基于类型的资源分析

OCAML

```
let rec append l1 l2 =
  match l1 with
  | [] -> l2
  | x::xs -> x::(append xs l2)
```

RAML

[HDW17] J. Hoffmann, A. Das, and S.-C. Weng. 2017. Towards Automatic Resource Bound Analysis for OCaml. In *POPL'17*.

基于类型的资源分析

OCAML

```
let rec append l1 l2 =
  match l1 with
  | [] -> l2
  | x::xs -> x::(append xs l2)
```

RAML

$\text{append} : \langle L^9(\alpha) \times L^0(\alpha), 3\rangle \rightarrow \langle L^0(\alpha), 0\rangle$

[HDW17] J. Hoffmann, A. Das, and S.-C. Weng. 2017. Towards Automatic Resource Bound Analysis for OCaml. In *POPL'17*.

基于类型的资源分析

OCAML

```
let rec append l1 l2 =
  match l1 with
  | [] -> l2
  | x::xs -> x::(append xs l2)
```

RAML

append : 带有资源消耗信息的类型

[HDW17] J. Hoffmann, A. Das, and S.-C. Weng. 2017. Towards Automatic Resource Bound Analysis for OCaml. In *POPL'17*.

基于类型的资源分析

OCAML

```
let rec append l1 l2 =
  match l1 with
  | [] -> l2
  | x::xs -> x::(append xs l2)
```

