编程语言新范式初探

王迪
北京大学
wangdi95@pku.edu.cn
2023 年 12 月 16 日

编程语言

- 编写软件的最基本工具
- 1966 - 2022 年的 57 届图灵奖中有 17 届（约 30%）颁给了编程语言的相关研究

意识中的计算

WA 24和36的最大公约数是多少?

24和36的最大公约数是12。

WA 请给出计算过程。

计算两个数的最大公约数通常使用欧几里德算法，以下是计算24和36的最大公约数的过程：

1. 用较大的数除以较小的数，即36 ÷ 24 = 1 余 12。
2. 接下来，将较小的数（24）除以余数（12），即24 ÷ 12 = 2 余 0。

当余数为0时，计算结束，最大公约数是上一步的余数，即12。所以，24和36的最大公约数是12。

意识中的计算

WA

24和36的最大公约数是多少?

24和36的最大公约数是12。

WA

请给出计算过程。

计算两个数的最大公约数通常使用欧几里德算法，以下是计算24和36的最大公约数的过程：

1. 用较大的数除以较小的数，即36 ÷ 24 = 1 余 12。
2. 接下来，将较小的数（24）除以余数（12），即24 ÷ 12 = 2 余 0。

当余数为0时，计算结束，最大公约数是上一步的余数，即12。所以，24和36的最大公约数是12。

硬件上的计算

编程语言

意识中的计算

WA 24和36的最大公约数是多少?

24和36的最大公约数是12。

WA 请给出计算过程。

计算两个数的最大公约数通常使用欧几里德算法，以下是计算24和36的最大公约数的过程：

1. 用较大的数除以较小的数，即36 ÷ 24 = 1 余 12。
2. 接下来，将较小的数（24）除以余数（12），即24 ÷ 12 = 2 余 0。

当余数为0时，计算结束，最大公约数是上一步的余数，即12。所以，24和36的最大公约数是12。

硬件上的计算

编程语言

意识中的计算

WA 24和36的最大公约数是多少?

24和36的最大公约数是12。

WA 请给出计算过程。

计算两个数的最大公约数通常使用欧几里德算法，以下是计算24和36的最大公约数的过程：

1. 用较大的数除以较小的数，即36 ÷ 24 = 1 余 12。
2. 接下来，将较小的数（24）除以余数（12），即24 ÷ 12 = 2 余 0。

当余数为0时，计算结束，最大公约数是上一步的余数，即12。所以，24和36的最大公约数是12。

硬件上的计算

让程序更加安全

让程序更容易写

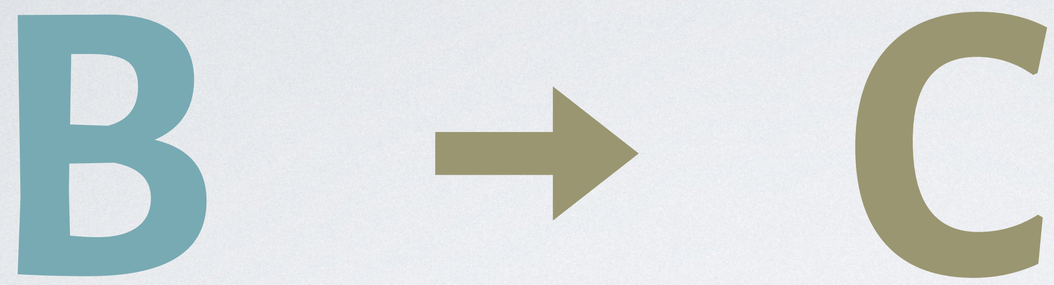
系统编程语言

系统编程语言

B

- 1969 年
- 贝尔实验室
- 为系统和编译器开发设计
- 无类型
- 非常慢！

系统编程语言



- 1969 年
- 贝尔实验室
- 为系统和编译器开发设计
- 无类型
- 非常慢！

- 1972 年
- 贝尔实验室
- 通用编程语言
  - 长期被用来开发操作系统、设备驱动、协议栈等
- 静态类型系统
- 很快啊！

系统编程语言

B → **C** → **Rust**

- 1969 年
- 贝尔实验室
- 为系统和编译器开发设计
- 无类型
- 非常慢！

- 1972 年
- 贝尔实验室
- 通用编程语言
  - 长期被用来开发操作系统、设备驱动、协议栈等
- 静态类型系统
- 很快啊！

- 2015 年
- Mozilla
- 通用编程语言
  - 强调性能、安全以及并发
- 基于类型的内存安全
- 性能还不错！

系统编程语言

B → **C** → **Rust**

- 1969 年
- 贝尔实验室
- 为系统和编译器开发设计
- 无类型
- 非常慢！

- 1972 年
- 贝尔实验室
- 通用编程语言
  - 长期被用来开发操作系统、设备驱动、协议栈等
- 静态类型系统
- 很快啊！

- 2015 年
- Mozilla
- 通用编程语言
  - 强调性能、安全以及并发
- 基于类型的内存安全
- 性能还不错！

系统编程语言

B → **C** → **Rust**

- 1969 年
- 贝尔实验室
- 为系统和编译器开发设计
- 无类型
- 非常慢！

- 1972 年
- 贝尔实验室
- 通用编程语言
  - 长期被用来开发操作系统、设备驱动、协议栈等
- 静态类型系统
- 很快啊！

- 2015 年
- Mozilla
- 通用编程语言
  - 强调性能、安全以及并发
- 基于类型的内存安全
- 性能还不错！

趋势：语言的安全性越来越高

无安全保障

非常慢

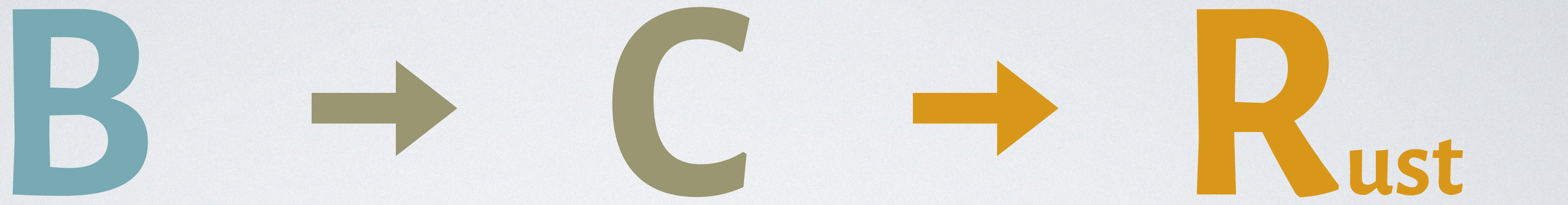
类型安全

很快啊

类型、内存、并发安全

挺快的

趋势：语言的安全性越来越高



无安全保障

非常慢

类型安全

很快啊

类型、内存、并发安全

挺快的

- 系统编程语言不仅要提供安全保障，还要提供性能保障

趋势：语言的安全性越来越高

B → **C** → **Rust**

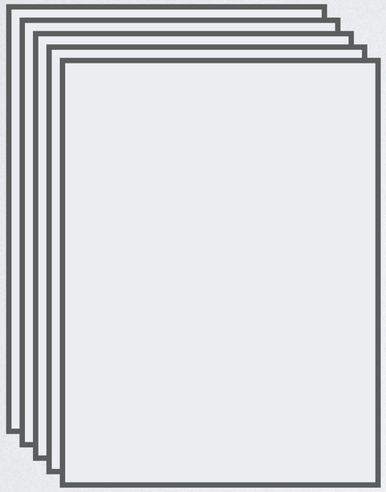
无安全保障 — 非常慢

类型安全 — 很快啊

类型、内存、并发安全 — 挺快的

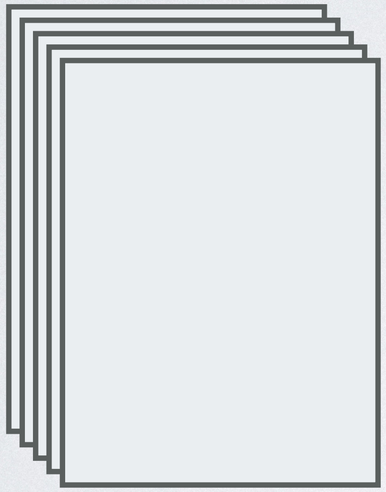
- 系统编程语言不仅要提供安全保障，还要提供性能保障
- 语言需要提供精细的对资源消耗的分析和管控

软件的资源消耗



软件

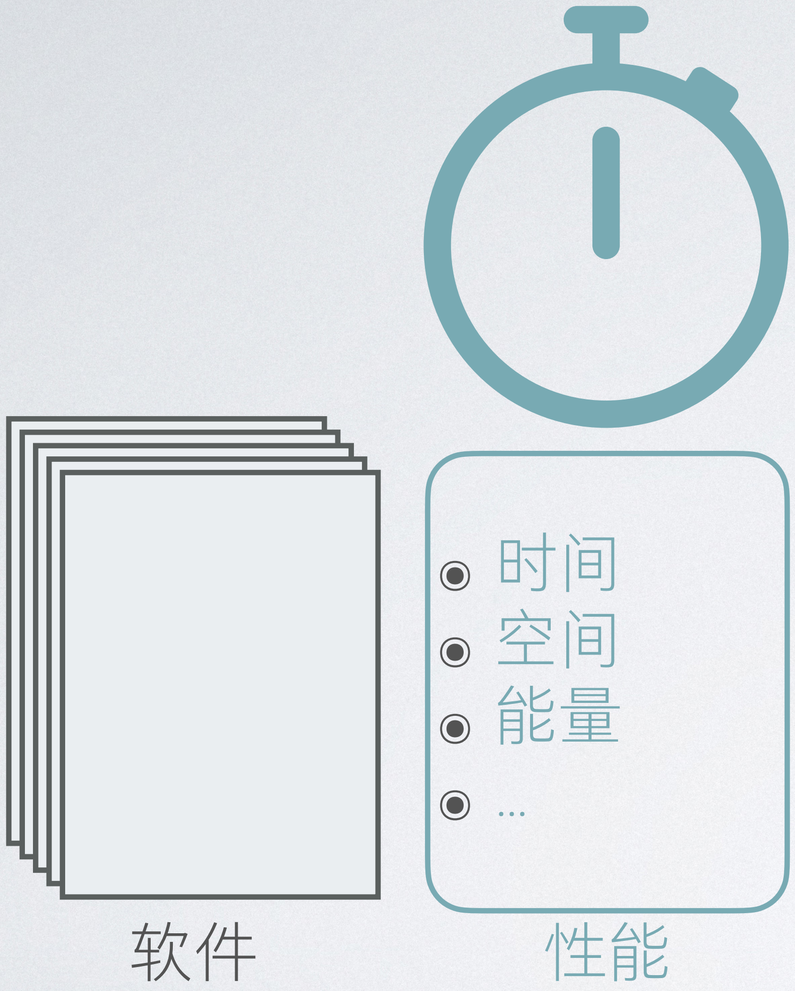
软件的资源消耗



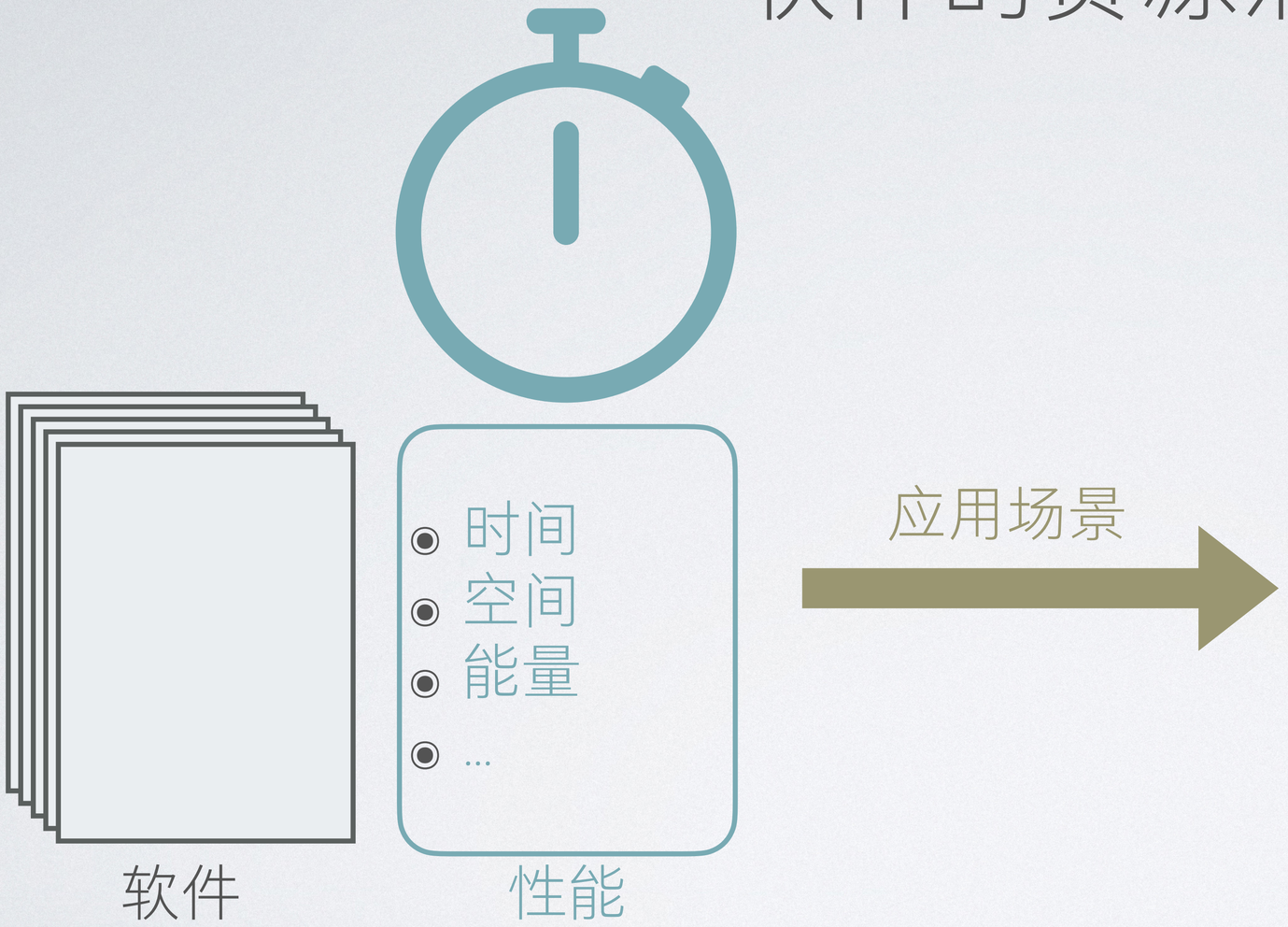
软件

性能

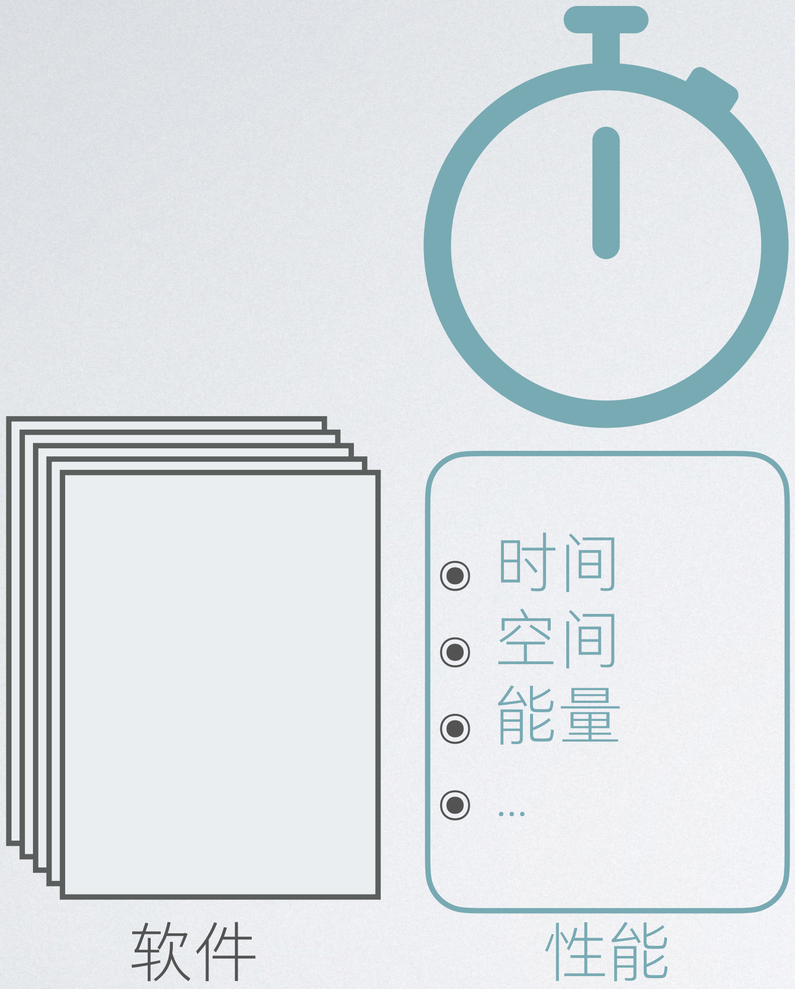
软件的资源消耗



软件的资源消耗



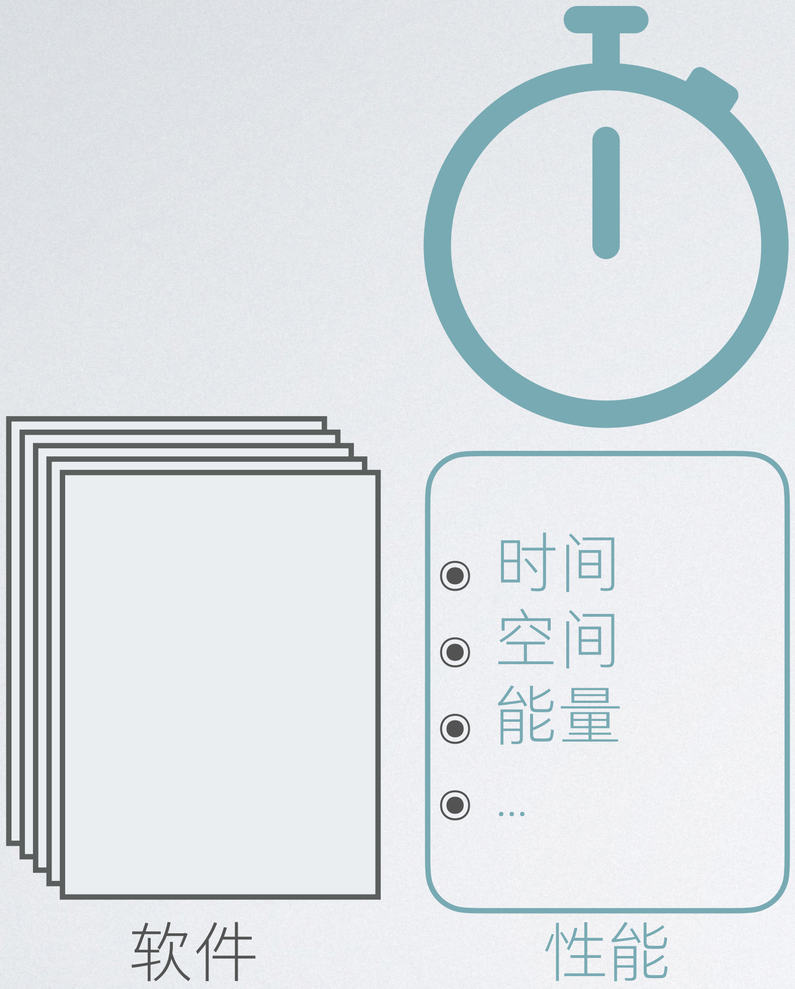
软件的资源消耗



应用场景

- 分析算法的时、空复杂度

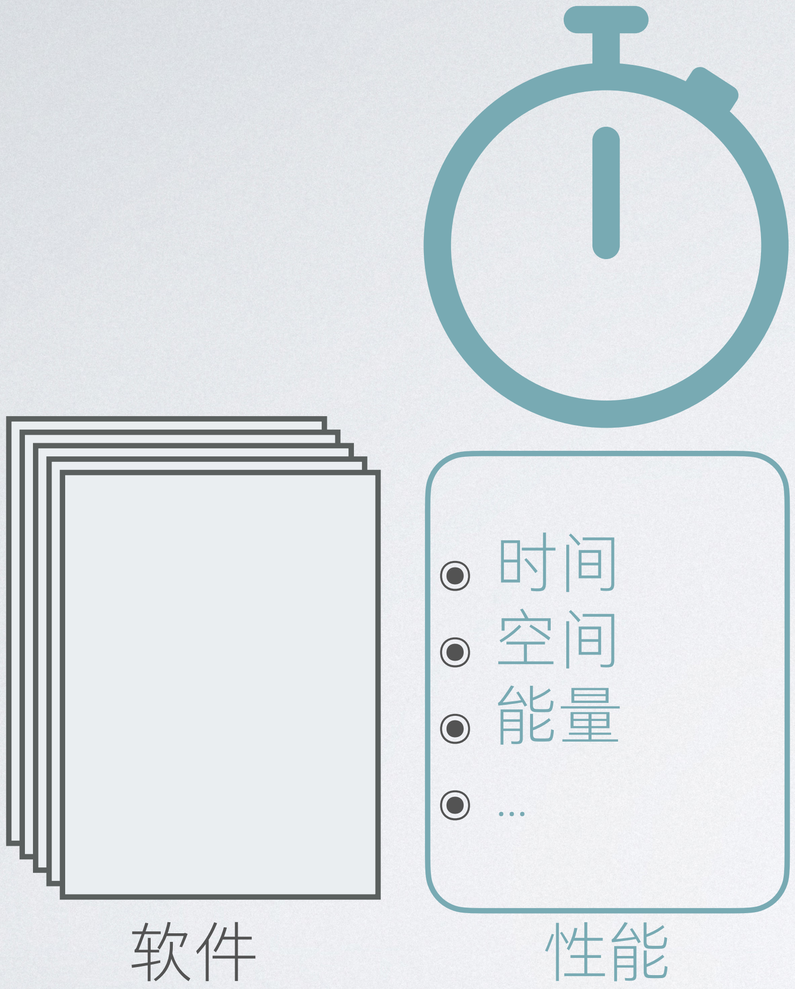
软件的资源消耗



应用场景

- 分析算法的时、空复杂度
- 预测智能合约的燃气消耗

软件的资源消耗



应用场景

- 分析算法的时、空复杂度
- 预测智能合约的燃气消耗
- 发现软件的拒绝服务漏洞

分析算法的时、空复杂度

该案例来源于静态分析工具 Infer 的官方文档：https://fbinfer.com/docs/next/checker-cost/.

