



编程语言新范式初探

王迪

北京大学

wangdi95@pku.edu.cn

2024年06月22日

下一代编程语言

仓颉开启预览

原生智能化

天生全场景

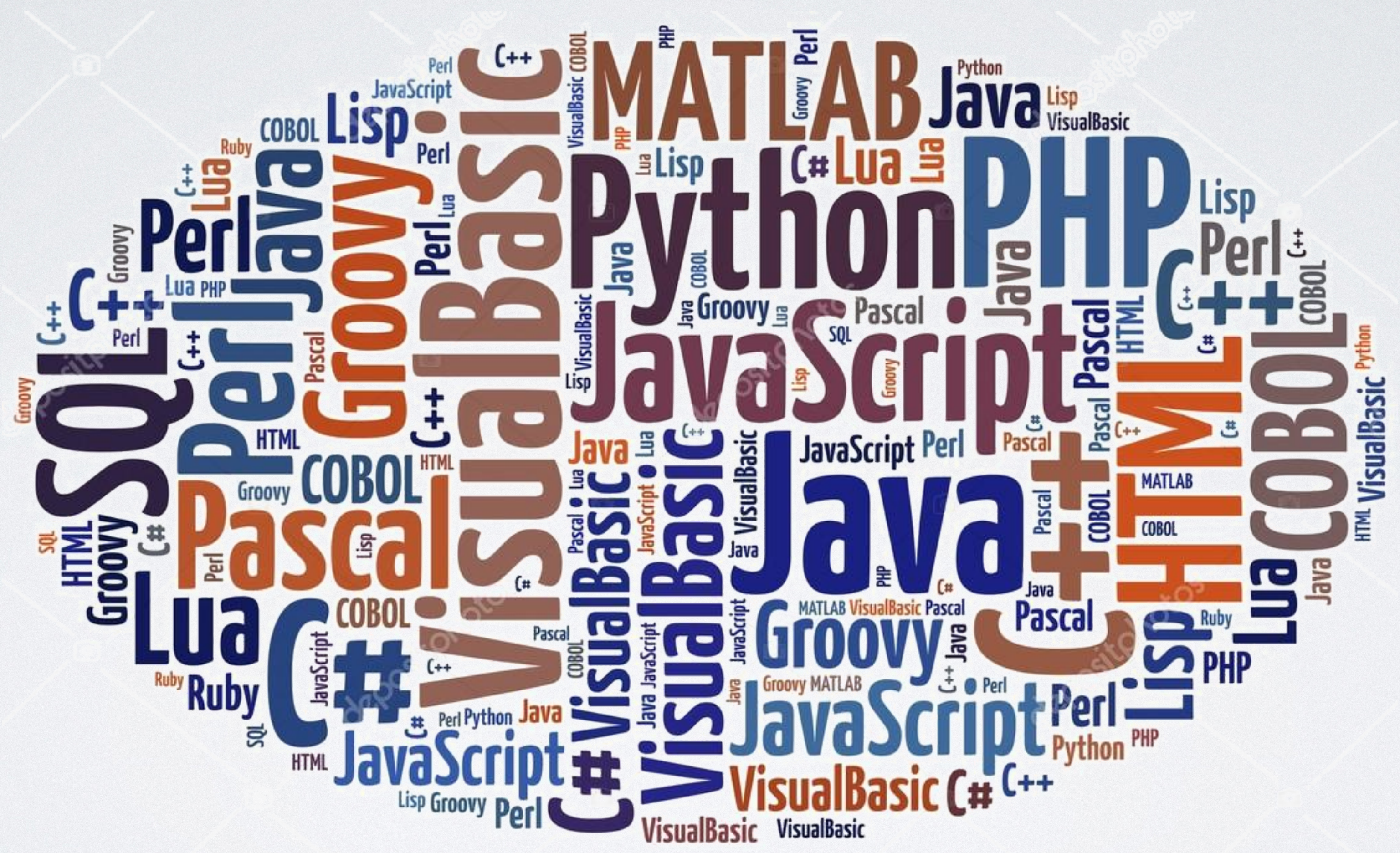
高性能

强安全

developer.huawei.com/consumer/cn/cangjie/



为啥有这么多编程语言？





编程语言与计算



编程语言与计算

如何计算两个数的最大公约数?