RAML

append : 带有资源消耗信息的类型

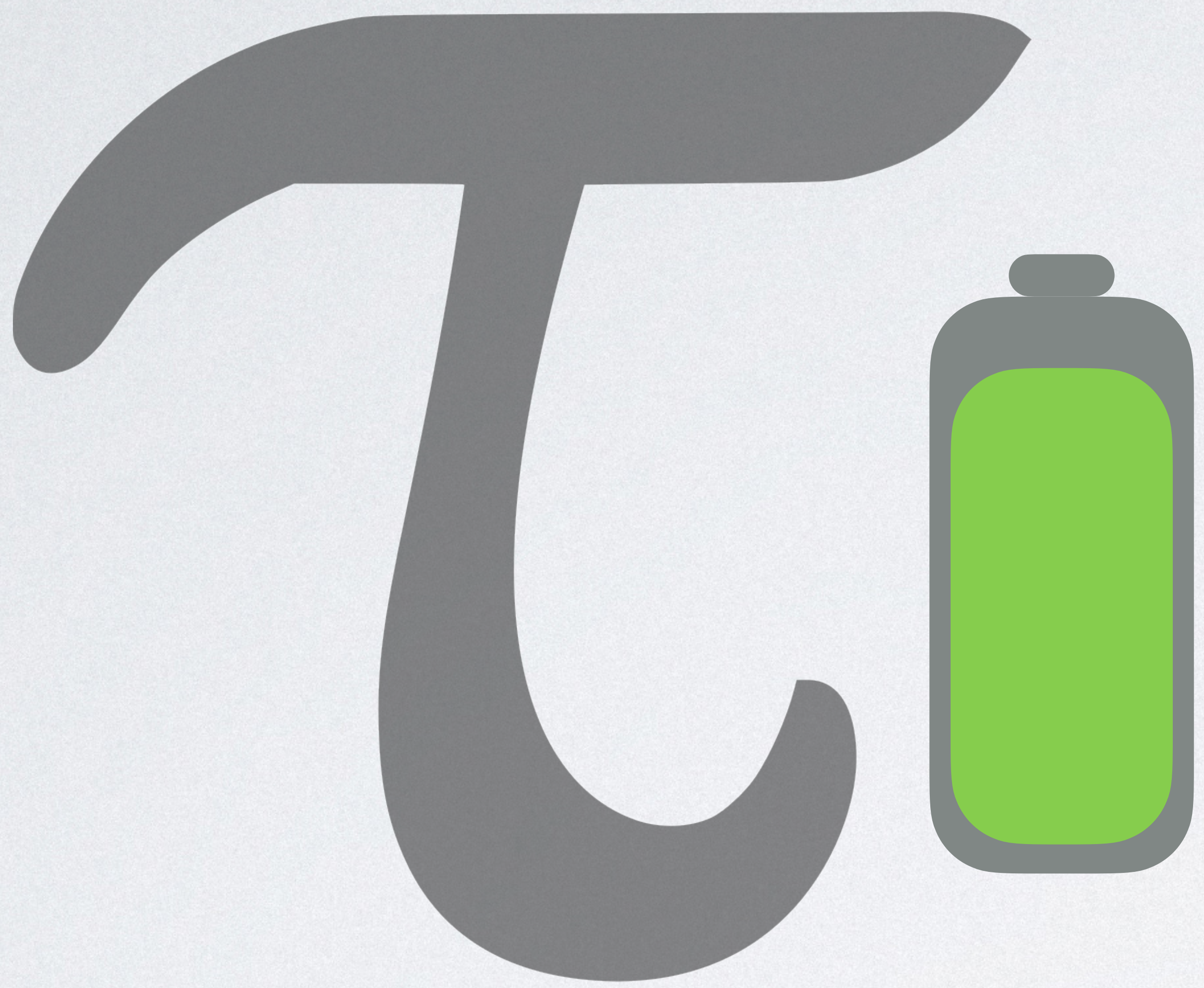
简化后可得到资源消耗的上界：

$$9|\ell_1| + 3 = O(|\ell_1|)$$

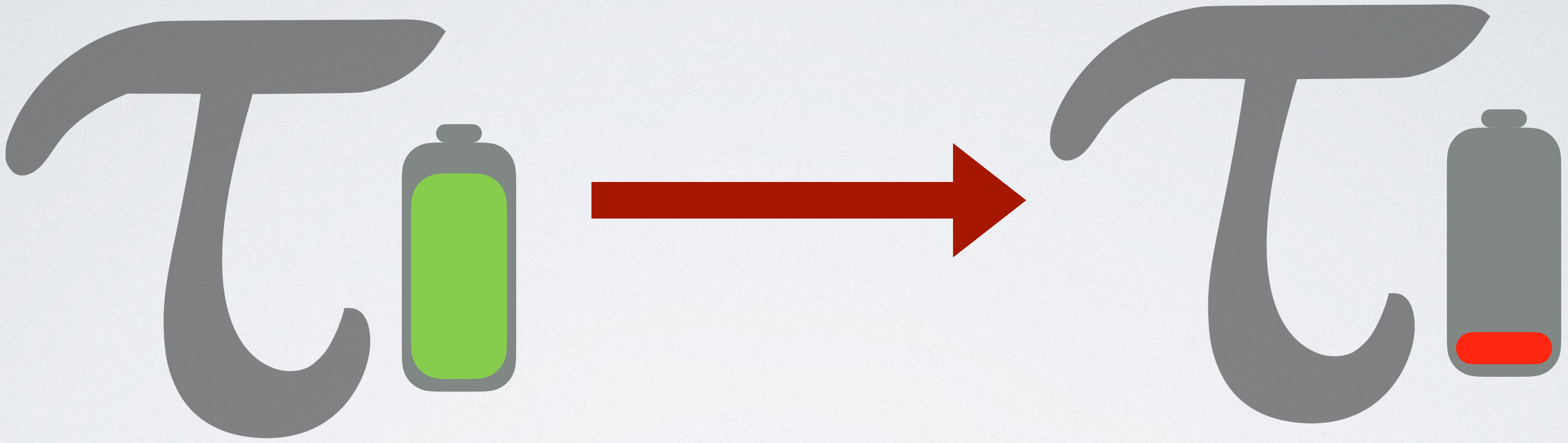
[HDW17] J. Hoffmann, A. Das, and S.-C. Weng. 2017. Towards Automatic Resource Bound Analysis for OCaml. In *POPL'17*.