分析算法的时、空复杂度

该案例来源于静态分析工具 Infer 的官方文档：https://fbinfer.com/docs/next/checker-cost/.

分析算法的时、空复杂度

```java
void loop(ArrayList<Integer> list) {
  for (int i = 0; i <= list.size(); i++) {
  }
}
```

该案例来源于静态分析工具 Infer 的官方文档：https://fbinfer.com/docs/next/checker-cost/.

分析算法的时、空复杂度

```
void loop(ArrayList<Integer> list) {
  for (int i = 0; i <= list.size(); i++) {
  }
}
```

$$8|list| + 16 = O(|list|)$$

该案例来源于静态分析工具 Infer 的官方文档：https://fbinfer.com/docs/next/checker-cost/.

分析算法的时、空复杂度

```
void loop(ArrayList<Integer> list) {
  for (int i = 0; i <= list.size(); i++) {
  }
}
```

$8|list| + 16 = O(|list|)$

```
void loop(ArrayList<Integer> list) {
  for (int i = 0; i <= list.size(); i++) {
    print(list); // new function call
  }
}
```

该案例来源于静态分析工具 Infer 的官方文档：https://fbinfer.com/docs/next/checker-cost/.

分析算法的时、空复杂度

```
void loop(ArrayList<Integer> list) {
  for (int i = 0; i <= list.size(); i++) {
  }
}
```

$8|list| + 16 = O(|list|)$

```
void loop(ArrayList<Integer> list) {
  for (int i = 0; i <= list.size(); i++) {
    print(list); // new function call
  }
}
```

$O(|list|^2)$

该案例来源于静态分析工具 Infer 的官方文档：https://fbinfer.com/docs/next/checker-cost/.

分析算法的时、空复杂度

```java
void loop(ArrayList<Integer> list) {
  for (int i = 0; i <= list.size(); i++) {
  }
}
```

```java
void loop(ArrayList<Integer> list) {
  for (int i = 0; i <= list.size(); i++) {
    print(list); // new function call
  }
}
```

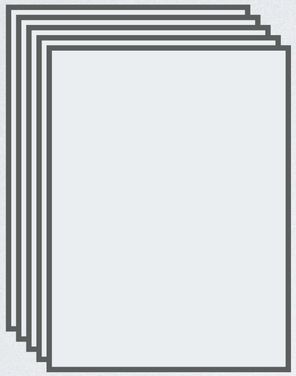
$8|list| + 16 = O(|list|)$

时间复杂度变高了！

$O(|list|^2)$

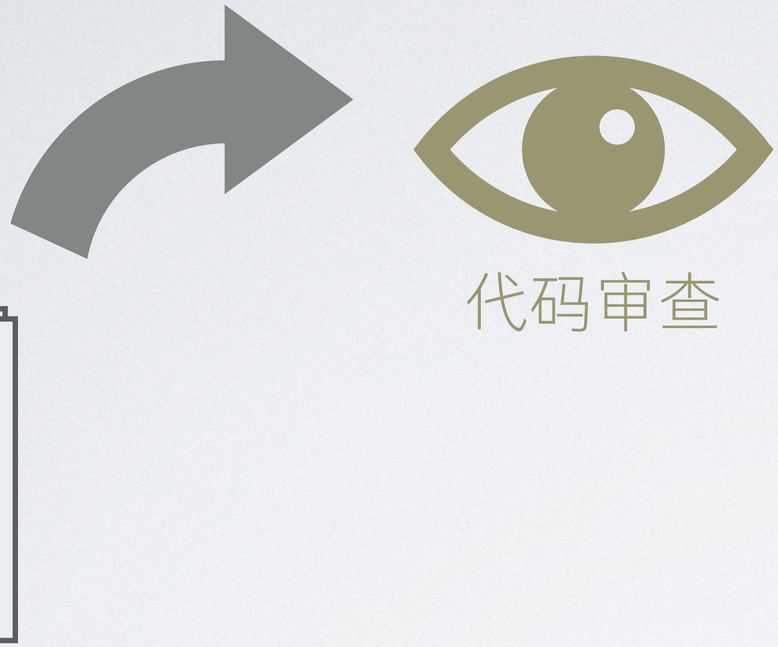
该案例来源于静态分析工具 Infer 的官方文档：https://fbinfer.com/docs/next/checker-cost/.

