell(n) =$ `check`(P), we check that, for each context $\gamma \in \tau(n)$,

$$P \in \Pi(\gamma) \iff P \in \Pi(Inl_{\dot{n}}(\gamma))$$

Method inlining (7)

The correctness of method inlining is decided as follows:

- assume that a solution τ to the TP analysis is available. We assign a fresh name to Dom(n'), then we restart the worklist algorithm from $\dot{n}$.
- each time we reach a node $\ell(n) =$ check(P), we check that, for each context $\gamma \in \tau(n)$,

$$P \in \Pi(\gamma) \iff P \in \Pi(Inl_{\dot{n}}(\gamma))$$

- if this is true for each node reached after the call $\dot{n}$, then $\dot{n}$ is inlineable in G.

Method inlining (8)

We define the *$\dot{n}$-inlined version* of a CFG $G = \langle N \cup \{n_\varepsilon\}, E, \text{Priv}_G, \text{Dom}_G \rangle$ as $\dot{G} = \langle N \cup \{n_\varepsilon\}, E, \text{Priv}_{\dot{G}}, \text{Dom}_{\dot{G}} \rangle$, where:

$$\text{Priv}_{\dot{G}}(n) = \begin{cases} \mathit{true} & \text{if } \text{Priv}_G(\dot{n}) \text{ and } \dot{n} \longrightarrow \mu(n) \\ \text{Priv}_G(n) & \text{otherwise} \end{cases}$$

$$\text{Dom}_{\dot{G}}(n) = \begin{cases} \text{Dom}_G(\dot{n}) & \text{if } \dot{n} \longrightarrow \mu(n) \\ \text{Dom}_G(n) & \text{otherwise} \end{cases}$$

Method inlining (9)

Method call

$$\frac{\ell(n) = \texttt{call} \quad n \longrightarrow n' \quad n \neq \dot{n}}{\sigma : n \rhd_{inl}^{\dot{n}} \sigma : n : n'}$$

$$\frac{\ell(\dot{n}) = \texttt{call} \quad \dot{n} \longrightarrow n'}{\sigma : \dot{n} \rhd_{inl}^{\dot{n}} \sigma : n'}$$

Method inlining (10)

Method return

$$\frac{\ell(n') = \texttt{return} \quad \dot{n} \dashrightarrow m \quad \dot{n} \not\longrightarrow \mu(n')}{\sigma : n : n' \rhd^{\dot{n}}_{inl} \sigma : m}$$

$$\frac{\ell(n') = \texttt{return} \quad \dot{n} \dashrightarrow m \quad \dot{n} \longrightarrow \mu(n')}{\sigma : n' \rhd^{\dot{n}}_{inl} \sigma : m}$$

Method inlining (11)

Theorem (*Correctness of method inlining.*)

If $\dot{n}$ is inlineable in G and $\dot{G}$ is the $\dot{n}$-inlined version of G:

$$\langle \sigma_0, x_0 \rangle \;\rhd\; \cdots \;\rhd\; \langle \sigma_k, x_k \rangle$$

$$\Longleftrightarrow$$

$$\langle \dot{\sigma}_0, x_0 \rangle \;\rhd^{\dot{n}}_{inl}\; \cdots \;\rhd^{\dot{n}}_{inl}\; \langle \dot{\sigma}_k, x_k \rangle$$

where $\sigma_0 = []$, $x_0 = \mathit{false}$, and $\dot{\sigma}_i = \mathit{inl}_{\dot{n}}(\sigma_i)$ for $i \in 0..k$.

$$\frac{\mathit{inl}_{\dot{n}}(\sigma) = \dot{\sigma} \quad \mathit{top}(\sigma) \neq \dot{n}}{\mathit{inl}_{\dot{n}}(\sigma : n') = \dot{\sigma} : n'} \qquad \frac{\mathit{inl}_{\dot{n}}(\sigma) = \dot{\sigma}}{\mathit{inl}_{\dot{n}}(\sigma : \dot{n} : n') = \dot{\sigma} : n'}$$

Conclusions

- interprocedural optimizations in presence of stack inspection
  - + based on solid static techniques (CFA)
  - + no update of the security context
  - + dynamic linking is possible
  - - overhead at linking time / deoptimization
- TO DO:
  - parametric permissions (ongoing work)
  - dynamic policies (ongoing work)
  - implementation & performance evaluation

Appendix - Def. of the TP Analysis (1)

$$TP_{in}(n) \quad = \quad \bigcup_{(m,n)\in E} TP_{out}(m,n)$$

$$TP_{out}(m,n) \quad = \quad \begin{cases} \{\{\mathsf{Dom}(n)\}\} & \text{if } \quad \bullet\!\!\longrightarrow n \\ \{\,\gamma \cup \{\mathsf{Dom}(n)\} \mid \gamma \in TP_{call}(m)\,\} & \text{if } m \longrightarrow n \\ TP_{trans}(m) & \text{if } m \dashrightarrow n \\ TP_{catch}(m) & \text{if } m \dashrightarrow_{\lightning} n \end{cases}$$

$$TP_{call}(n) \quad = \quad \begin{cases} \{\{\mathsf{Dom}(n)\}\} & \text{if } \mathsf{Priv}(n) \text{ and } TP_{in}(n) \neq \emptyset \\ \quad TP_{in}(n) & \text{otherwise} \end{cases}$$

Appendix - Def. of the TP Analysis (2)

$$TP_{trans}(n) = \begin{cases} \{\, \gamma \in TP_{in}(n) \mid P \in \Pi(\gamma) \,\} & \text{if } \ell(n) = \mathsf{check}(P) \\ \{\, \gamma \in TP_{in}(n) \mid \mathsf{Trans}(n, \{\mathsf{Dom}(n)\}) \,\} & \text{if } \ell(n) = \mathtt{call}, \mathsf{Priv}(n) \\ \{\, \gamma \in TP_{in}(n) \mid \mathsf{Trans}(n, \gamma) \,\} & \text{otherwise} \end{cases}$$

$$\mathsf{Trans}(n, \gamma) \quad \overset{def}{=} \quad \exists m \in \rho(n).\ \gamma \cup \{\mathsf{Dom}(m)\} \in TP_{in}(m)$$

$$TP_{catch}(n) = \begin{cases} \{\, \gamma \in TP_{in}(n) \mid P \notin \Pi(\gamma) \,\} & \text{if } \ell(n) = \mathsf{check}(P) \\ \{\, \gamma \in TP_{in}(n) \mid \mathsf{Catch}(n, \{\mathsf{Dom}(n)\}) \,\} & \text{if } \ell(n) = \mathtt{call}, \mathsf{Priv}(n) \\ \{\, \gamma \in TP_{in}(n) \mid \mathsf{Catch}(n, \gamma) \,\} & \text{otherwise} \end{cases}$$

$$\mathsf{Catch}(n, \gamma) \quad \overset{def}{=} \quad \exists m \in \xi_1(n).\ \gamma \cup \{\mathsf{Dom}(m)\} \in TP_{catch}(m)$$