基于类型的资源分析

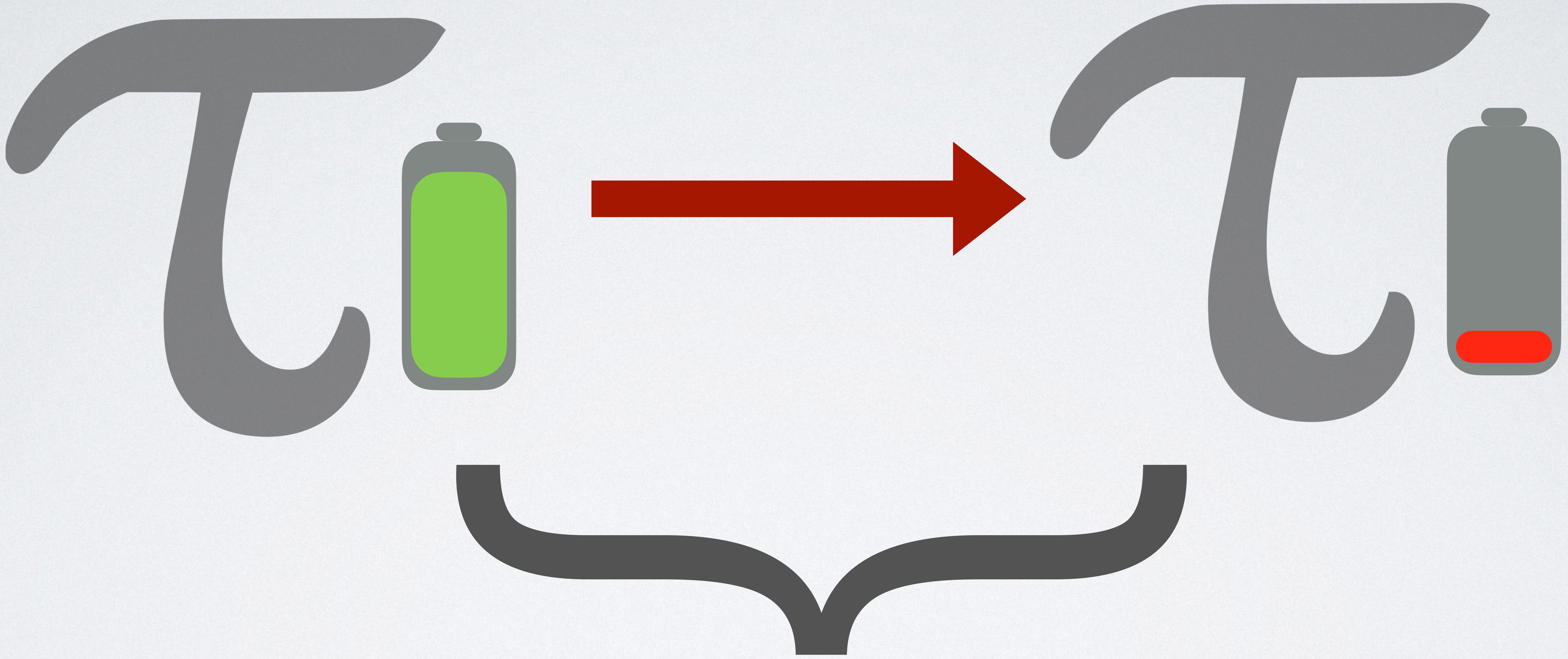
基于类型的资源分析



基于类型的资源分析



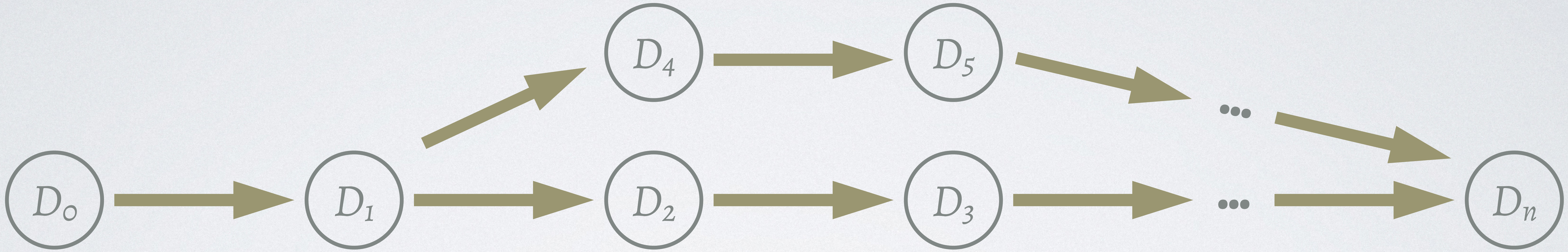
基于类型的资源分析