资源分析的时机

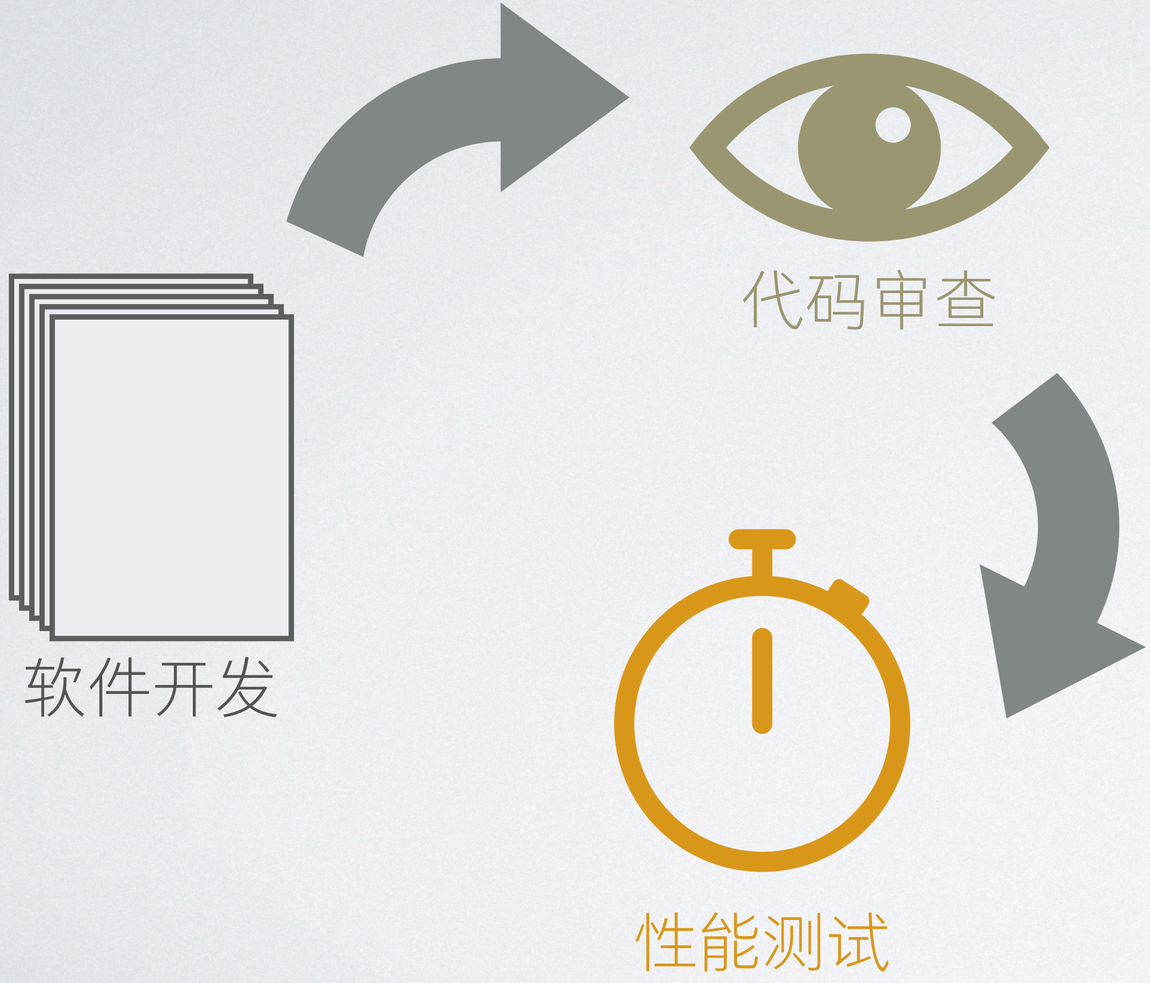


软件开发

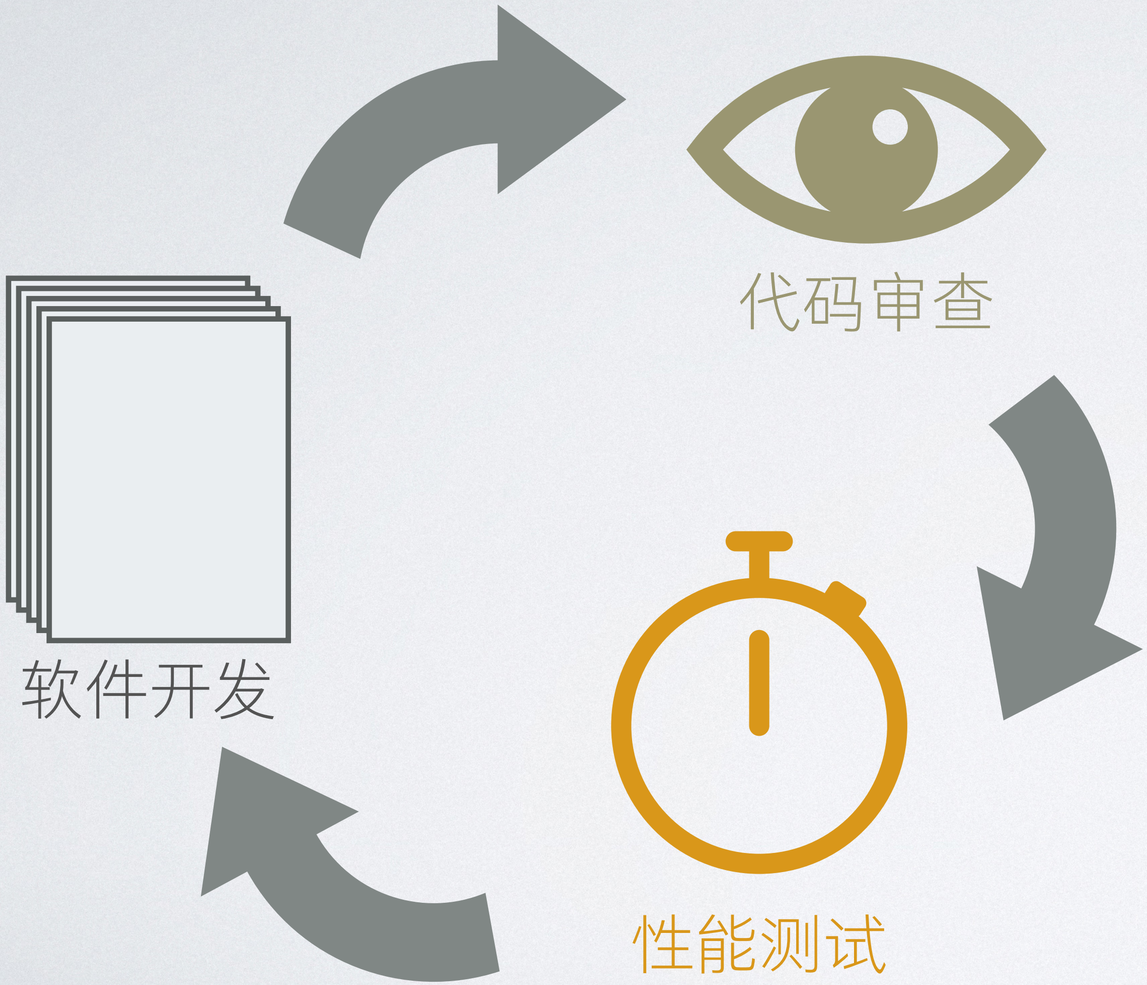
资源分析的时机



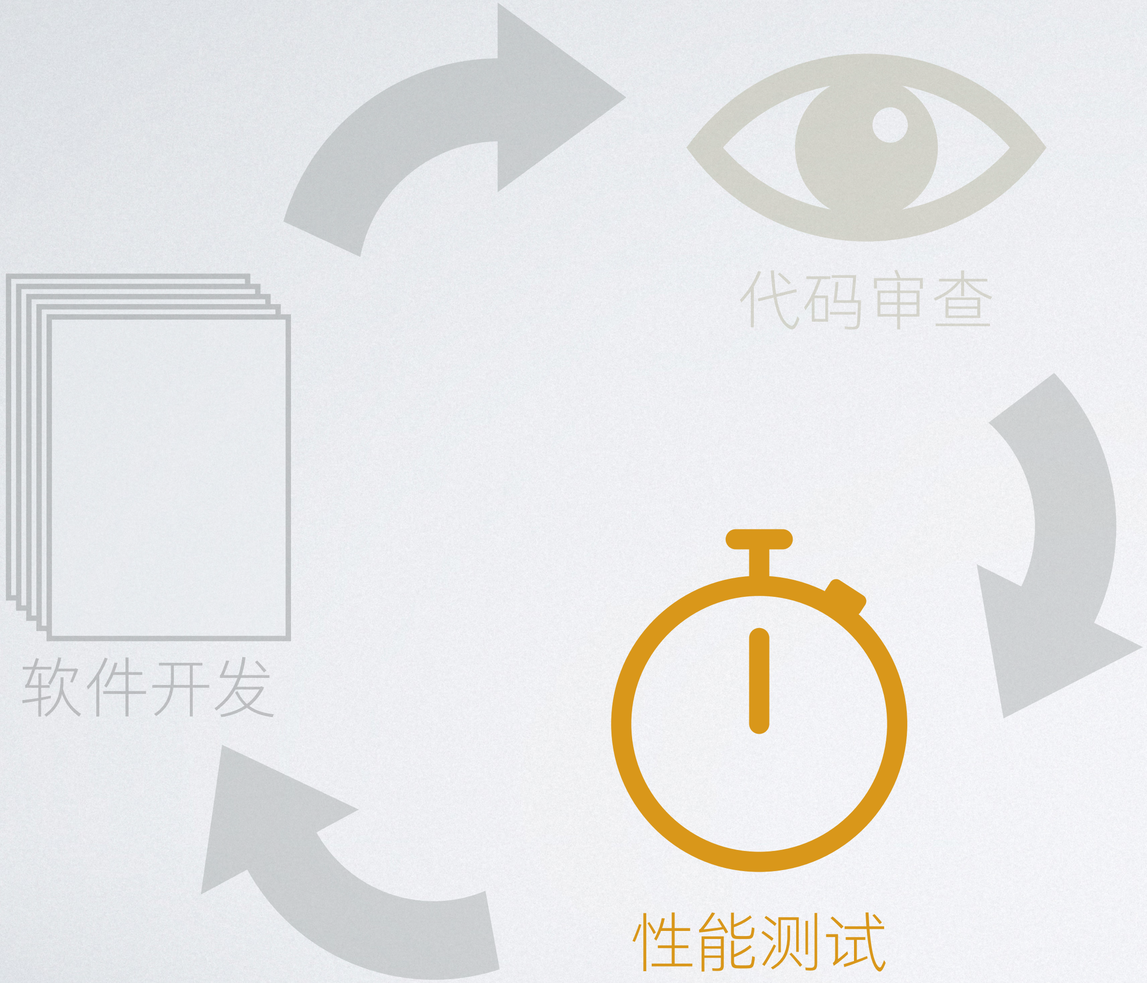
资源分析的时机



资源分析的时机



资源分析的时机



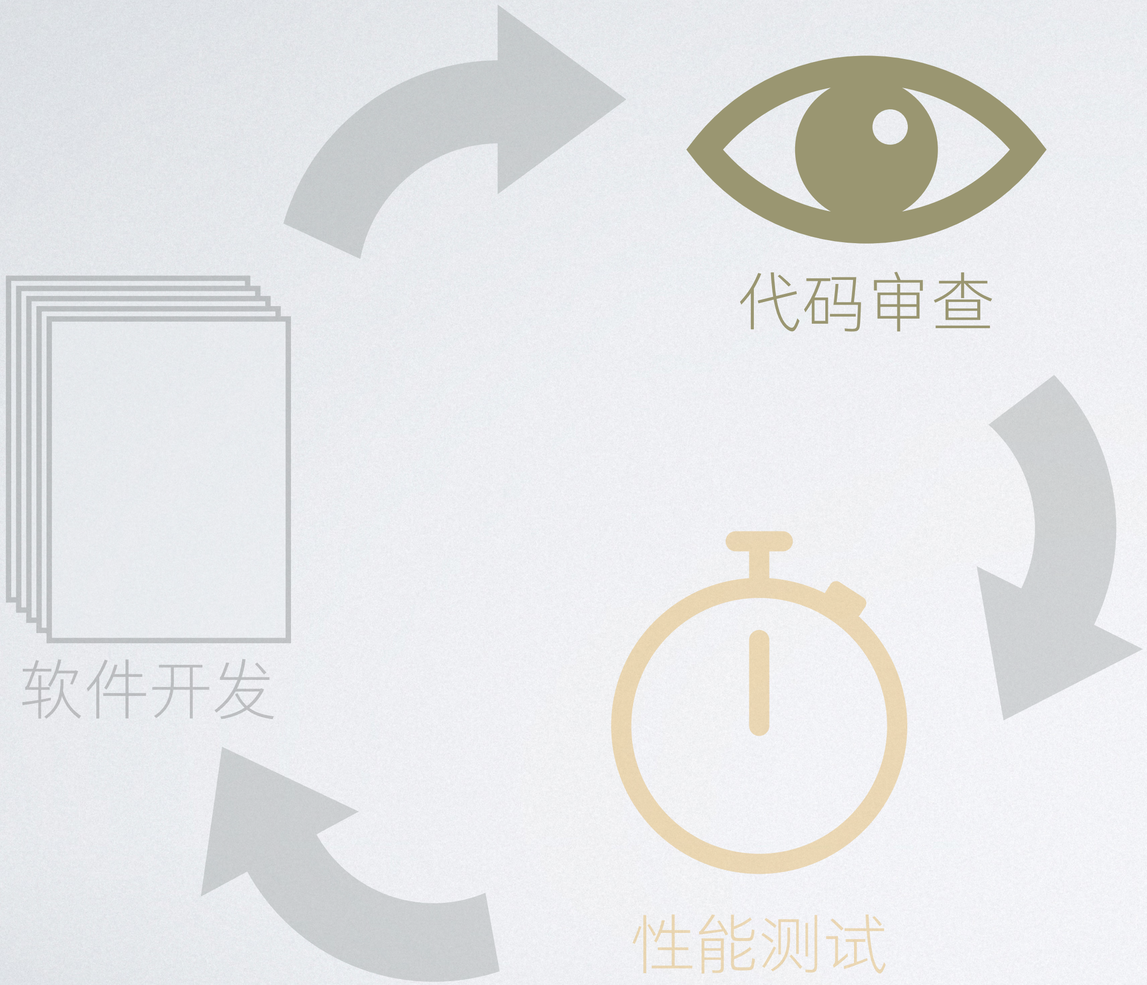
性能测试的优点：

- 实用性较强，对语言基本没要求
- 可以得到具体的测试用例

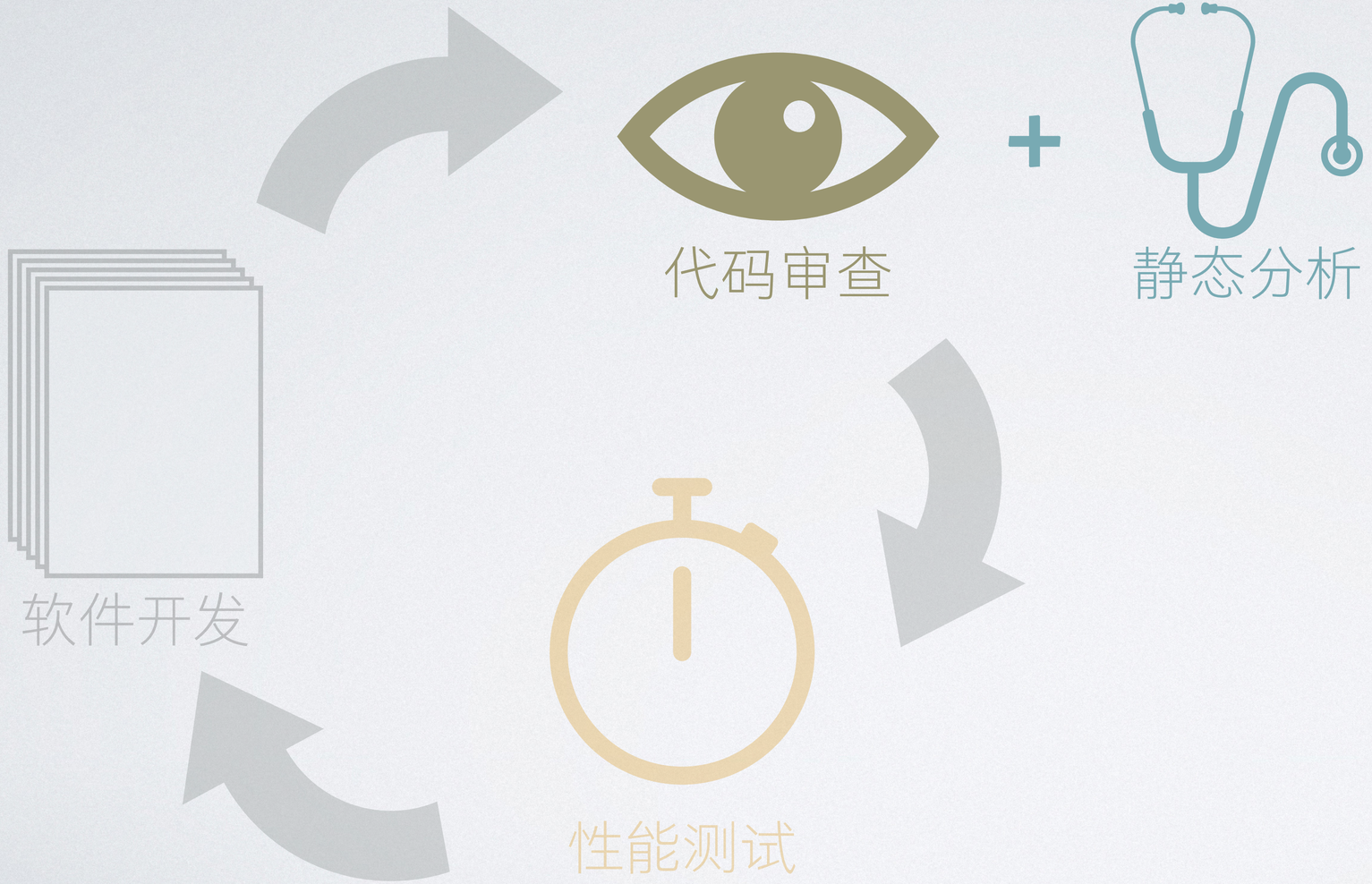
性能测试的缺点：

- 测试覆盖率不全
- 测试会进行较长的时间

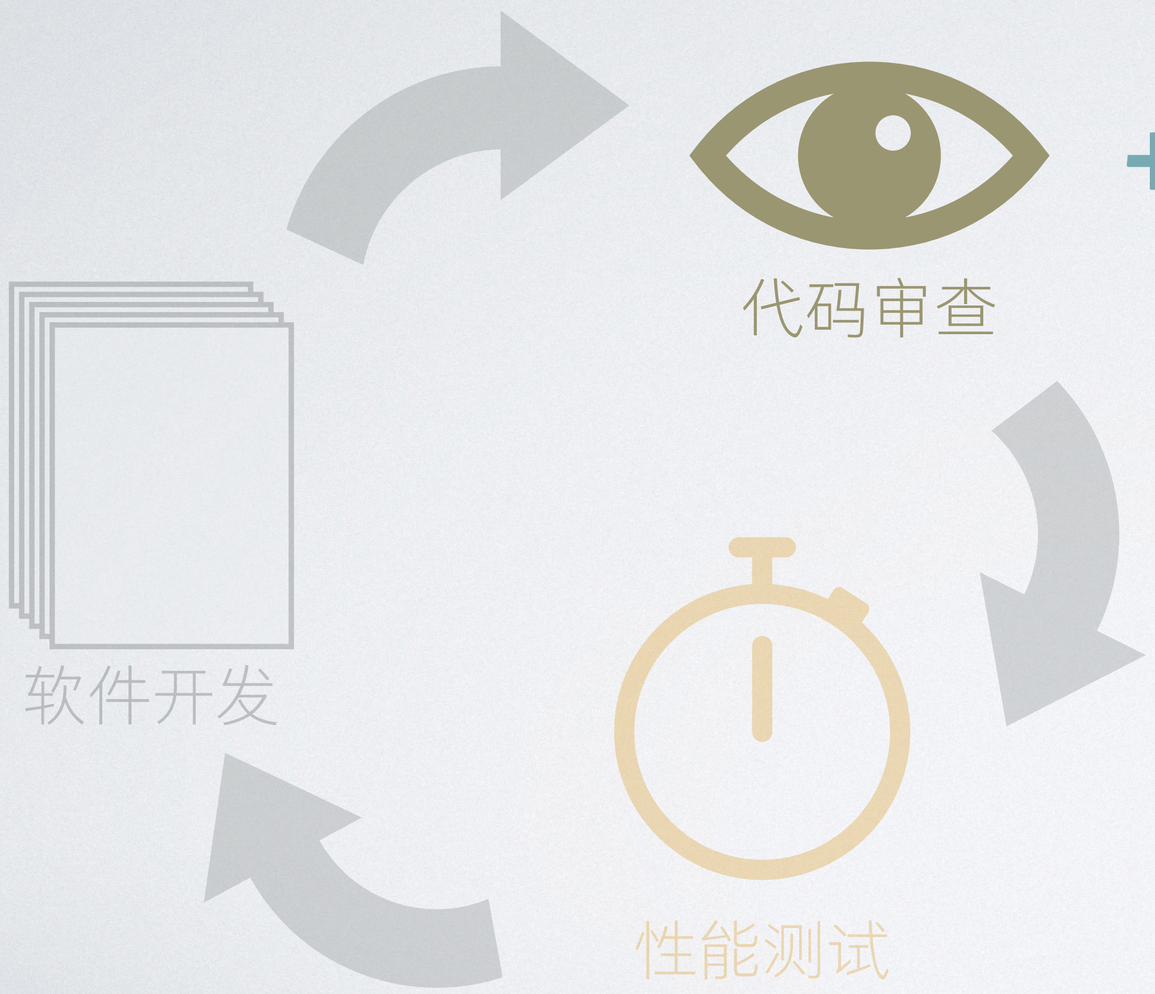
资源分析的时机



资源分析的时机



资源分析的时机



静态分析的优点：

- 分析结果考虑了所有可能的输入情况
- 可以通过模块化提高分析效率

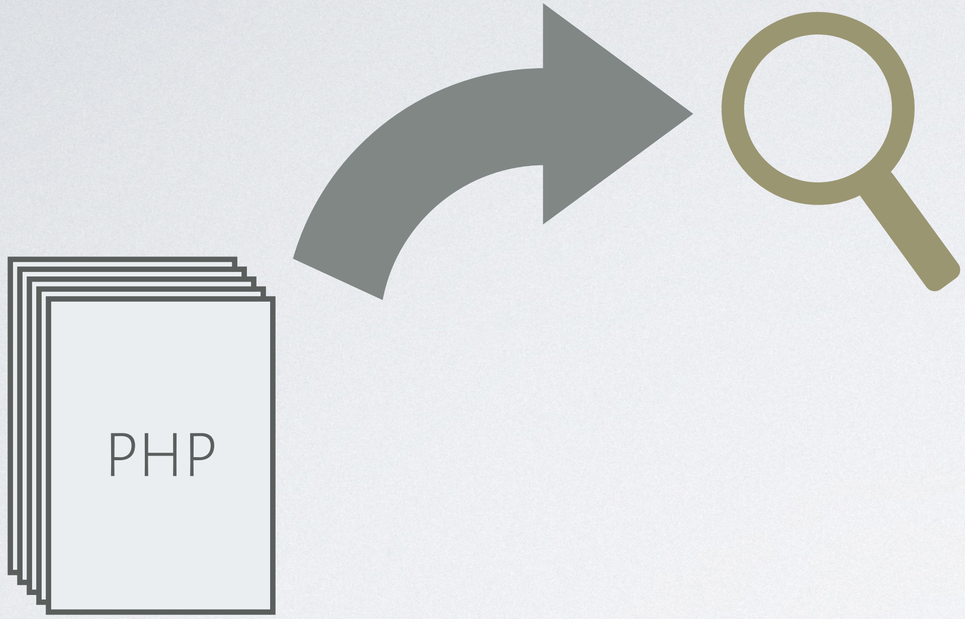
静态分析的缺点：

- 需要语言提供支持，对语言特性有要求
- 通常只能求出上界，可能导致假阳性

发现软件的拒绝服务漏洞

[1] CVE - CVE-2011-4885: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-4885.
[2] PHP 5.3.8 - Hashtables Denial of Service: https://www.exploit-db.com/exploits/18296/.
[3] PHP: PHP 5 ChangeLog: http://www.php.net/ChangeLog-5.php#5.3.9.

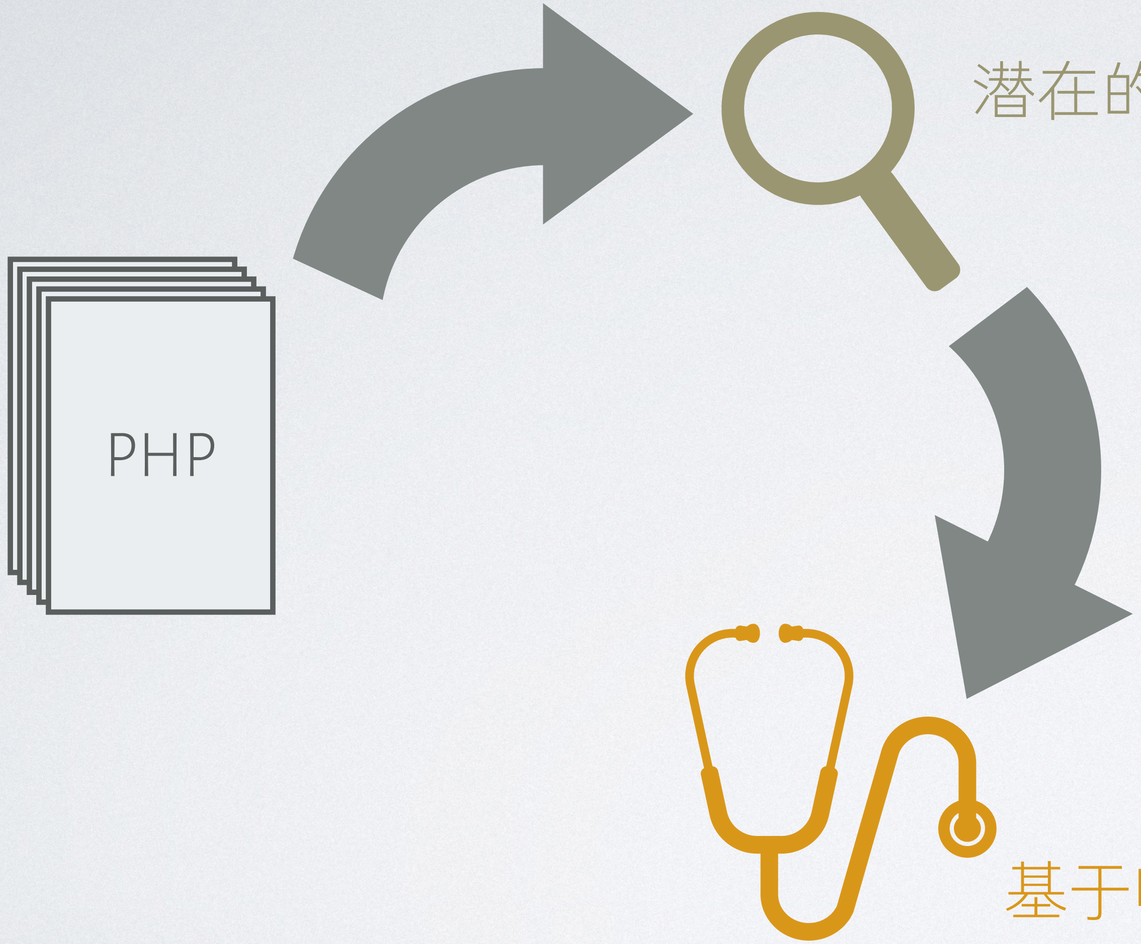
发现软件的拒绝服务漏洞



潜在的拒绝服务漏洞[1]

[1] CVE - CVE-2011-4885: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-4885.
[2] PHP 5.3.8 - Hashtables Denial of Service: https://www.exploit-db.com/exploits/18296/.
[3] PHP: PHP 5 ChangeLog: http://www.php.net/ChangeLog-5.php#5.3.9.

发现软件的拒绝服务漏洞

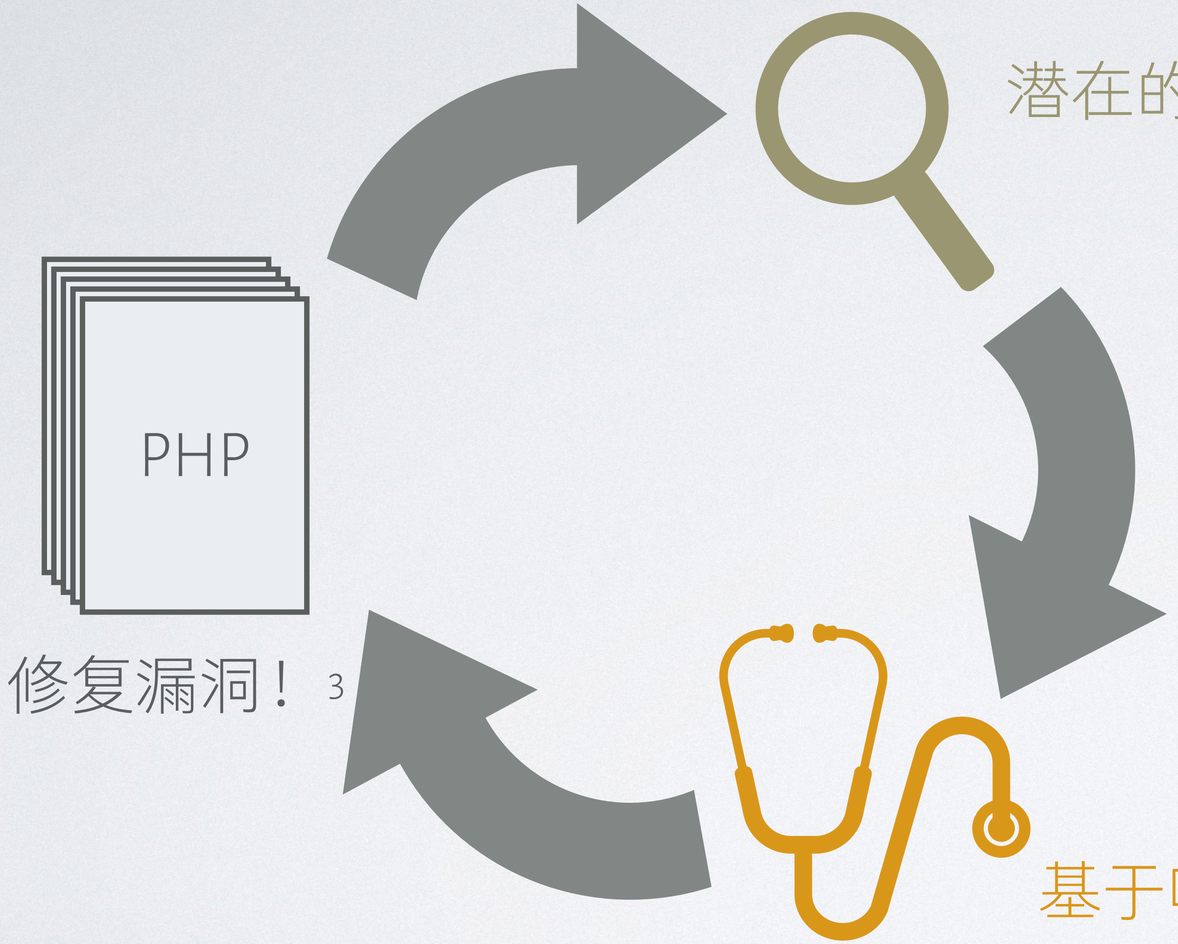


潜在的拒绝服务漏洞[1]

基于哈希冲突的具体攻击实例[2]

[1] CVE - CVE-2011-4885: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-4885.
[2] PHP 5.3.8 - Hashtables Denial of Service: https://www.exploit-db.com/exploits/18296/.
[3] PHP: PHP 5 ChangeLog: http://www.php.net/ChangeLog-5.php#5.3.9.

发现软件的拒绝服务漏洞



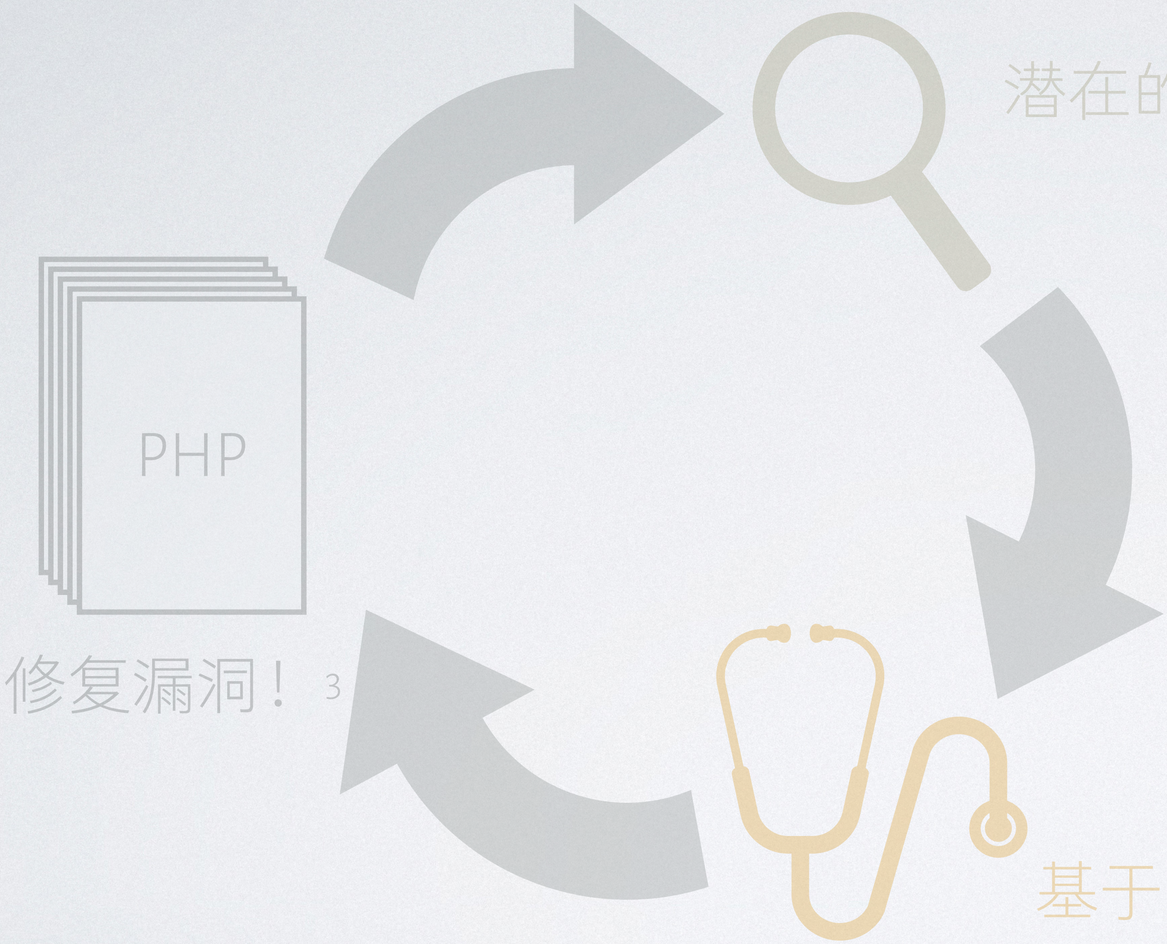
潜在的拒绝服务漏洞[1]

基于哈希冲突的具体攻击实例[2]

修复漏洞！[3]

[1] CVE - CVE-2011-4885: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-4885.
[2] PHP 5.3.8 - Hashtables Denial of Service: https://www.exploit-db.com/exploits/18296/.
[3] PHP: PHP 5 ChangeLog: http://www.php.net/ChangeLog-5.php#5.3.9.

发现软件的拒绝服务漏洞



一类拒绝服务漏洞与资源消耗相关：
比如不良哈希函数导致时间复杂度从线性退化到平方

[1] CVE - CVE-2011-4885: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-4885.
[2] PHP 5.3.8 - Hashtables Denial of Service: https://www.exploit-db.com/exploits/18296/.
[3] PHP: PHP 5 ChangeLog: http://www.php.net/ChangeLog-5.php#5.3.9.

资源安全编程语言

资源安全编程语言

- Rust 的流行说明了在语言设计中追踪更多的安全性质对系统编程是有益的

资源安全编程语言

- Rust 的流行说明了在语言设计中追踪更多的安全性质对系统编程是有益的
- 如何从资源安全的需求出发，改进 Rust 或者设计新的系统编程语言？

资源安全编程语言

- Rust 的流行说明了在语言设计中追踪更多的安全性质对系统编程是有益的
- 如何从资源安全的需求出发，改进 Rust 或者设计新的系统编程语言？
- “资源安全”涉及三方面：

资源安全编程语言

- Rust 的流行说明了在语言设计中追踪更多的安全性质对系统编程是有益的
- 如何从资源安全的需求出发，改进 Rust 或者设计新的系统编程语言？
- “资源安全”涉及三方面：

算法复杂度符合设计

资源安全编程语言

- Rust 的流行说明了在语言设计中追踪更多的安全性质对系统编程是有益的
- 如何从资源安全的需求出发，改进 Rust 或者设计新的系统编程语言？
- “资源安全”涉及三方面：

算法复杂度符合设计　　编译过程优化资源消耗

资源安全编程语言

- Rust 的流行说明了在语言设计中追踪更多的安全性质对系统编程是有益的
- 如何从资源安全的需求出发，改进 Rust 或者设计新的系统编程语言？
- “资源安全”涉及三方面：