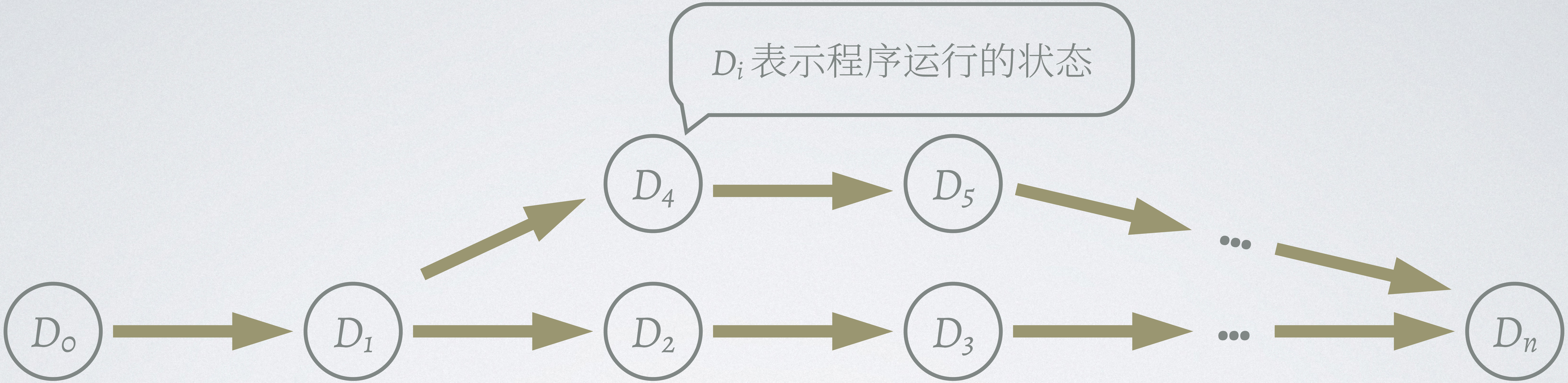
资源消耗

均摊分析的势能方法

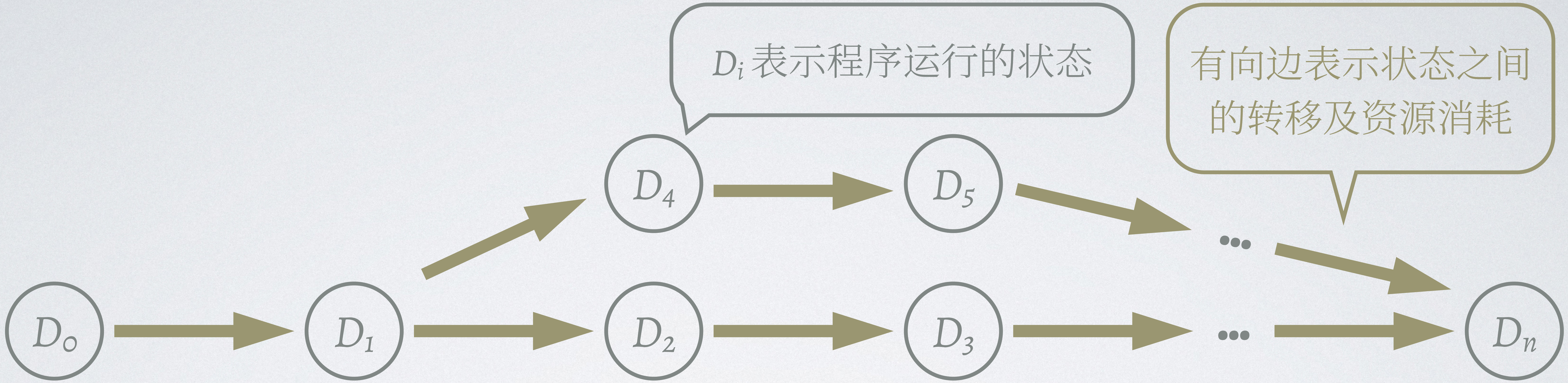
均摊分析的势能方法



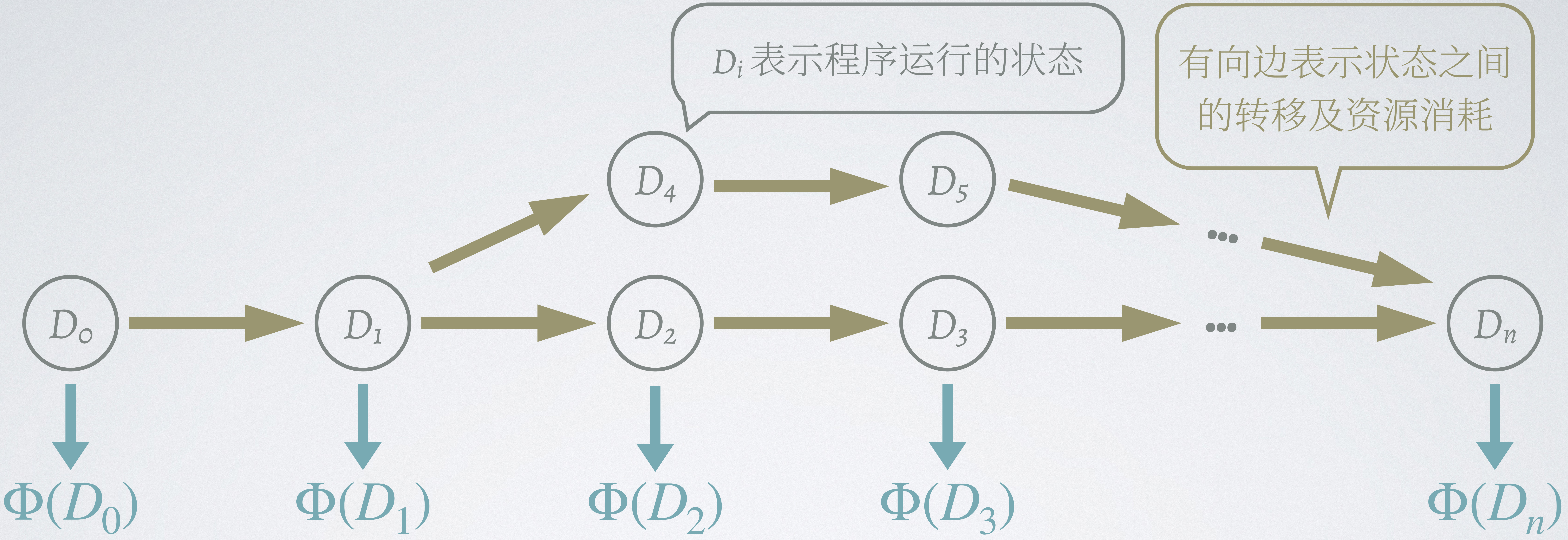