算法复杂度符合设计　编译过程优化资源消耗　没有资源相关安全漏洞

工作1：资源制导最坏情况输入生成

D. Wang and J. Hoffmann. Type-Guided Worst-Case Input Generation. In *POPL'19*.

工作1：资源制导最坏情况输入生成

D. Wang and J. Hoffmann. Type-Guided Worst-Case Input Generation. In *POPL'19*.

基于势能的自动资源分析

指导

基于符号执行的最坏情况输入生成

工作1：资源制导最坏情况输入生成

D. Wang and J. Hoffmann. Type-Guided Worst-Case Input Generation. In *POPL'19*.

基于势能的自动资源分析

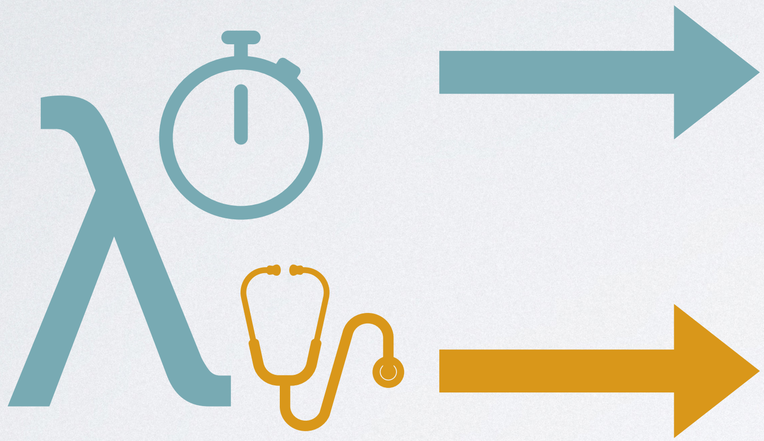
指导

基于符号执行的最坏情况输入生成

首个有理论正确性保证的，基于静态资源分析的最坏情况输入生成算法

工作1：资源制导最坏情况输入生成

D. Wang and J. Hoffmann. Type-Guided Worst-Case Input Generation. In *POPL'19*.



基于势能的自动资源分析

指导

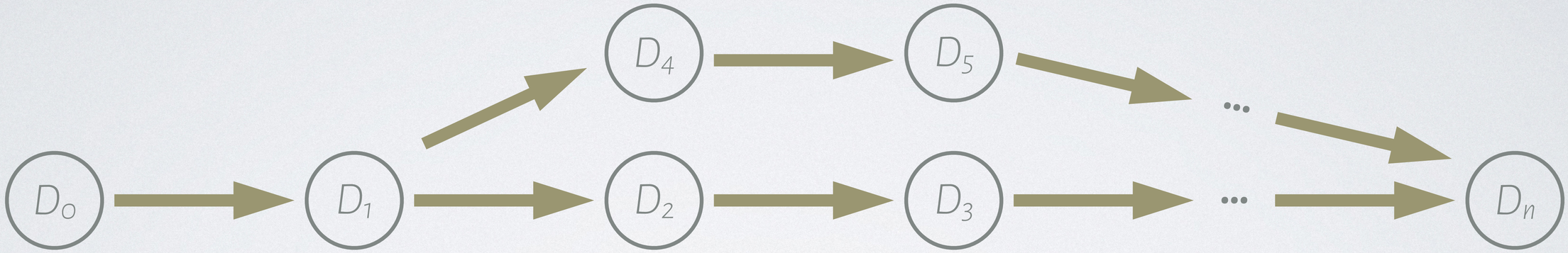
基于符号执行的最坏情况输入生成

如果一条程序执行路径没有任何势能浪费，那么它一定具有最坏的资源消耗

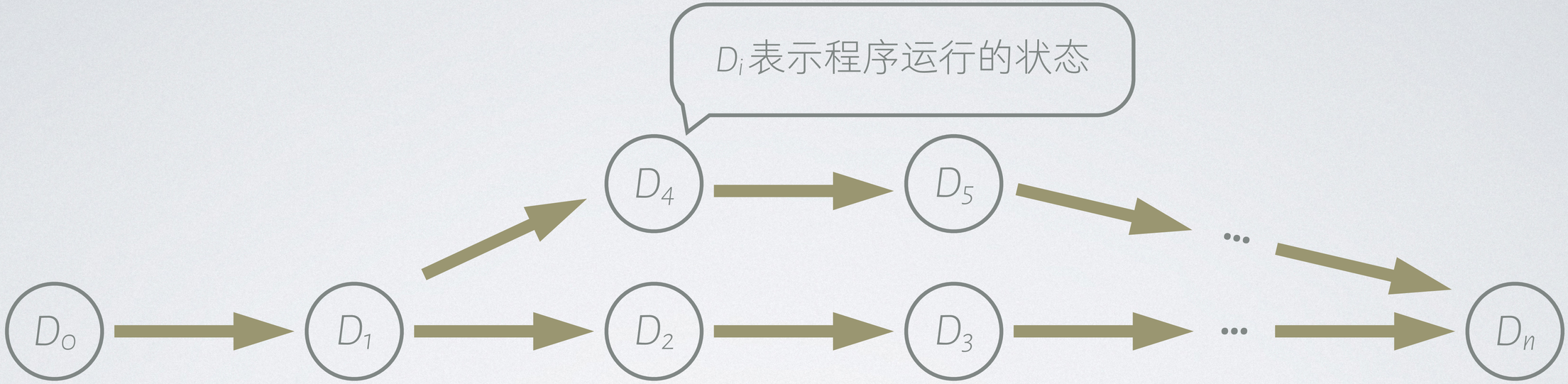
首个有理论正确性保证的，基于静态资源分析的最坏情况输入生成算法

基于势能的自动资源分析

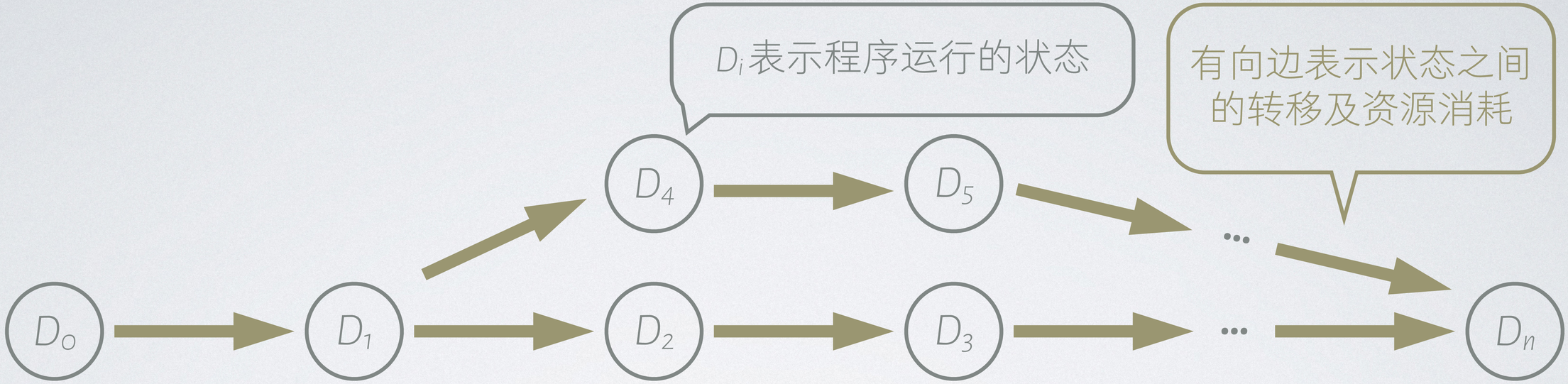
基于势能的自动资源分析



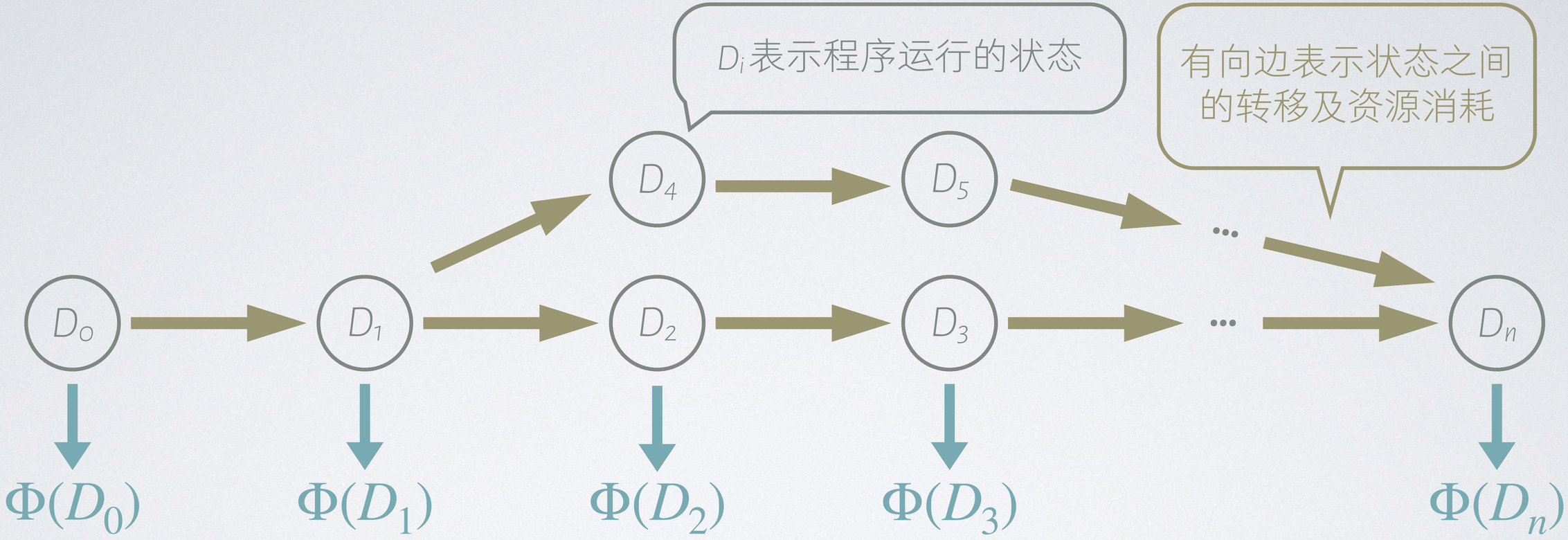
基于势能的自动资源分析



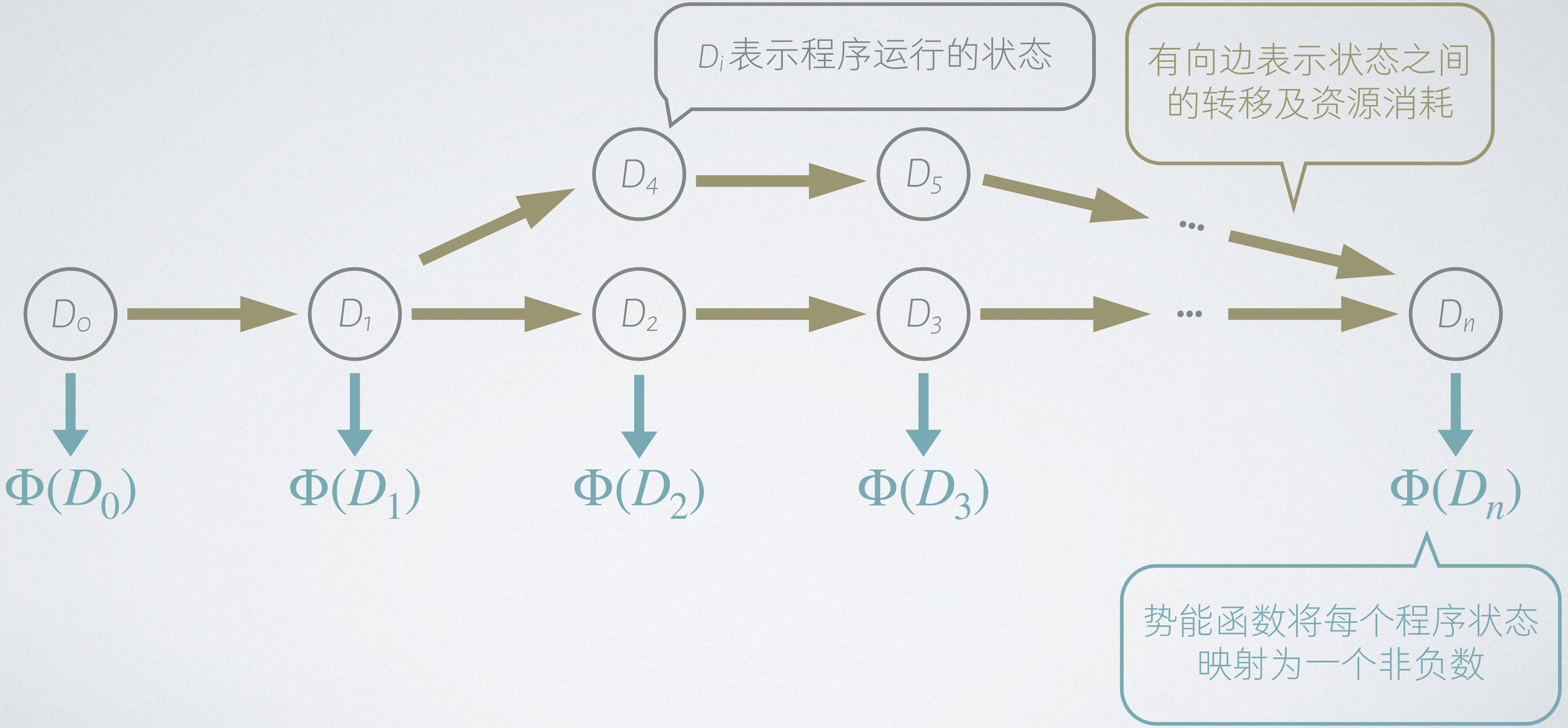
基于势能的自动资源分析



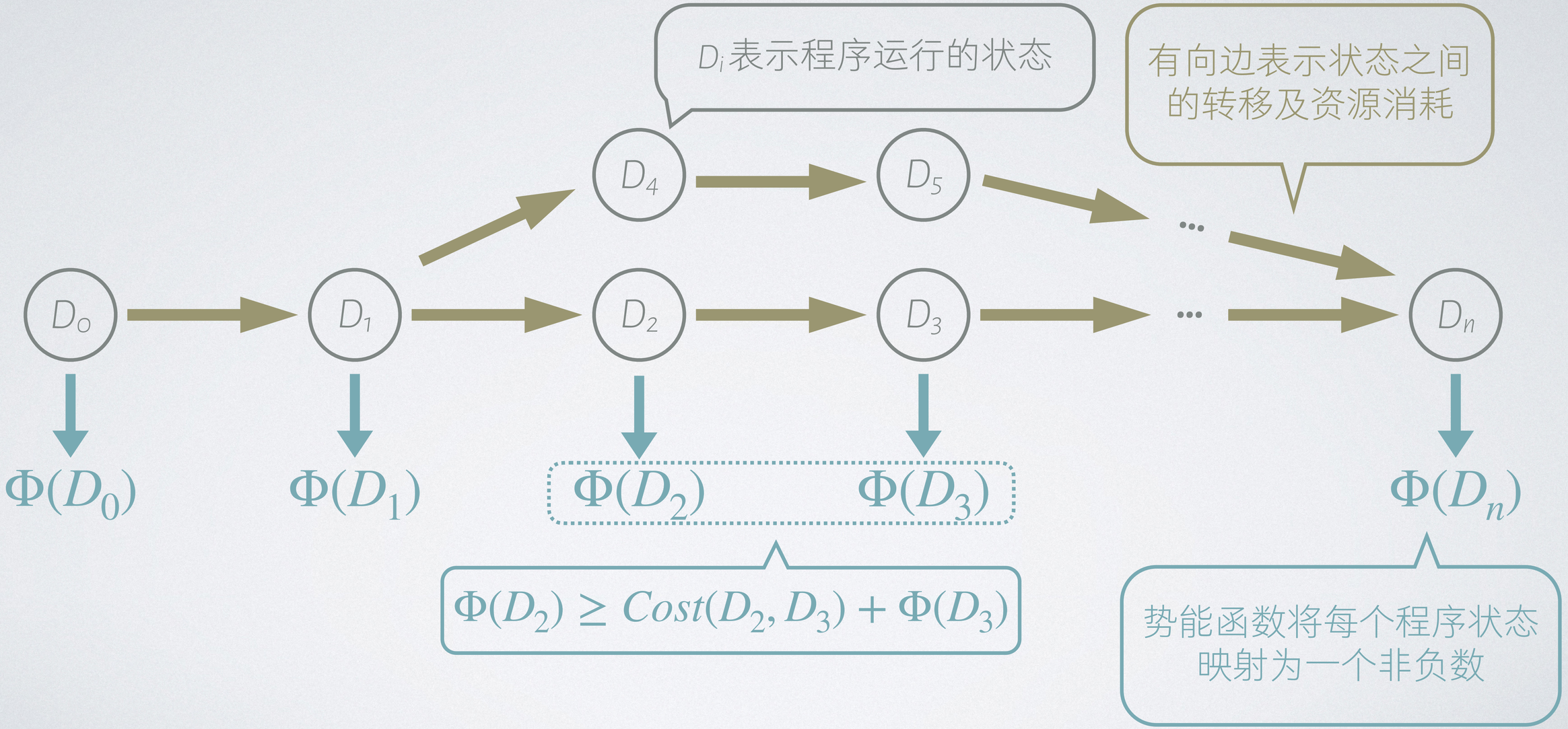
基于势能的自动资源分析



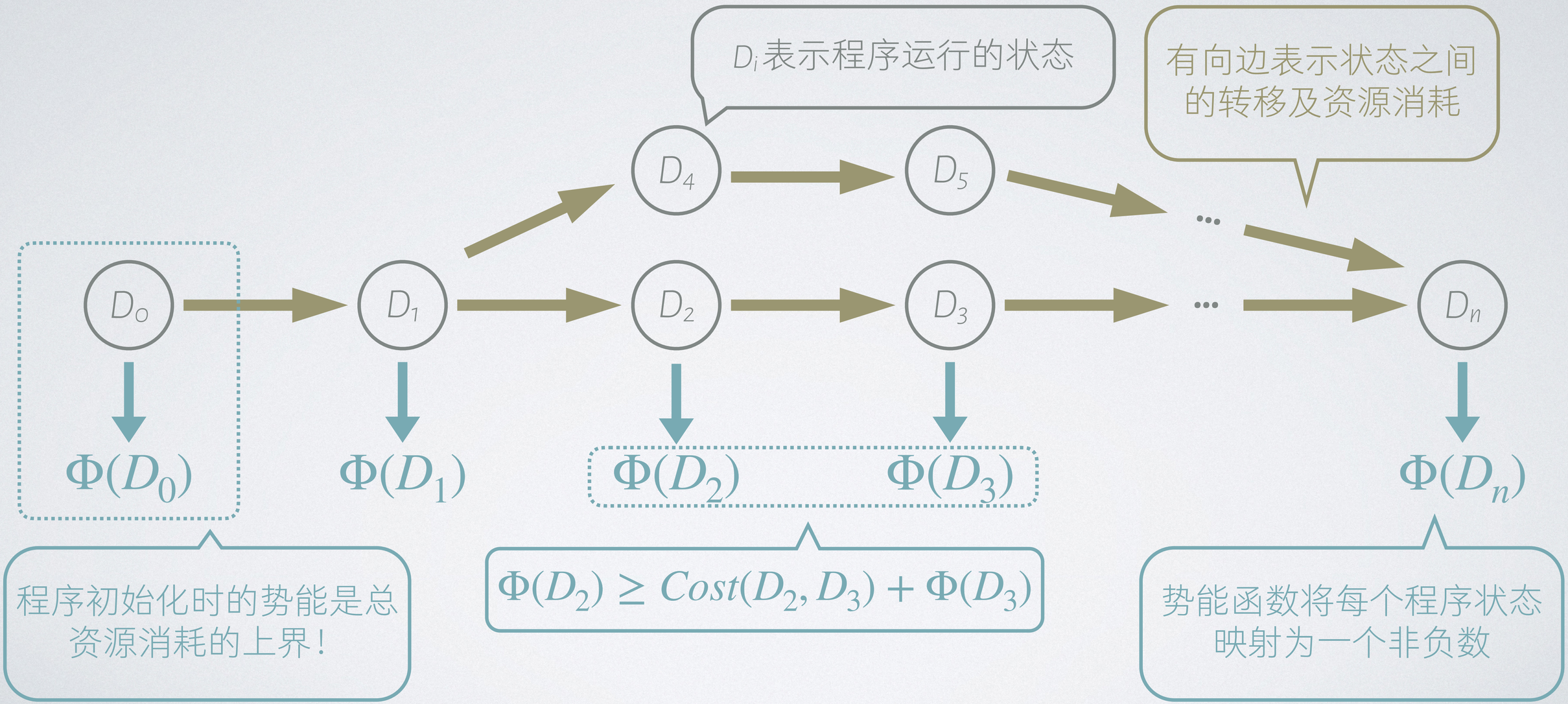
基于势能的自动资源分析



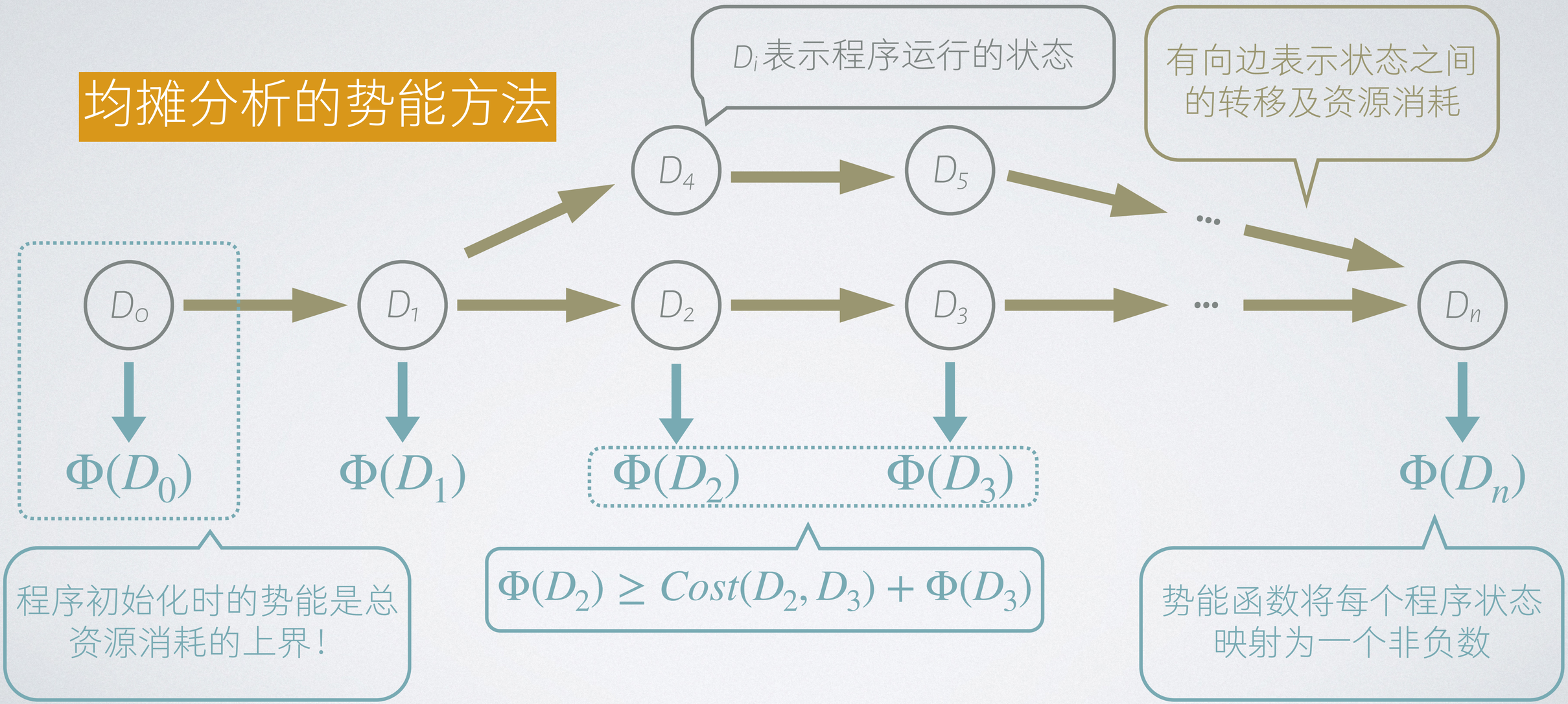
基于势能的自动资源分析



基于势能的自动资源分析



基于势能的自动资源分析



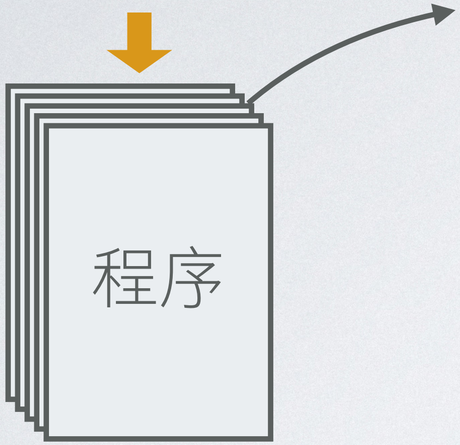
基于符号执行的最坏情况测试生成

基于符号执行的最坏情况测试生成

输入的规约
（例如列表长度）

基于符号执行的最坏情况测试生成

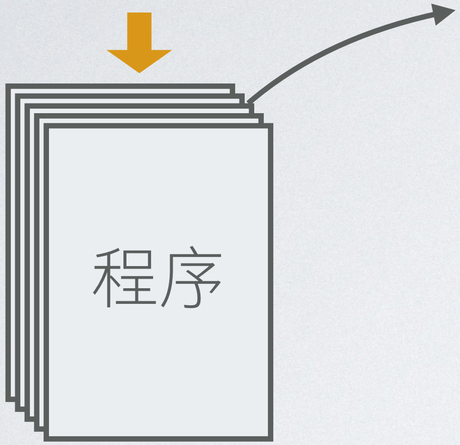
输入的规约
（例如列表长度）



可能的执行路径1

基于符号执行的最坏情况测试生成

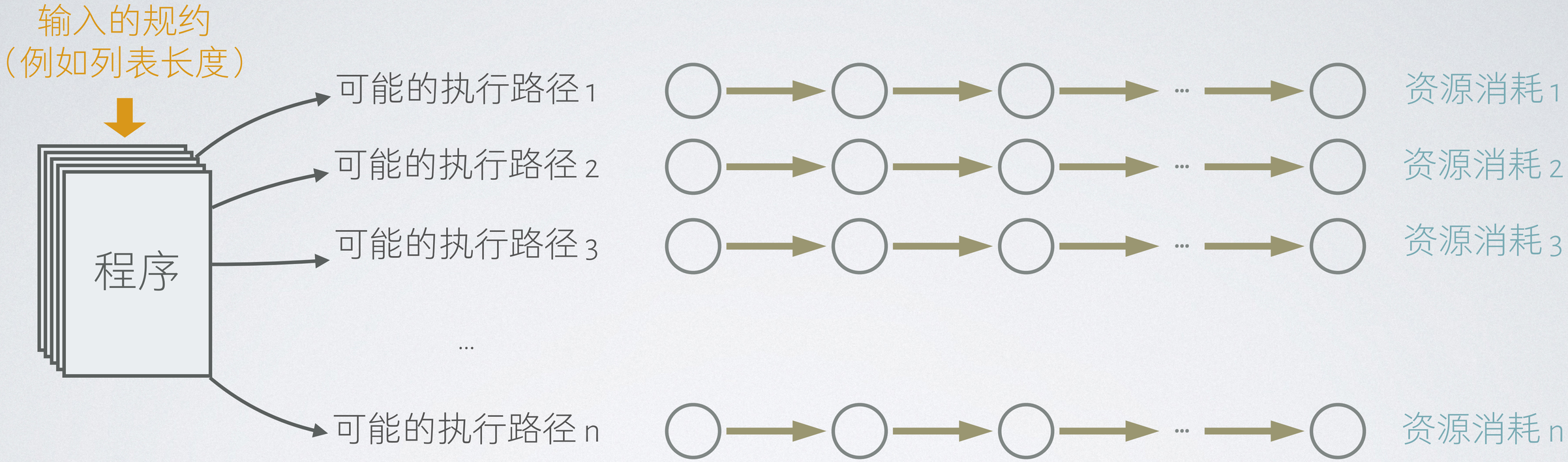
输入的规约
（例如列表长度）



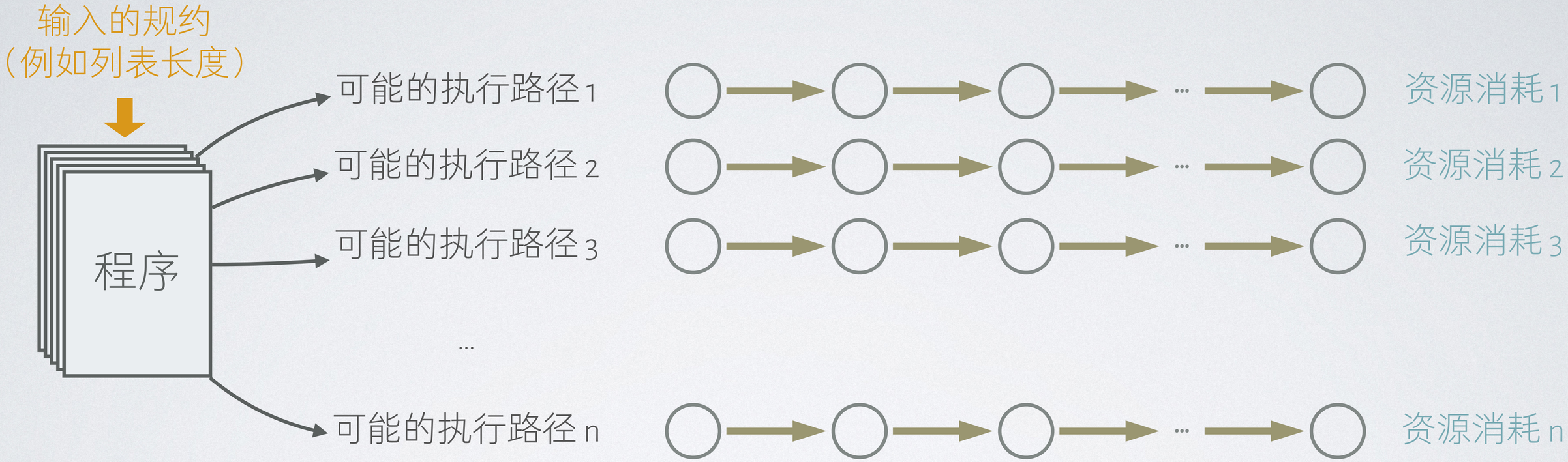
可能的执行路径1

资源消耗1

基于符号执行的最坏情况测试生成

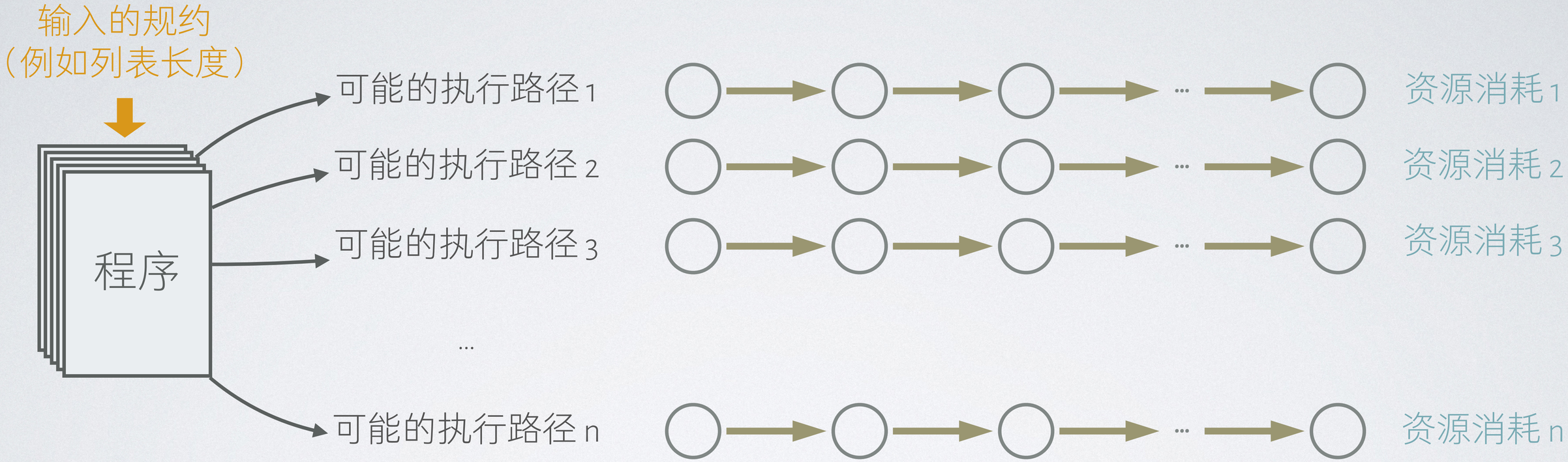


基于符号执行的最坏情况测试生成



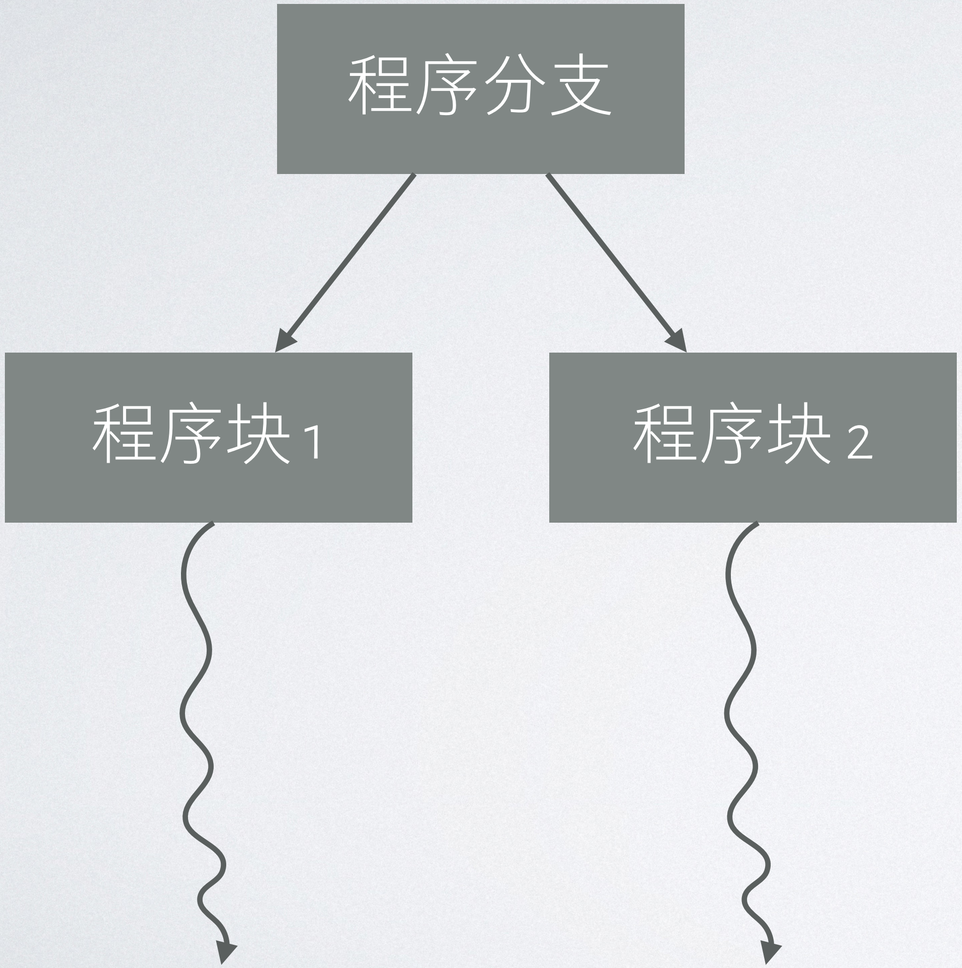
- 给定一个输入的规约，搜索一条具有最大资源消耗的程序执行路径

基于符号执行的最坏情况测试生成

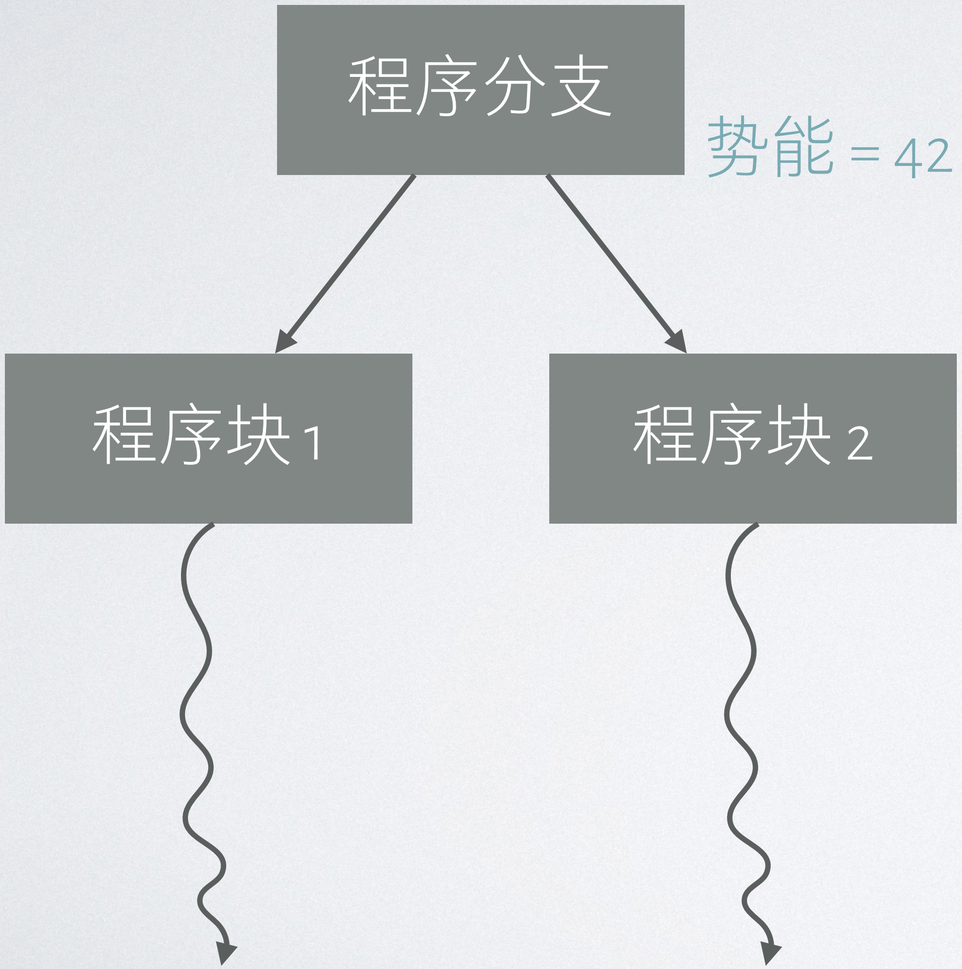


- 给定一个输入的规约，搜索一条具有最大资源消耗的程序执行路径
- 是否可以利用静态资源分析的结果来指导符号执行的搜索过程呢？

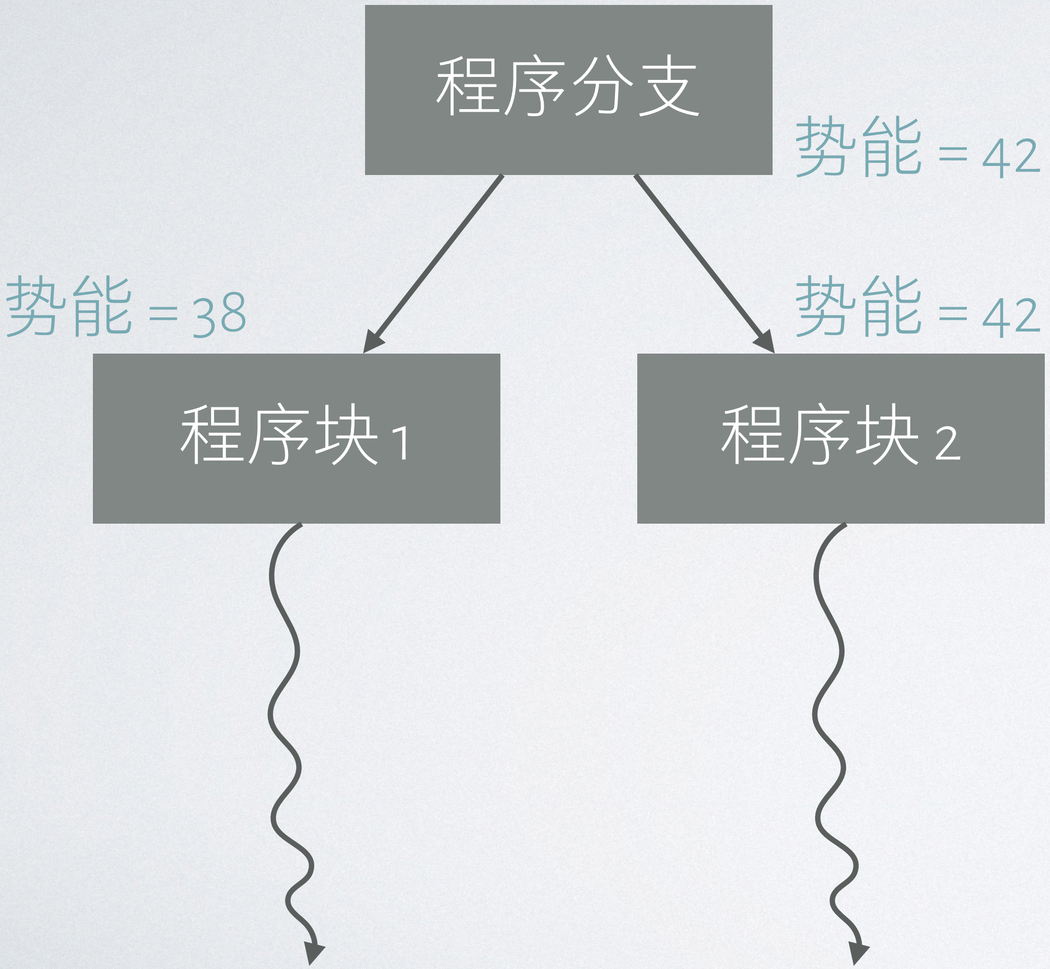
资源制导的符号执行



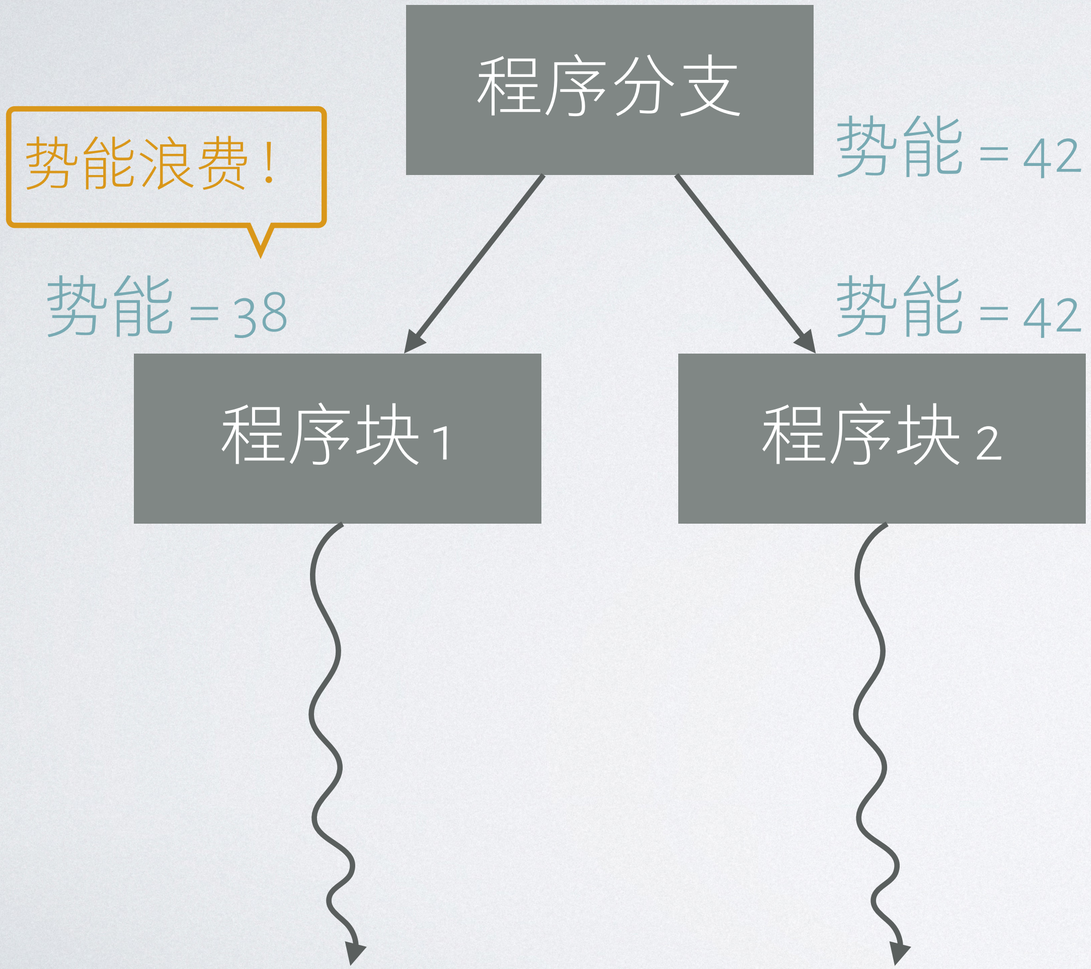
资源制导的符号执行



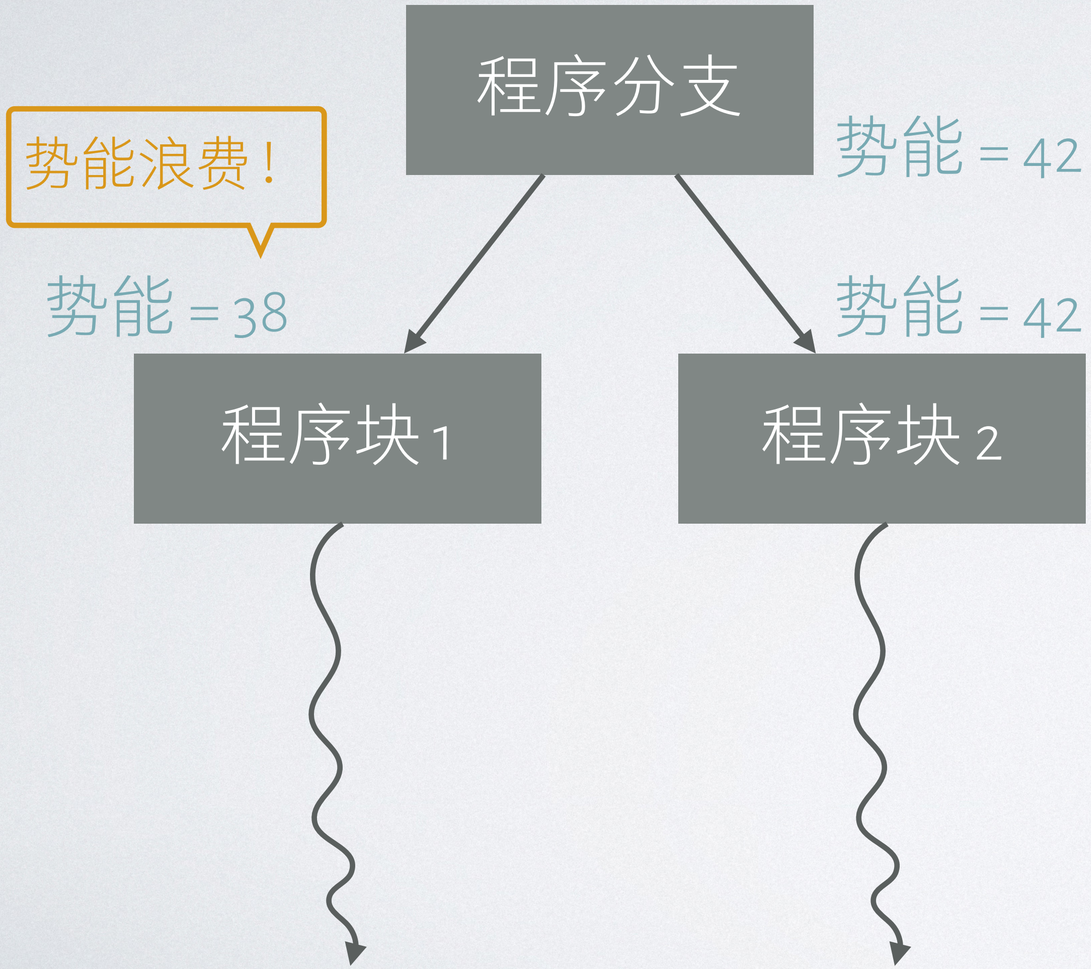
资源制导的符号执行



资源制导的符号执行

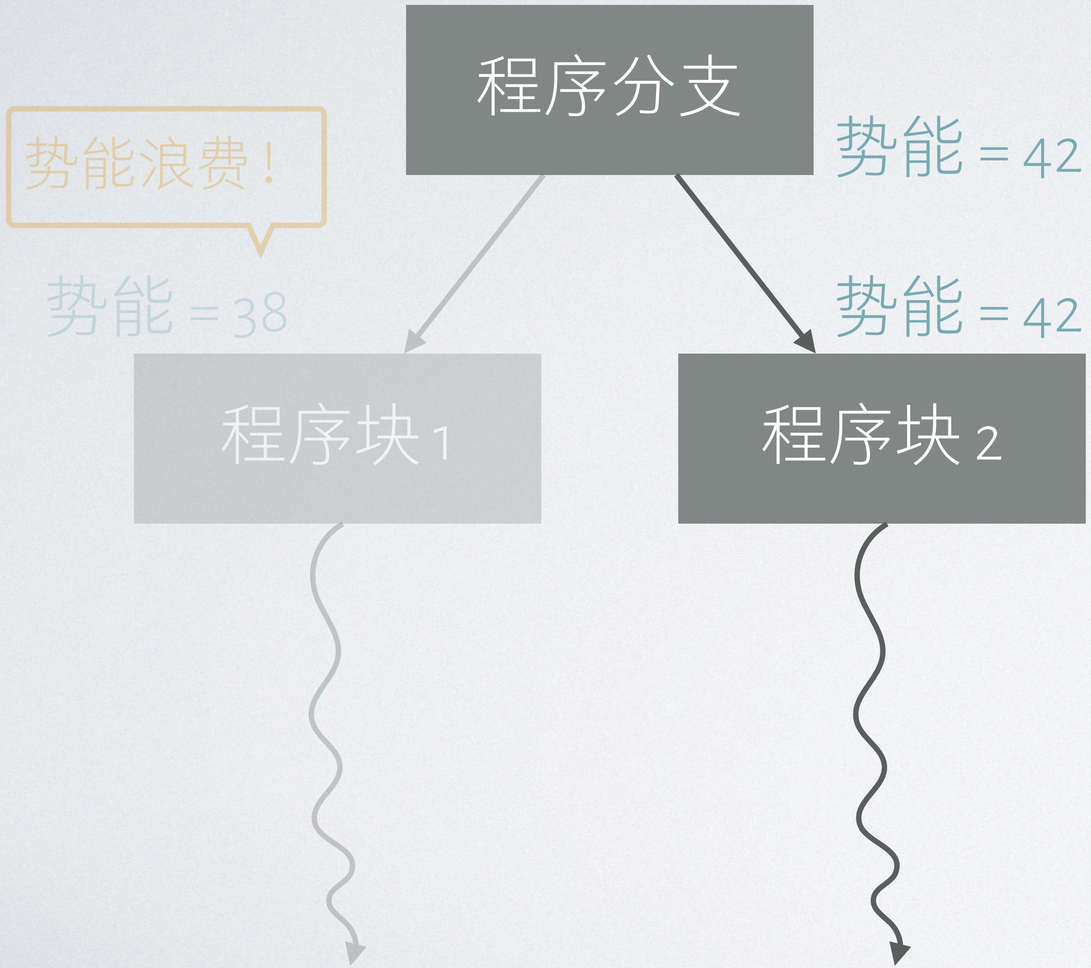


资源制导的符号执行



如果一条程序执行路径没有任何势能浪费，那么它一定具有最坏的资源消耗

资源制导的符号执行



如果一条程序执行路径没有任何势能浪费，那么它一定具有最坏的资源消耗

通过这个信息减少搜索量！

工作 2：通过转译分析 Rust 程序的资源消耗

X. Peng and **D. Wang**. Rust Resource Analysis by Functional Translation. *Working Paper*.

[1] S. Ho and J. Protzenko. 2022. AENEAS: Rust Verification by Functional Translation. In ICFP'22.

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

工作 2：通过转译分析 Rust 程序的资源消耗

X. Peng and **D. Wang**. Rust Resource Analysis by Functional Translation. *Working Paper*.


```
fn list_consume(l: &mut List) {
    match l {
        Nil => {}
        Cons(_, tl) => {
            tick<1>();
            list_consume(tl);
        }
    }
    *l = Nil;
}

fn list_consume_twice(l: &mut List) {
    list_consume(l);
    list_consume(l);
}
```

[1] S. Ho and J. Protzenko. 2022. AENEAS: Rust Verification by Functional Translation. In ICFP'22.

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

工作 2：通过转译分析 Rust 程序的资源消耗

X. Peng and **D. Wang**. Rust Resource Analysis by Functional Translation. *Working Paper*.


```
fn list_consume(l: &mut List) {
    match l {
        Nil => {}
        Cons(_, tl) => {
            tick<1>();
            list_consume(tl);
        }
    }
    *l = Nil;
}

fn list_consume_twice(l: &mut List) {
    list_consume(l);
    list_consume(l);
}
```

```
let rec list_consume_fwd_back l =
  match l with
  | [] -> []
  | _ :: tl ->
    let _ = Raml.tick 1.0 in
    let _ = list_consume_fwd_back tl in
    []

let list_consume_twice_fwd_back l =
  let l0 = list_consume_fwd_back l in
  list_consume_fwd_back l0
```

[1] S. Ho and J. Protzenko. 2022. AENEAS: Rust Verification by Functional Translation. In ICFP'22.

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

工作 2：通过转译分析 Rust 程序的资源消耗

X. Peng and **D. Wang**. Rust Resource Analysis by Functional Translation. *Working Paper*.


```
fn list_consume(l: &mut List) {
    match l {
        Nil => {}
        Cons(_, tl) => {
            tick<1>();
            list_consume(tl);
        }
    }
    *l = Nil;
}

fn list_consume_twice(l: &mut List) {
    list_consume(l);
    list_consume(l);
}
```

```
let rec list_consume_fwd_back l =
  match l with
  | [] -> []
  | _ :: tl ->
    let _ = Raml.tick 1.0 in
    let _ = list_consume_fwd_back tl in
    []

let list_consume_twice_fwd_back l =
  let l0 = list_consume_fwd_back l in
  list_consume_fwd_back l0
```

```
== list_consume_twice_fwd_back :

Simplified bound:
 1*M
where
 M is the number of ::-nodes of the argument
```

[1] S. Ho and J. Protzenko. 2022. AENEAS: Rust Verification by Functional Translation. In ICFP'22.

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

工作 2：通过转译分析 Rust 程序的资源消耗

X. Peng and **D. Wang**. Rust Resource Analysis by Functional Translation. *Working Paper*.


```
fn list_consume(l: &mut List) {
    match l {
        Nil => {}
        Cons(_, tl) => {
            tick<1>();
            list_consume(tl);
        }
    }
    *l = Nil;
}

fn list_consume_twice(l: &mut List) {
    list_consume(l);
    list_consume(l);
}
```

```
let rec list_consume_fwd_back l =
  match l with
  | [] -> []
  | _ :: tl ->
    let _ = Raml.tick 1.0 in
    let _ = list_consume_fwd_back tl in
    []

let list_consume_twice_fwd_back l =
  let l0 = list_consume_fwd_back l in
  list_consume_fwd_back l0
```

```
== list_consume_twice_fwd_back :

Simplified bound:
 1*M
where
 M is the number of ::-nodes of the argument
```

- 这种方法的不足：
  - 对 Rust 特性覆盖不全面
  - 依赖 RAML 的分析能力
  - 不支持并发等场景

[1] S. Ho and J. Protzenko. 2022. AENEAS: Rust Verification by Functional Translation. In ICFP'22.

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

其它工作

其它工作

- 基于测试生成的动态资源分析
  - H. Wu and **D. Wang**. Worst-Case Analysis is Maximum-A-Posteriori Estimation. *Working Paper*.
  - Z. Chen and **D. Wang**. Path Fuzzing for Worst-Case Analysis. *Working Paper*.

其它工作

- 基于测试生成的动态资源分析
  - H. Wu and **D. Wang**. Worst-Case Analysis is Maximum-A-Posteriori Estimation. *Working Paper*.
  - Z. Chen and **D. Wang**. Path Fuzzing for Worst-Case Analysis. *Working Paper*.
- 基于势能方法的静态资源分析
  - **D. Wang**, D. M. Kahn, and J. Hoffmann. Raising Expectations: Automating Expected Cost Analysis with Types. In *ICFP'20*.
  - **D. Wang**, J. Hoffmann, and T. Reps. Central Moment Analysis for Cost Accumulators in Probabilistic Programs. In *PLDI'21*.
  - A. Das, **D. Wang**, and J. Hoffmann. Probabilistic Resource-Aware Session Types. In *POPL'23*.

其它工作

- 基于测试生成的动态资源分析
  - H. Wu and **D. Wang**. Worst-Case Analysis is Maximum-A-Posteriori Estimation. *Working Paper*.
  - Z. Chen and **D. Wang**. Path Fuzzing for Worst-Case Analysis. *Working Paper*.
- 基于势能方法的静态资源分析
  - **D. Wang**, D. M. Kahn, and J. Hoffmann. Raising Expectations: Automating Expected Cost Analysis with Types. In *ICFP'20*.
  - **D. Wang**, J. Hoffmann, and T. Reps. Central Moment Analysis for Cost Accumulators in Probabilistic Programs. In *PLDI'21*.
  - A. Das, **D. Wang**, and J. Hoffmann. Probabilistic Resource-Aware Session Types. In *POPL'23*.
- 从语言设计的角度，这些工作有什么启发？

Resource-Aware Programming

Resource-Aware Programming

- 类比 Rust：通过类型标注，让程序员写一些跟内存安全相关的性质！

Resource-Aware Programming

- 类比 Rust：通过类型标注，让程序员写一些跟内存安全相关的性质！
- 是否可以：通过资源标注，让程序员写一些跟资源消耗相关的性质？