均摊分析的势能方法



均摊分析的势能方法

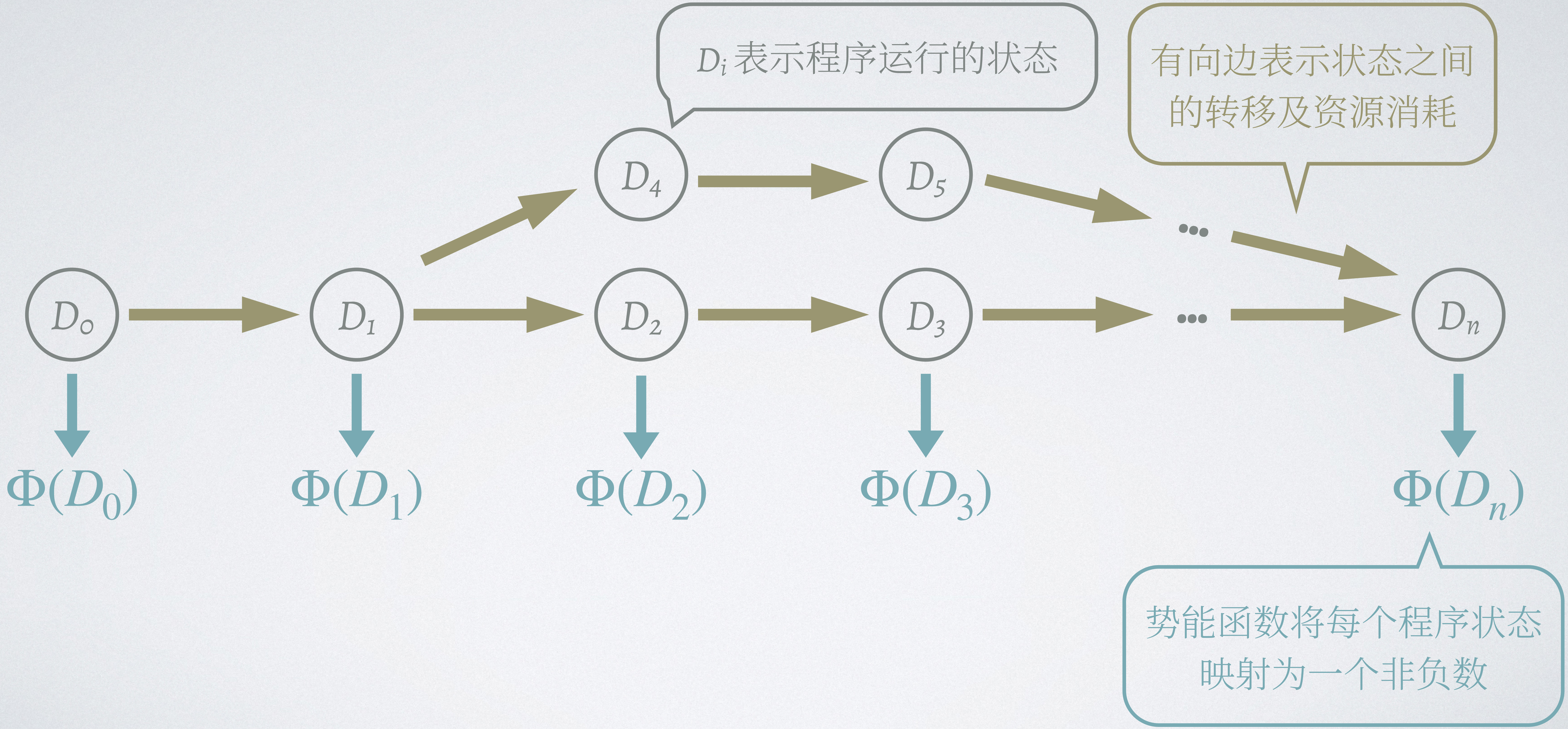


均摊分析的势能方法



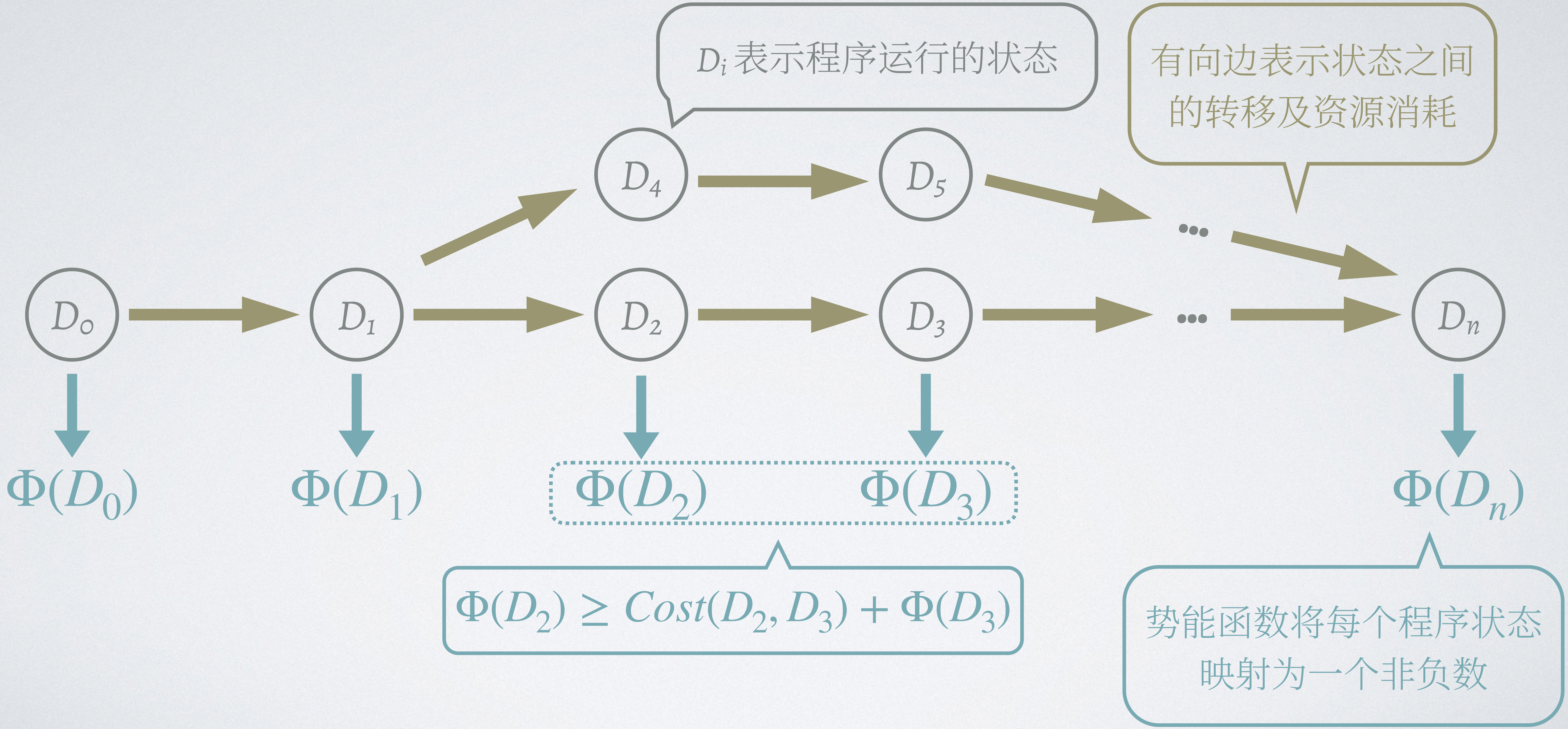
D_i 表示程序运行的状态
有向边表示状态之间的转移及资源消耗
D_4
D_5
…
D_0
D_1
D_2
D_3
…
D_n
$\Phi(D_0)$
$\Phi(D_1)$
$\Phi(D_2)$
$\Phi(D_3)$
$\Phi(D_n)$

均摊分析的势能方法



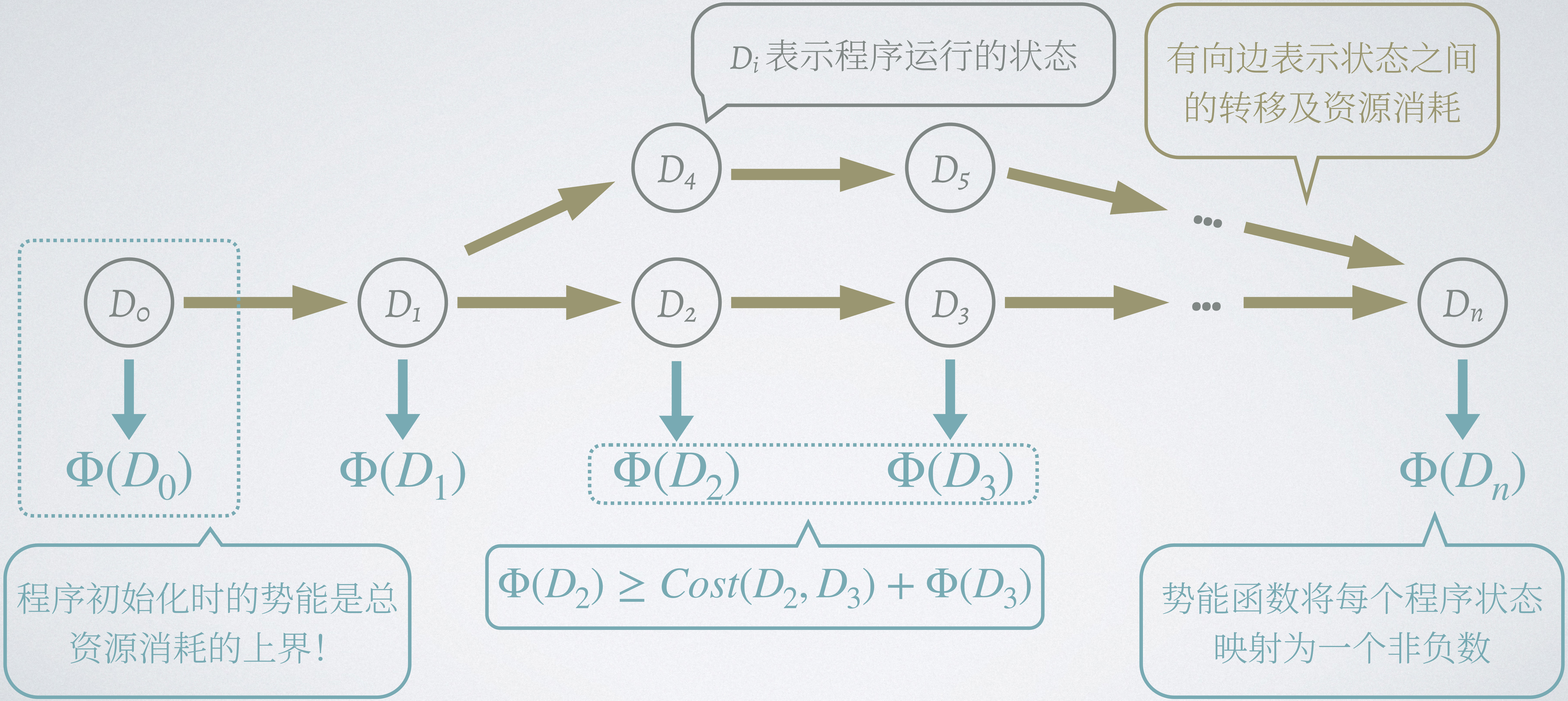
D_i 表示程序运行的状态
有向边表示状态之间的转移及资源消耗
D_4
D_5
D_0
D_1
D_2
D_3
D_n
$\Phi(D_0)$
$\Phi(D_1)$
$\Phi(D_2)$
$\Phi(D_3)$
$\Phi(D_n)$
势能函数将每个程序状态映射为一个非负数

均摊分析的势能方法



D_i 表示程序运行的状态
有向边表示状态之间的转移及资源消耗
D_0
D_1
D_2
D_3
D_4
D_5
...
D_n
$\Phi(D_0)$
$\Phi(D_1)$
$\Phi(D_2)$
$\Phi(D_3)$
$\Phi(D_n)$
$\Phi(D_2) \geq Cost(D_2, D_3) + \Phi(D_3)$
势能函数将每个程序状态映射为一个非负数

均摊分析的势能方法



D_i 表示程序运行的状态
有向边表示状态之间的转移及资源消耗
D_0 D_1 D_2 D_3 D_4 D_5 … D_n
$\Phi(D_0)$ $\Phi(D_1)$ $\Phi(D_2)$ $\Phi(D_3)$ $\Phi(D_n)$
程序初始化时的势能是总资源消耗的上界！
$\Phi(D_2) \geq Cost(D_2, D_3) + \Phi(D_3)$
势能函数将每个程序状态映射为一个非负数

带有势能标注的类型

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

带有势能标注的类型

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 tick 显式标注
程序的资源消耗模型

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \to \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 tick 显式标注程序的资源消耗模型

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \to \langle L^0(\alpha), 0\rangle$

$L^p(a)$

列表中的每个元素都携带了 p 单位的势能

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 tick 显式标注程序的资源消耗模型

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \to \langle L^0(\alpha), 0\rangle$

$L^p(a)$

列表中的每个元素都携带了 p 单位的势能

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 tick 显式标注程序的资源消耗模型

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \rightarrow \langle L^0(\alpha), 0\rangle$

$L^p(a)$

列表中的每个元素都携带了 p 单位的势能

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 tick 显式标注程序的资源消耗模型

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \to \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 tick 显式标注程序的资源消耗模型

$L^p(a)$

列表中的每个元素都携带了 p 单位的势能

[l1: $L^1(a)$, l2: $L^0(a)$]; 0 units

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \rightarrow \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 tick 显式标注程序的资源消耗模型

$L^p(a)$

列表中的每个元素都携带了 p 单位的势能

```
[l1: L¹(a), l2: L⁰(a)]; 0 units
// l1 被消耗
```

带有势能标注的类型

$Cost = |\ell_1|$

$\text{append} : \langle L^1(\alpha) \times L^0(\alpha), 0\rangle \to \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 tick 显式标注程序的资源消耗模型