Resource-Aware Programming

- 类比 Rust：通过类型标注，让程序员写一些跟内存安全相关的性质！
- 是否可以：通过资源标注，让程序员写一些跟资源消耗相关的性质？
- 从动态、静态资源分析的工作中得到的启发：

Resource-Aware Programming

- 类比 Rust：通过类型标注，让程序员写一些跟内存安全相关的性质！
- 是否可以：通过资源标注，让程序员写一些跟资源消耗相关的性质？
- 从动态、静态资源分析的工作中得到的启发：
  - 语言设计可能可以让我们绕过一些“写完代码再分析”的难题

Resource-Aware Programming

- 类比 Rust：通过类型标注，让程序员写一些跟内存安全相关的性质！
- 是否可以：通过资源标注，让程序员写一些跟资源消耗相关的性质？
- 从动态、静态资源分析的工作中得到的启发：
  - 语言设计可能可以让我们绕过一些“写完代码再分析”的难题
  - 在写代码时就考虑资源消耗可能更容易写出高效率的代码

工作 3：资源制导的程序合成

T. Knoth, **D. Wang**, N. Polikarpova, and J. Hoffmann. Resource-Guided Program Synthesis. In *PLDI'19*.

工作 3：资源制导的程序合成

T. Knoth, **D. Wang**, N. Polikarpova, and J. Hoffmann. Resource-Guided Program Synthesis. In *PLDI'19*.

- 虽然是程序合成的工作，但其核心是一个带资源标注类型的函数式语言

工作 3：资源制导的程序合成

T. Knoth, **D. Wang**, N. Polikarpova, and J. Hoffmann. Resource-Guided Program Synthesis. In *PLDI'19*.

- 虽然是程序合成的工作，但其核心是一个带资源标注类型的函数式语言
- 程序合成的部分是归约为类型制导的程序合成问题

工作 3：资源制导的程序合成

T. Knoth, **D. Wang**, N. Polikarpova, and J. Hoffmann. Resource-Guided Program Synthesis. In *PLDI'19*.

- 虽然是程序合成的工作，但其核心是一个带资源标注类型的函数式语言
- 程序合成的部分是归约为类型制导的程序合成问题
- 相关工作：
  - P. Wang, **D. Wang**, and A. Chlipala. TiML: A Functional Language for Practical Complexity Analysis with Invariants. In *OOPSLA'17*.
  - T. Knoth, **D. Wang**, A. Reynolds, J. Hoffmann, and N. Polikarpova. Liquid Resource Types. In *ICFP'20*.

以 Refinement Types 为规约

以 Refinement Types 为规约

$\{ v : B \mid \Psi \}$

一个满足 Ψ 的具有类型 B 的值 v

以 Refinement Types 为规约

`{ v: B | Ψ }` 一个满足 Ψ 的具有类型 B 的值 v

`{ v: Int | v ≥ 0 }` 一个非负整数

以 Refinement Types 为规约

`{ v: B | Ψ }` 一个满足 Ψ 的具有类型 B 的值 v

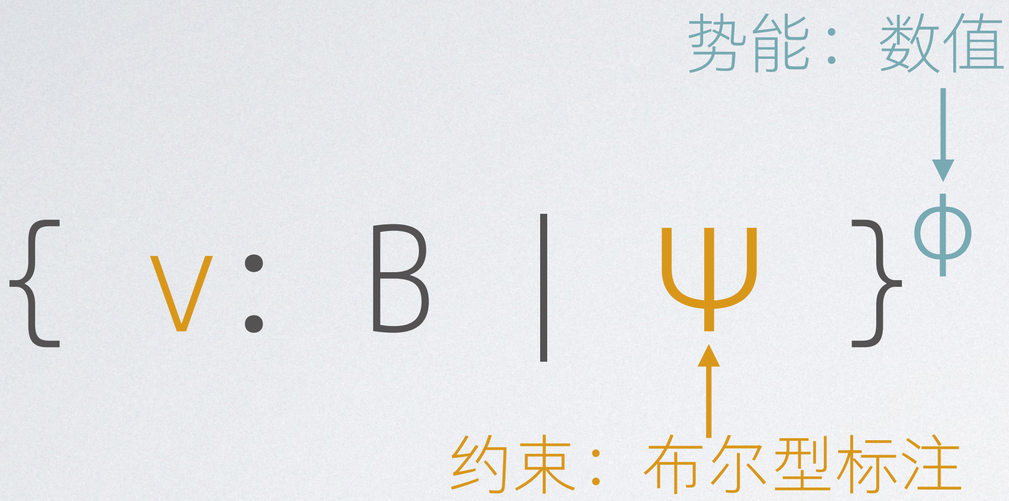
`{ v: Int | v ≥ 0 }` 一个非负整数

```
(xs: List a) -> { v: List a | len(v) = len(xs) + 1 }
```

一个函数，输入为一个列表 xs，输出为一个长度恰好比 xs 多 1 的列表

ReSyn： Refinement Types + 线性势能函数

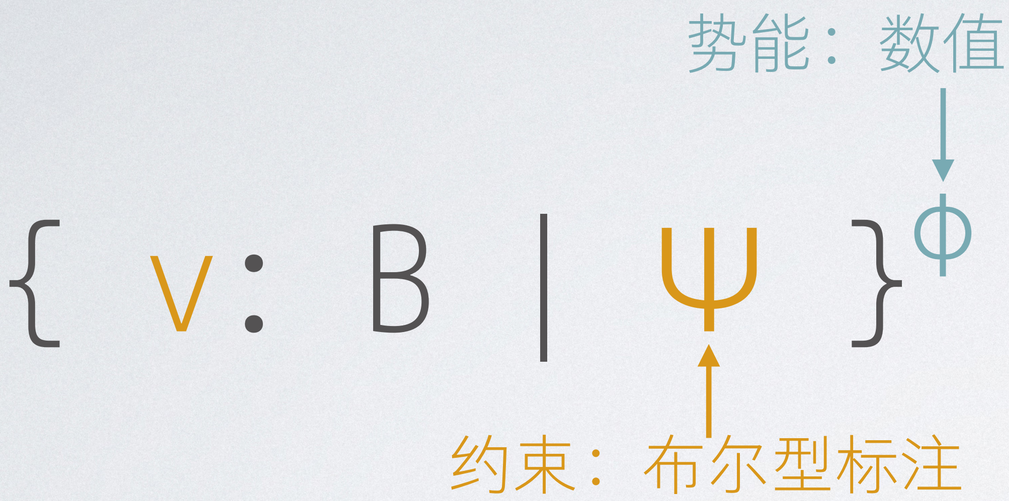
ReSyn： Refinement Types + 线性势能函数



$\{\ v{:}\ \mathrm{Int} \mid v \geq 0\ \}^{5 \cdot v}$

一个非负整数，携带了5倍于其自身值的势能

ReSyn： Refinement Types + 线性势能函数



$\{ v: Int \mid v \geq 0 \}^{5 \cdot v}$

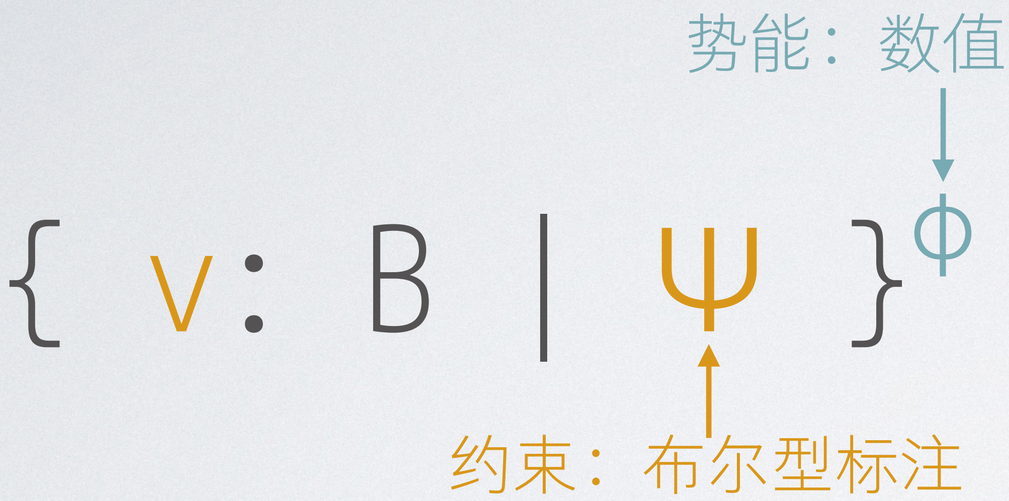
一个非负整数，携带了5倍于其自身值的势能

$List\ \{ v: a \mid true \}^{ite(v \geq 0, 1, 0)}$

一个列表，其中的每个非负元素
携带了1单位的势能

ite = if-then-else

ReSyn： Refinement Types + 线性势能函数



理论贡献：设计了将两种标注整合的类型系统，并证明了该类型系统的安全性，和基于该类型系统的程序合成算法的正确性

$\{ v: Int \mid v \geq 0 \}^{5 \cdot v}$

一个非负整数，携带了5倍于其自身值的势能

$List\ \{ v: a \mid true \}^{ite(v \geq 0, 1, 0)}$

一个列表，其中的每个非负元素
携带了1单位的势能

ite = if-then-else

资源制导的程序合成

```
common : (xs: SortedList a¹) -> (ys: SortedList a¹) ->
{ v: SortedList a | elems(v) = elems(xs) ∩ elems(ys) }
```

资源制导的程序合成

```
common : (xs: SortedList a¹) -> (ys: SortedList a¹) ->
{ v: SortedList a | elems(v) = elems(xs) ∩ elems(ys) }
```

资源制导的程序合成

```
common : (xs: SortedList a[1]) -> (ys: SortedList a[1]) ->
{ v: SortedList a | elems(v) = elems(xs) ∩ elems(ys) }
```

```
let rec common xs ys =
  match xs with
  | [] -> []
  | x::xt ->
    if not (member x ys)
    then common xt ys
    else x::(common xt ys)
```

平方时间复杂度

资源制导的程序合成

```
common : (xs: SortedList a^1) -> (ys: SortedList a^1) ->
{ v: SortedList a | elems(v) = elems(xs) ∩ elems(ys) }
```

```
let rec common xs ys =
  match xs with
  | [] -> []
  | x::xt ->
    if not (member x ys)
    then common xt ys
    else x::(common xt ys)
```

平方时间复杂度

```
let rec common xs ys =
  match xs with
  | [] -> []
  | x::xt ->
    match ys with
    | [] -> []
    | y::yt ->
      if x < y then common xt ys
      else if y < x then common xs yt
      else x::(common xt yt)
```

ReSyn：线性时间复杂度

编程语言

意识中的计算

WA 24和36的最大公约数是多少?