$L^p(a)$

列表中的每个元素都携带了 p 单位的势能

```
[l1: L¹(a), l2: L⁰(a)]; 0 units
// l1 被消耗
[l2: L⁰(a)]; 0 units
```

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \rightarrow \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 tick 显式标注
程序的资源消耗模型

Lp(a)

列表中的每个元素都
携带了 p 单位的势能

```
[l1: L¹(a), l2: L⁰(a)]; 0 units
// l1 被消耗
[l2: L⁰(a)]; 0 units
// l2 被消耗且返回类型符合签名
```

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \to \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 tick 显式标注程序的资源消耗模型

Lp(a)

列表中的每个元素都携带了 p 单位的势能

```
[l1: L¹(a), l2: L⁰(a)]; 0 units
// l1 被消耗
[l2: L⁰(a)]; 0 units
// l2 被消耗且返回类型符合签名
[l2: L⁰(a), x: a, xs: L¹(a)]; 1 unit
```

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \rightarrow \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 tick 显式标注程序的资源消耗模型

Lp(a)

列表中的每个元素都携带了 p 单位的势能

```
[l1: L¹(a), l2: L⁰(a)]; 0 units
// l1 被消耗
[l2: L⁰(a)]; 0 units
// l2 被消耗且返回类型符合签名
[l2: L⁰(a), x: a, xs: L¹(a)]; 1 unit
[l2: L⁰(a), x: a, xs: L¹(a)]; 0 units
```

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \to \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 tick 显式标注
程序的资源消耗模型

Lp(a)

列表中的每个元素都
携带了 p 单位的势能

```
[l1: L¹(a), l2: L⁰(a)]; 0 units
// l1 被消耗
[l2: L⁰(a)]; 0 units
// l2 被消耗且返回类型符合签名
[l2: L⁰(a), x: a, xs: L¹(a)]; 1 unit
[l2: L⁰(a), x: a, xs: L¹(a)]; 0 units
[x: a, rest: L⁰(a)]; 0 units
```

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \rightarrow \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 tick 显式标注程序的资源消耗模型

Lp(a)

列表中的每个元素都携带了 p 单位的势能

```
[l1: L¹(a), l2: L⁰(a)]; 0 units
// l1 被消耗
[l2: L⁰(a)]; 0 units
// l2 被消耗且返回类型符合签名
[l2: L⁰(a), x: a, xs: L¹(a)]; 1 unit
[l2: L⁰(a), x: a, xs: L¹(a)]; 0 units
[x: a, rest: L⁰(a)]; 0 units
// x 和 rest 被消耗且返回类型符合签名
```

带有势能标注的类型

$Cost = |\ell_1|$

append : $\langle L^1(\alpha) \times L^0(\alpha), 0\rangle \rightarrow \langle L^0(\alpha), 0\rangle$

```
let rec append l1 l2 =
  match l1 with
  | [] ->
    l2
  | x::xs ->
    let () = tick(1) in
    let rest = append xs l2 in
    x::rest
```

通过 tick 显式标注程序的资源消耗模型

Lp(a)

列表中的每个元素都携带了 p 单位的势能

```
[l1: L¹(a), l2: L⁰(a)]; 0 units
// l1 被消耗
[l2: L⁰(a)]; 0 units
// l2 被消耗且返回类型符合签名
[l2: L⁰(a), x: a, xs: L¹(a)]; 1 unit
[l2: L⁰(a), x: a, xs: L¹(a)]; 0 units
[x: a, rest: L⁰(a)]; 0 units
// x 和 rest 被消耗且返回类型符合签名
```

原理：每个程序点的势能函数由程序操作的数据结构的静态类型标注所决定

AARA 的研究现状

[HDW17]	多元多项式形式的资源消耗上界，均摊资源分析
[Atkey10]	命令式编程语言，支持堆操作
[JHL+10]	函数式编程语言，支持高阶函数
[HM18]	对数形式的资源消耗上界（可分析伸展树）
[KH20]	指数形式的资源消耗上界
[WKH20]	对概率程序的期望资源消耗分析

[Atkey10] R. Atkey. 2010. Amortised Resource Analysis with Separation Logic. In *ESOP'10*.
[JHL+10] S. Jost, K. Hammond, H.-W. Loidl, and M. Hofmann. 2010. Static Determination of Quantitative Resource Usage for Higher-Order Programs. In *POPL'10*.
[HM18] M. Hofmann and G. Moser. 2018. Analysis of Logarithmic Amortised Complexity. Available on: https://arxiv.org/abs/1807.08242.
[KH20] D. M. Kahn and J. Hoffmann. 2020. Exponential Automatic Amortized Resource Analysis. In *FoSSaCS'20*.