24和36的最大公约数是12。

WA 请给出计算过程。

计算两个数的最大公约数通常使用欧几里德算法，以下是计算24和36的最大公约数的过程：

1. 用较大的数除以较小的数，即36 ÷ 24 = 1 余 12。
2. 接下来，将较小的数（24）除以余数（12），即24 ÷ 12 = 2 余 0。

当余数为0时，计算结束，最大公约数是上一步的余数，即12。所以，24和36的最大公约数是12。

让程序更加安全

让程序更容易写

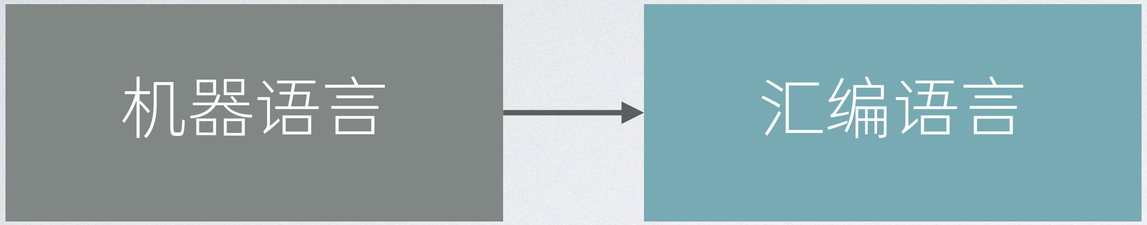
硬件上的计算

趋势：语言的抽象层次越来越高

趋势：语言的抽象层次越来越高

机器语言

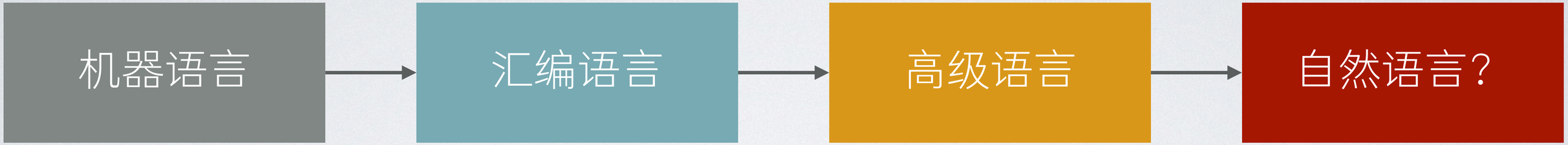
趋势：语言的抽象层次越来越高



趋势：语言的抽象层次越来越高

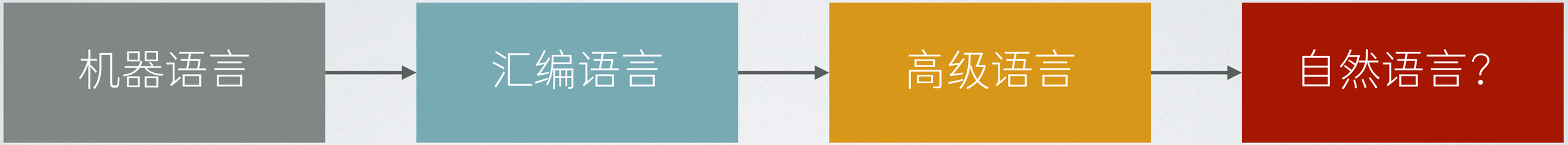
命令式语言
函数式语言
……

趋势：语言的抽象层次越来越高



命令式语言
函数式语言
……

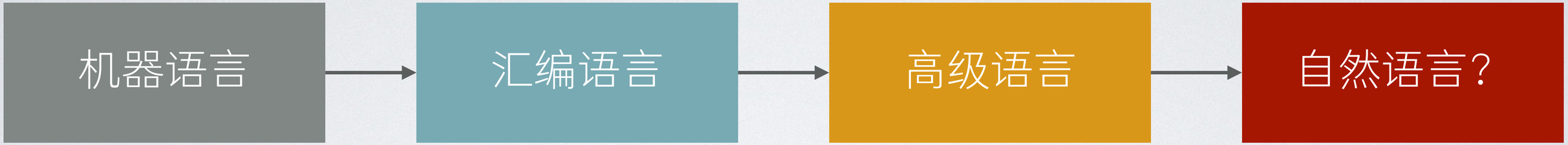
趋势：语言的抽象层次越来越高



命令式语言
函数式语言
……

E. W. Dijkstra. 1978. On the Foolishness of “Natural Language Programming.”

趋势：语言的抽象层次越来越高

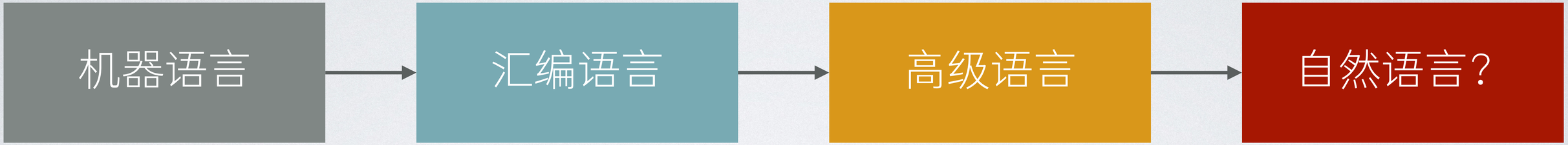


命令式语言
函数式语言
……

E. W. Dijkstra. 1978. On the Foolishness of “Natural Language Programming.”

在 AI 的浪潮下，自然语言编程是否能带来新型的抽象？

趋势：语言的抽象层次越来越高



命令式语言
函数式语言
……

E. W. Dijkstra. 1978. On the Foolishness of “Natural Language Programming.”

在 AI 的浪潮下，自然语言编程是否能带来新型的抽象？
或者，是否有更适合的对 AI 进行编程的范式？

SCENIC：驾驶场景描述语言

```
spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) *
           Range(10, 20) deg
Car left of spot by 0.5,
facing badAngle relative to roadDirection
```

D. J. Fremont el al. 2019. SCENIC: A Language for Scenario Specification and Scene Generation. In *PLDI'19*.

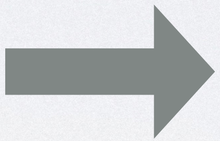
SCENIC：驾驶场景描述语言

```
spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) *
           Range(10, 20) deg
Car left of spot by 0.5,
facing badAngle relative to roadDirection
```

D. J. Fremont el al. 2019. SCENIC: A Language for Scenario Specification and Scene Generation. In *PLDI'19*.

SCENIC：驾驶场景描述语言

```
spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) *
           Range(10, 20) deg
Car left of spot by 0.5,
facing badAngle relative to roadDirection
```



生成式概率编程：输出为符合要求的随机场景

D. J. Fremont el al. 2019. SCENIC: A Language for Scenario Specification and Scene Generation. In *PLDI'19*.

概率编程：通过程序描述概率模型

概率编程：通过程序描述概率模型

```
ra = resource analysis
pp = probabilistic programming
```

```
prof_like_my_work_on_ra ~ Bernoulli(0.4)
prof_like_my_work_on_pp ~ Bernoulli(0.6)
```

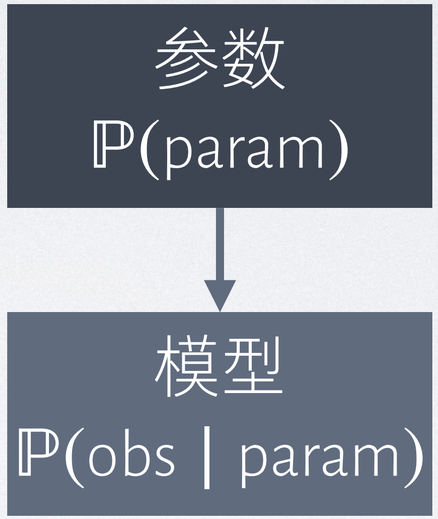
参数
$\mathbb{P}$(param)

概率编程：通过程序描述概率模型

```
ra = resource analysis
pp = probabilistic programming
```

```
prof_like_my_work_on_ra ~ Bernoulli(0.4)
prof_like_my_work_on_pp ~ Bernoulli(0.6)

if (prof like both):
  receive_invitation ~ Bernoulli(0.8)
else if (prof only like pp):
  receive_invitation ~ Bernoulli(0.5)
else:
  …
```



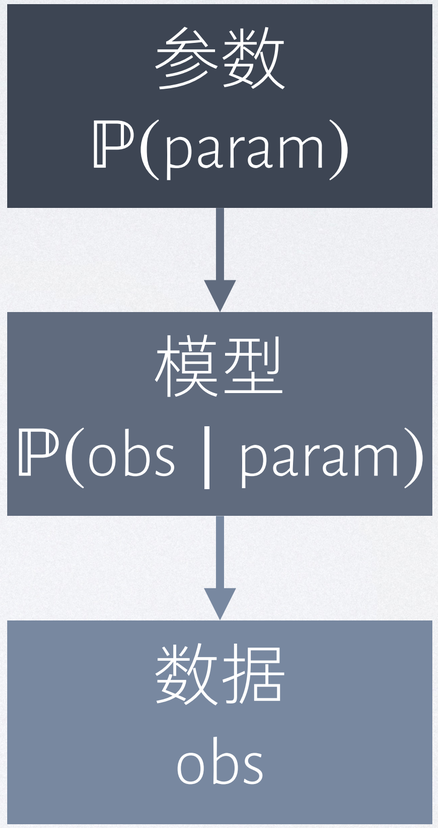
概率编程：通过程序描述概率模型

```
ra = resource analysis
pp = probabilistic programming
```

```
prof_like_my_work_on_ra ~ Bernoulli(0.4)
prof_like_my_work_on_pp ~ Bernoulli(0.6)

if (prof like both):
  receive_invitation ~ Bernoulli(0.8)
else if (prof only like pp):
  receive_invitation ~ Bernoulli(0.5)
else:
  …

observe (receive_invitation == true)
```



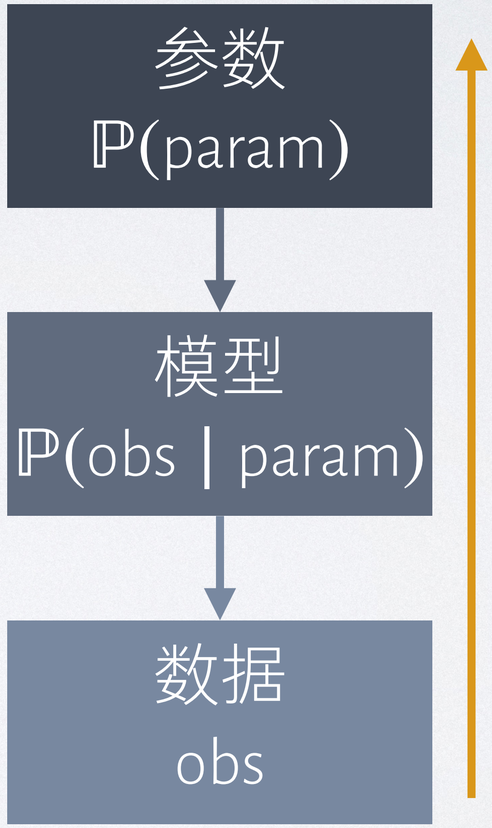
概率编程：通过程序描述概率模型

```
ra = resource analysis
pp = probabilistic programming
```

```
prof_like_my_work_on_ra ~ Bernoulli(0.4)
prof_like_my_work_on_pp ~ Bernoulli(0.6)

if (prof like both):
  receive_invitation ~ Bernoulli(0.8)
else if (prof only like pp):
  receive_invitation ~ Bernoulli(0.5)
else:
  …

observe (receive_invitation == true)
```



贝叶斯推断
ℙ(param | obs)

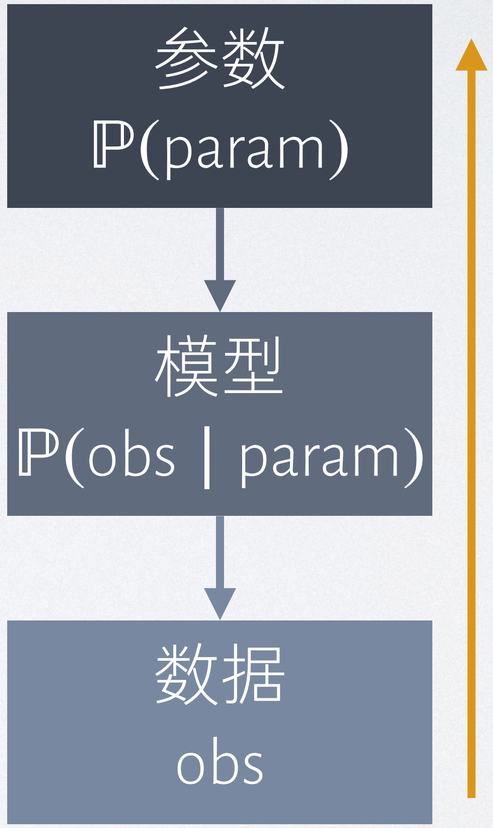
概率编程：通过程序描述概率模型

```
ra = resource analysis
pp = probabilistic programming
```

```
prof_like_my_work_on_ra ~ Bernoulli(0.4)
prof_like_my_work_on_pp ~ Bernoulli(0.6)

if (prof like both):
  receive_invitation ~ Bernoulli(0.8)
else if (prof only like pp):
  receive_invitation ~ Bernoulli(0.5)
else:
  …

observe (receive_invitation == true)
```



面试官喜欢我在概率编程上的工作的后验概率是多少？

贝叶斯推断
ℙ(param | obs)

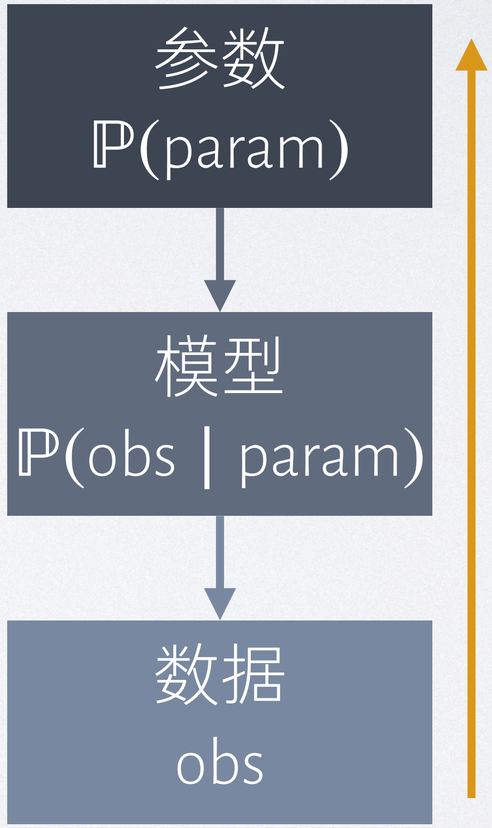
概率编程：通过程序描述概率模型

```
ra = resource analysis
pp = probabilistic programming
```

```
prof_like_my_work_on_ra ~ Bernoulli(0.4)
prof_like_my_work_on_pp ~ Bernoulli(0.6)

if (prof like both):
  receive_invitation ~ Bernoulli(0.8)
else if (prof only like pp):
  receive_invitation ~ Bernoulli(0.5)
else:
  …

observe (receive_invitation == true)
```

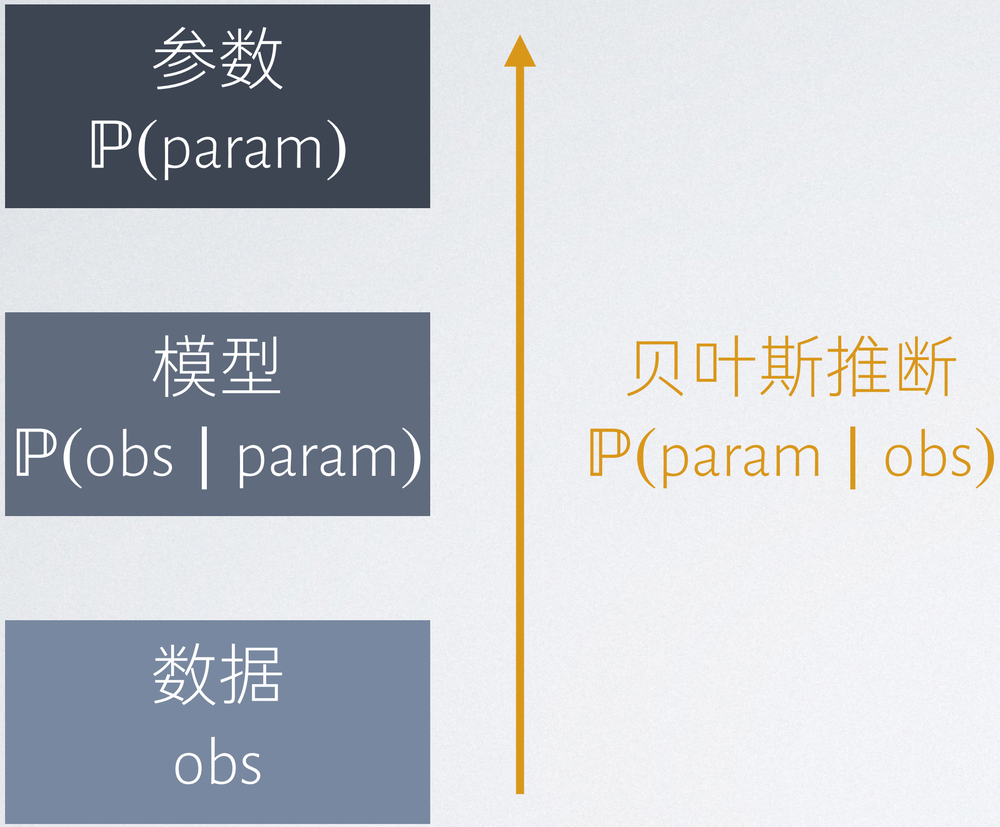


面试官喜欢我在概率编程上的工作的后验概率是多少？

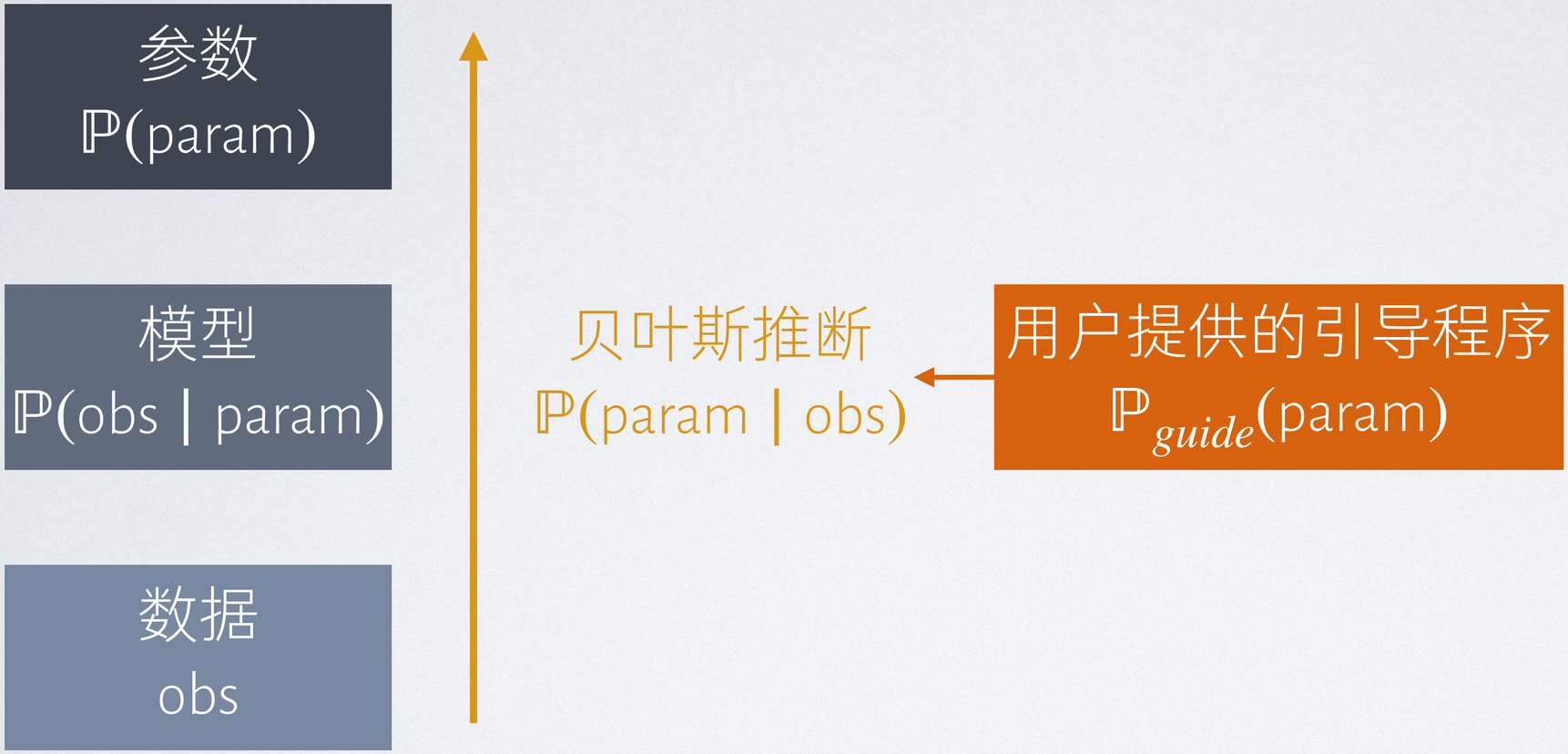
贝叶斯推断
ℙ(param | obs)

一个困难的问题

可编程概率推断



可编程概率推断


参数
ℙ(param)
模型
ℙ(obs | param)
数据
obs
贝叶斯推断
ℙ(param | obs)
用户提供的引导程序
$\mathbb{P}_{guide}$(param)

可编程概率推断

参数
$\mathbb{P}$(param)

模型
$\mathbb{P}$(obs | param)

数据
obs

贝叶斯推断
$\mathbb{P}$(param | obs)

用户提供的引导程序
$\mathbb{P}_{guide}$(param)

优势：提供了对具体模型针对性调优的灵活性

可编程概率推断

参数
$\mathbb{P}$(param)

模型
$\mathbb{P}$(obs | param)

数据
obs

贝叶斯推断
$\mathbb{P}$(param | obs)

用户提供的引导程序
$\mathbb{P}_{guide}$(param)

优势：提供了对具体模型针对性调优的灵活性

劣势：用户需要确保模型-引导的一致性

可编程概率推断

参数
$\mathbb{P}$(param)

模型
$\mathbb{P}$(obs | param)

数据
obs

贝叶斯推断
$\mathbb{P}$(param | obs)

用户提供的引导程序
$\mathbb{P}_{guide}$(param)

优势：提供了对具体模型针对性调优的灵活性

劣势：用户需要确保模型-引导的一致性

一个复杂的任务

Pyro：

4. Make sure your model and guide distributions have the same support

工作 4：基于协程的模型-引导编程

D. Wang, J. Hoffmann, and T. Reps. Sound Probabilistic Inference via Guide Types. In *PLDI'21*.
L. Pham, **D. Wang**, F. Saad , and J. Hoffmann. Probabilistic Inference with Soundly Composed Guide Programs. *Working Paper*.

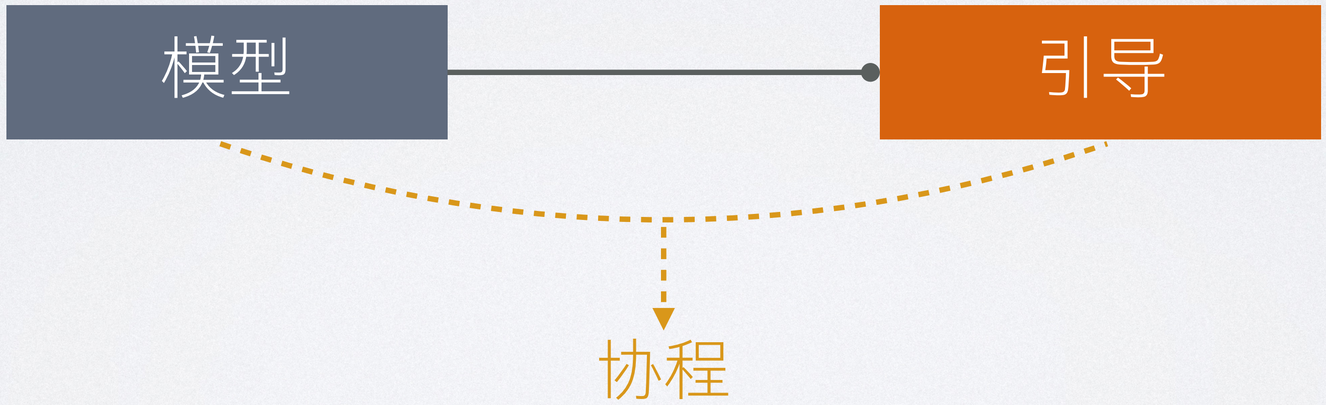
模型

引导

核心贡献：首个使用并发编程技术来确保可编程贝叶斯推断中的模型-引导一致性

工作 4：基于协程的模型-引导编程

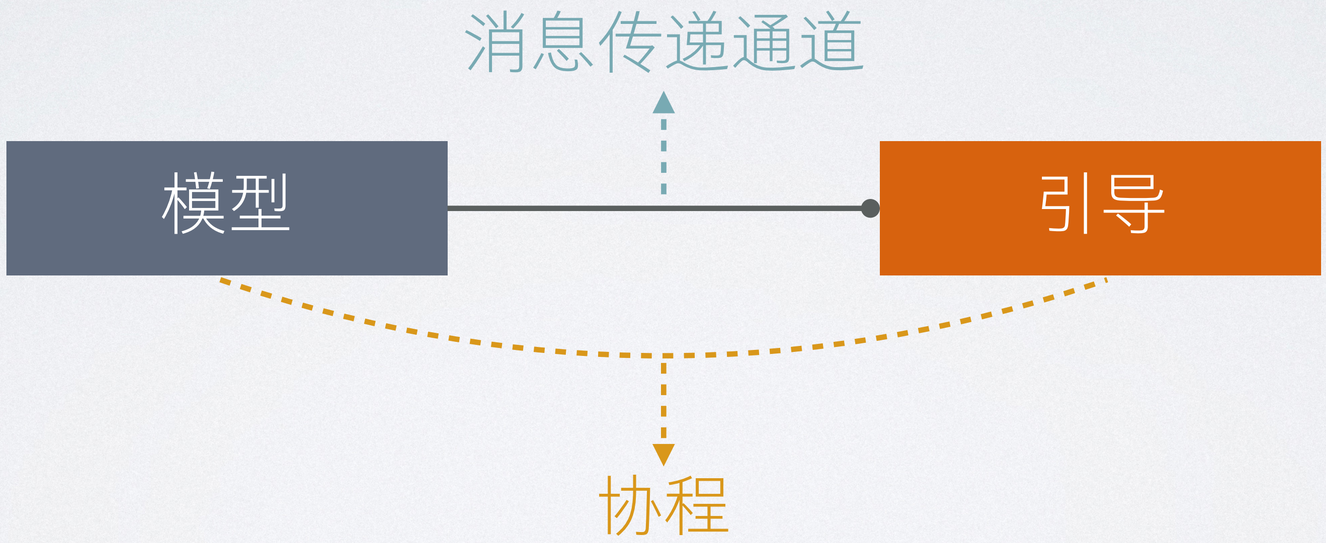
D. Wang, J. Hoffmann, and T. Reps. Sound Probabilistic Inference via Guide Types. In *PLDI'21*.
L. Pham, **D. Wang**, F. Saad , and J. Hoffmann. Probabilistic Inference with Soundly Composed Guide Programs. *Working Paper*.



核心贡献：首个使用并发编程技术来确保可编程贝叶斯推断中的模型-引导一致性

工作 4：基于协程的模型-引导编程

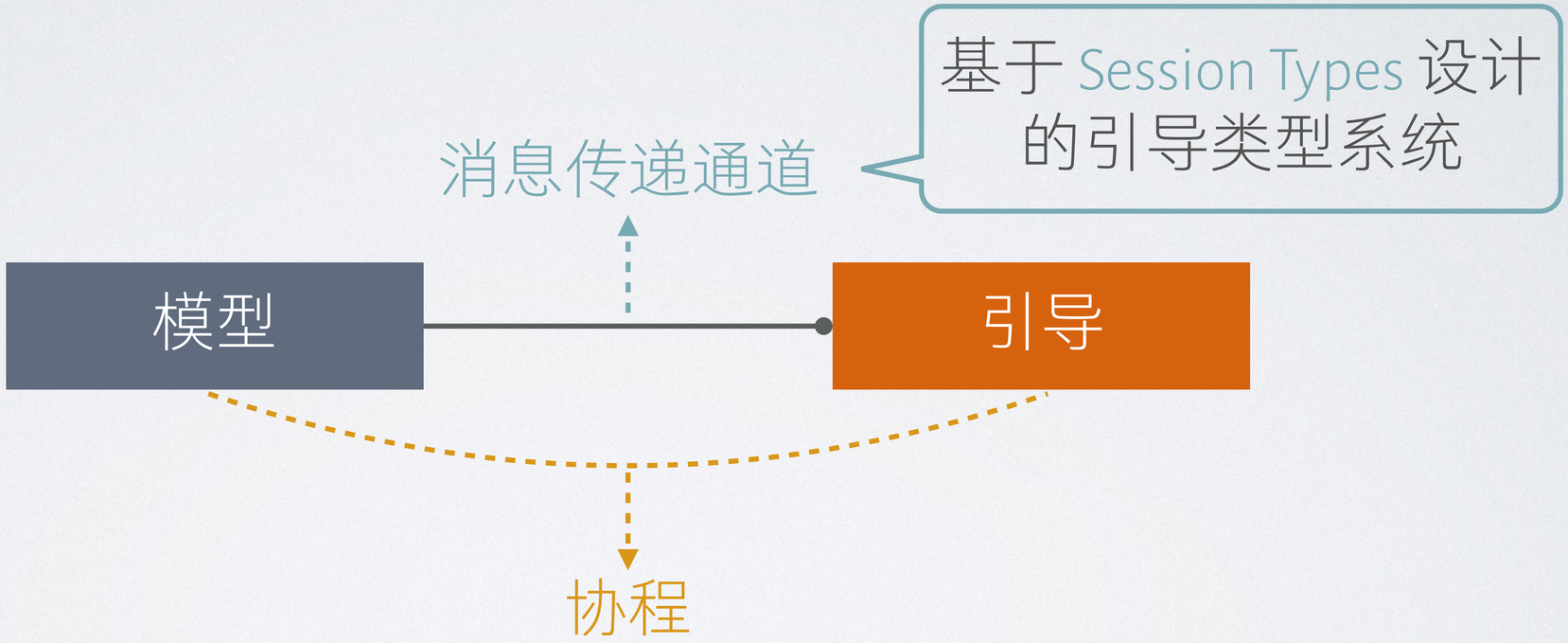
D. Wang, J. Hoffmann, and T. Reps. Sound Probabilistic Inference via Guide Types. In *PLDI'21*.
L. Pham, **D. Wang**, F. Saad , and J. Hoffmann. Probabilistic Inference with Soundly Composed Guide Programs. *Working Paper*.



核心贡献：首个使用并发编程技术来确保可编程贝叶斯推断中的模型-引导一致性

工作 4：基于协程的模型-引导编程

D. Wang, J. Hoffmann, and T. Reps. Sound Probabilistic Inference via Guide Types. In *PLDI'21*.
L. Pham, **D. Wang**, F. Saad , and J. Hoffmann. Probabilistic Inference with Soundly Composed Guide Programs. *Working Paper*.



核心贡献：首个使用并发编程技术来确保可编程贝叶斯推断中的模型-引导一致性

让 AI 来做引导?

- 回顾“驾驶场景生成”问题：典型的 AI 生成内容（AIGC）的应用

让 AI 来做引导?

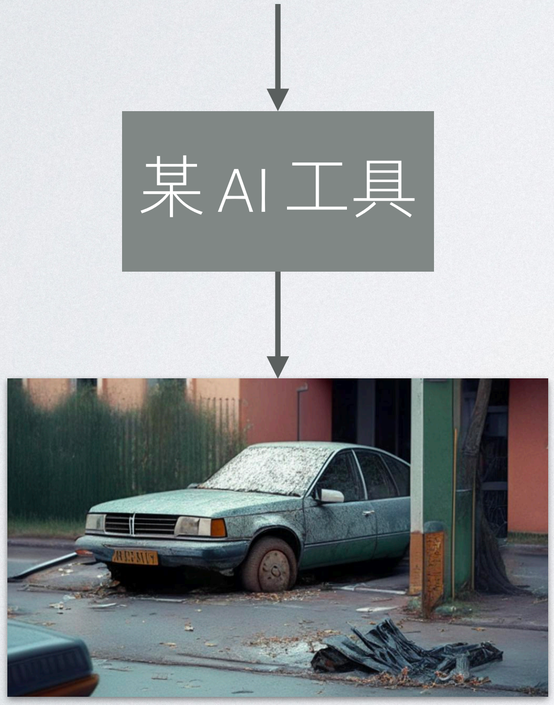
- 回顾“驾驶场景生成”问题：典型的 AI 生成内容（AIGC）的应用

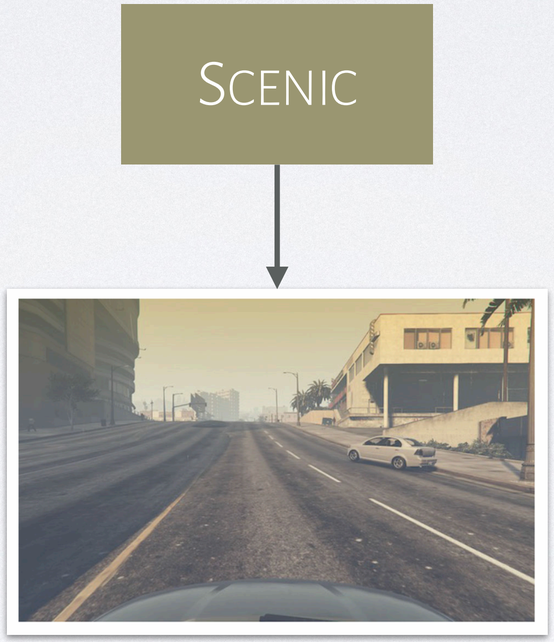
“A scene of a badly-parked car”

让 AI 来做引导?

- 回顾“驾驶场景生成”问题：典型的 AI 生成内容（AIGC）的应用

“A scene of a badly-parked car”

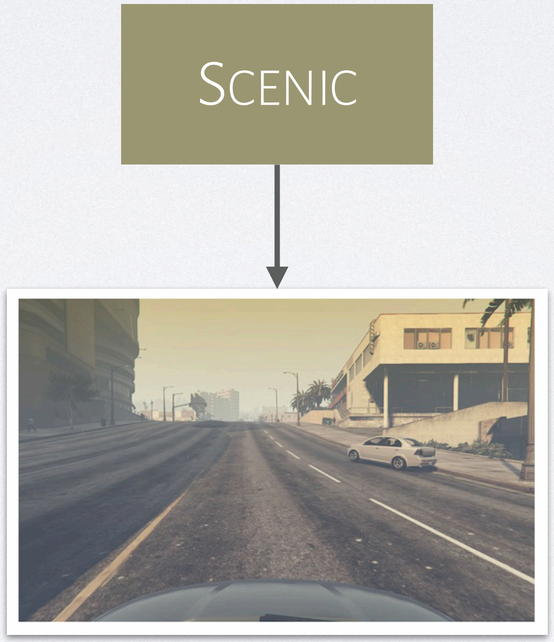




让 AI 来做引导?

- 回顾“驾驶场景生成”问题：典型的 AI 生成内容（AIGC）的应用

“A scene of a badly-parked car”

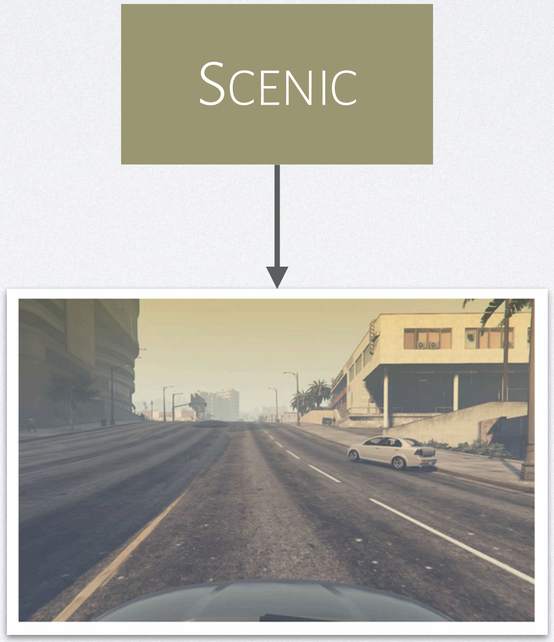


- AIGC 的优势：内容丰富、擅长处理模糊性

让 AI 来做引导?

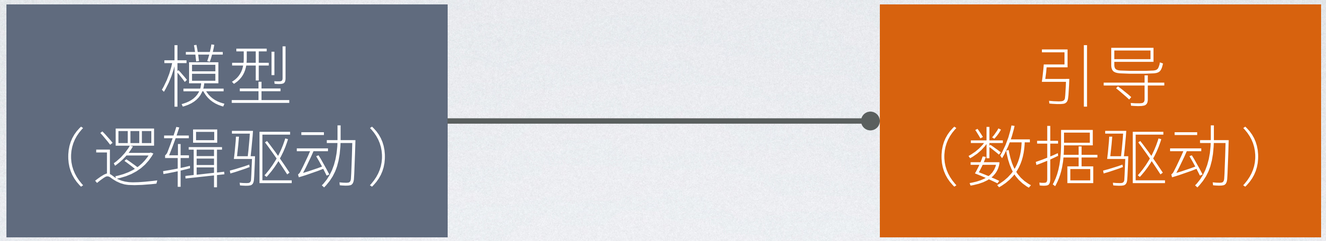
- 回顾“驾驶场景生成”问题：典型的 AI 生成内容（AIGC）的应用

“A scene of a badly-parked car”



- AIGC 的优势：内容丰富、擅长处理模糊性
- 概率编程的优势：可解释性高、擅长表达逻辑

生成式模糊编程语言



生成式模糊编程语言

生成式模糊编程语言

模型
（逻辑驱动）

引导
（数据驱动）

智能系统
（如 GPT）提
供采样能力

```
spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) *
           Range(10, 20) deg
Car left of spot by 0.5,
facing badAngle relative to roadDirection
```

+

“A scene of a badly-parked car”

生成式模糊编程语言

模型
（逻辑驱动）

引导
（数据驱动）

智能系统
（如 GPT）提
供采样能力

```
spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) *
           Range(10, 20) deg
Car left of spot by 0.5,
facing badAngle relative to roadDirection
```

+

“A scene of a badly-parked car”

- 需要逻辑的生成场景：

生成式模糊编程语言

模型
（逻辑驱动）

引导
（数据驱动）

智能系统
（如 GPT）提
供采样能力

```
spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) *
           Range(10, 20) deg
Car left of spot by 0.5,
facing badAngle relative to roadDirection
```

+

“A scene of a badly-parked car”

- 需要逻辑的生成场景：
  - 生成建筑设计图，要求满足安全规定

生成式模糊编程语言

模型
（逻辑驱动）

引导
（数据驱动）

智能系统
（如 GPT）提
供采样能力

```
spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) *
           Range(10, 20) deg
Car left of spot by 0.5,
facing badAngle relative to roadDirection
```

+

“A scene of a badly-parked car”

- 需要逻辑的生成场景：
  - 生成建筑设计图，要求满足安全规定
  - 生成法律文本，要求符合相关规范

生成式模糊编程语言

模型
（逻辑驱动）

引导
（数据驱动）

智能系统（如 GPT）提供采样能力

```
spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) *
           Range(10, 20) deg
Car left of spot by 0.5,
facing badAngle relative to roadDirection
```

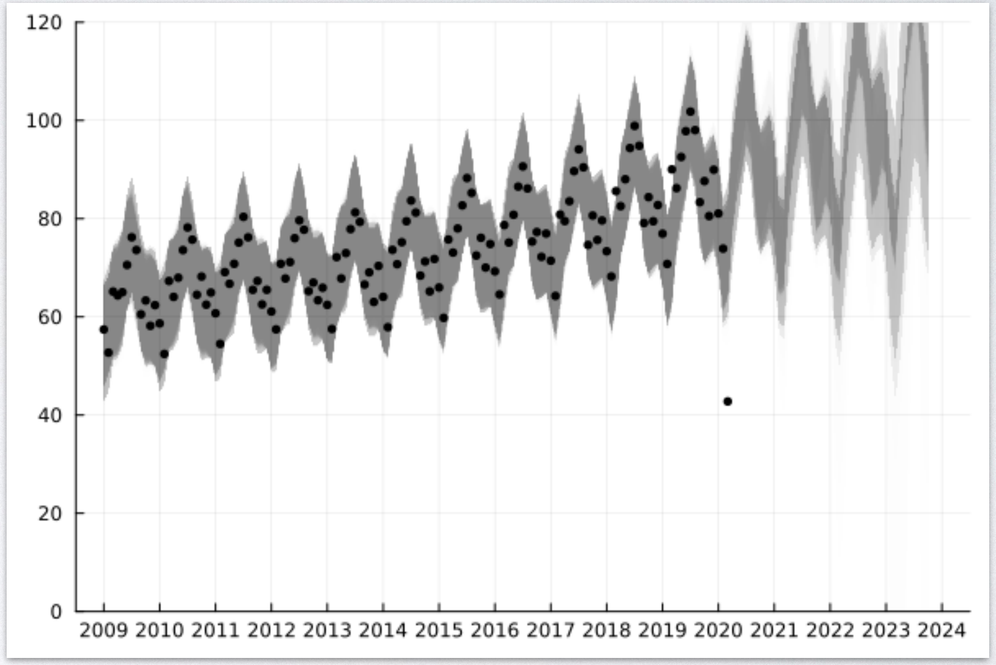
+

“A scene of a badly-parked car”

- 需要逻辑的生成场景：
  - 生成建筑设计图，要求满足安全规定
  - 生成法律文本，要求符合相关规范
  - 生成考试题目，要求不能超纲

AI 的能力：采样 + 反馈

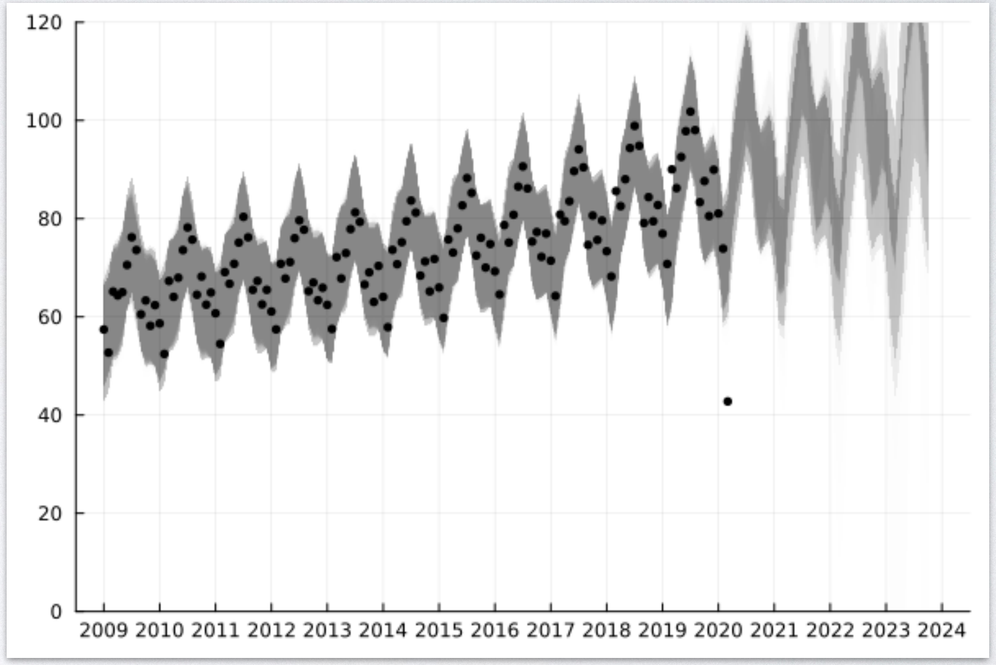
- 除了采样能力，智能系统往往可以通过给予反馈进行自适应的演化



美国国内民航总里程（按月）

AI 的能力：采样 + 反馈

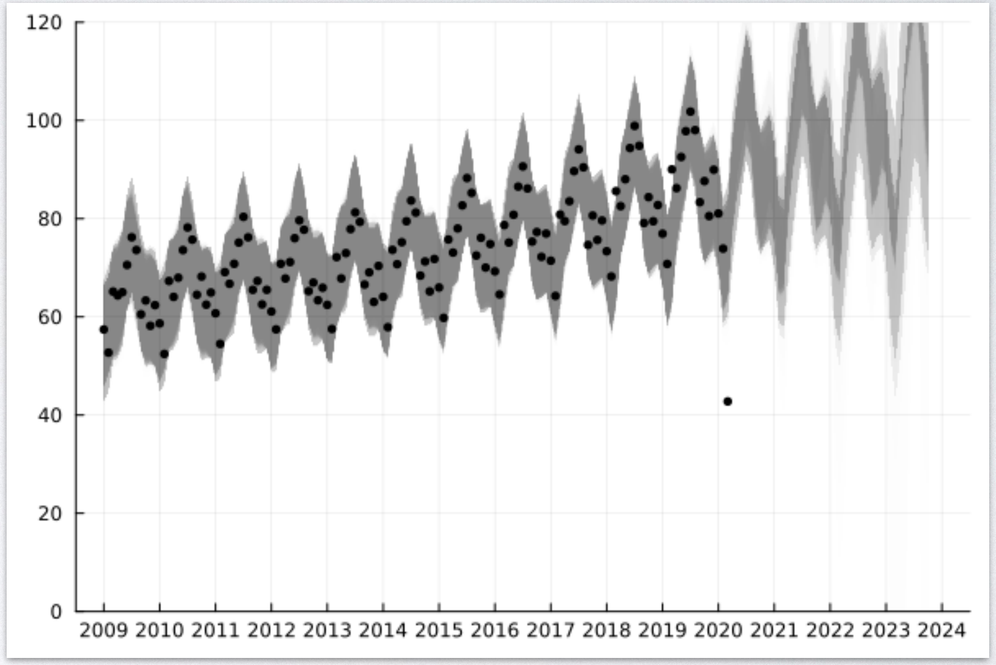
- 除了采样能力，智能系统往往可以通过给予反馈进行自适应的演化



美国国内民航总里程（按月）

AI 的能力：采样 + 反馈

- 除了采样能力，智能系统往往可以通过给予反馈进行自适应的演化



美国国内民航总里程（按月）

编程语言新范式初探

让程序更加安全：资源安全系统编程语言

让程序更容易写：AI 驱动的模糊编程语言