计算两个数的最大公约数，可以使用欧几里得算法，也称为辗转相除法。这个算法基于一个简单的原理： $\text{gcd}(a, b) = \text{gcd}(b, a \bmod b)$ ，其中 \bmod 表示模运算。

以下是计算两个数的最大公约数的步骤：

1. 如果其中一个数为0，则另一个数就是它们的最大公约数。
2. 否则，用较大的数减去较小的数，得到一个新的差。
3. 重复步骤2，直到差为0为止。
4. 当差为0时，两个数中较大的数就是它们的最大公约数。

意识中的计算

编程语言与计算

如何计算两个数的最大公约数?

计算两个数的最大公约数，可以使用欧几里得算法，也称为辗转相除法。这个算法基于一个简单的原理： $\text{gcd}(a, b) = \text{gcd}(b, a \bmod b)$ ，其中 \bmod 表示模运算。

以下是计算两个数的最大公约数的步骤：

1. 如果其中一个数为0，则另一个数就是它们的最大公约数。
2. 否则，用较大的数减去较小的数，得到一个新的差。
3. 重复步骤2，直到差为0为止。
4. 当差为0时，两个数中较大的数就是它们的最大公约数。

意识中的计算



硬件上的计算

编程语言与计算

如何计算两个数的最大公约数?

计算两个数的最大公约数，可以使用欧几里得算法，也称为辗转相除法。这个算法基于一个简单的原理： $\text{gcd}(a, b) = \text{gcd}(b, a \bmod b)$ ，其中 \bmod 表示模运算。

以下是计算两个数的最大公约数的步骤：

1. 如果其中一个数为0，则另一个数就是它们的最大公约数。
2. 否则，用较大的数减去较小的数，得到一个新的差。
3. 重复步骤2，直到差为0为止。
4. 当差为0时，两个数中较大的数就是它们的最大公约数。



意识中的计算

硬件上的计算

编程语言与计算

如何计算两个数的最大公约数?

计算两个数的最大公约数，可以使用欧几里得算法，也称为辗转相除法。这个算法基于一个简单的原理： $\text{gcd}(a, b) = \text{gcd}(b, a \bmod b)$ ，其中 \bmod 表示模运算。

以下是计算两个数的最大公约数的步骤：

1. 如果其中一个数为0，则另一个数就是它们的最大公约数。
2. 否则，用较大的数减去较小的数，得到一个新的差。
3. 重复步骤2，直到差为0为止。
4. 当差为0时，两个数中较大的数就是它们的最大公约数。

```
int gcd(int a, int b) {  
    while (b != 0) {  
        int t = b;  
        b = a % b;  
        a = t;  
    }  
    return a;  
}
```



意识中的计算

程序

硬件上的计算

编程语言与计算

如何计算两个数的最大公约数?

计算两个数的最大公约数，可以使用欧几里得算法，也称为辗转相除法。这个算法基于一个简单的原理： $\text{gcd}(a, b) = \text{gcd}(b, a \bmod b)$ ，其中 \bmod 表示模运算。

以下是计算两个数的最大公约数的步骤：

1. 如果其中一个数为0，则另一个数就是它们的最大公约数。
2. 否则，用较大的数减去较小的数，得到一个新的差。
3. 重复步骤2，直到差为0为止。
4. 当差为0时，两个数中较大的数就是它们的最大公约数。

编程语言：C

```
int gcd(int a, int b) {  
    while (b != 0) {  
        int t = b;  
        b = a % b;  
        a = t;  
    }  
    return a;  
}
```



意识中的计算

程序

硬件上的计算



编程语言：驾驭计算的工具



编程语言：驾驭计算的工具

「计算两个数的最大公约数」



编程语言：驾驭计算的工具

「计算两个数的最大公约数」

「解一个数独」



编程语言：驾驭计算的工具

计算
任务

「计算两个数的最大公约数」

「解一个数独」



编程语言：驾驭计算的工具

计算
任务

「计算两个数的最大公约数」

「解一个数独」

「判断一张图片是不是人脸」



编程语言：驾驭计算的工具

计算
任务

「计算两个数的最大公约数」

「解一个数独」

「判断一张图片是不是人脸」

「给出一个棋盘状态下的最优决策」



编程语言：驾驭计算的工具

计算
任务

「计算两个数的最大公约数」

「解一个数独」

「判断一张图片是不是人脸」

「给出一个棋盘状态下的最优决策」

「写一首李白风格的关于火锅的现代诗」



编程语言：驾驭计算的工具

计算
任务

「计算两个数的最大公约数」

「解一个数独」

「判断一张图片是不是人脸」

「给出一个棋盘状态下的最优决策」

「写一首李白风格的关于火锅的现代诗」

「生成一张黑色柴犬在雪地的图片」

编程语言：驾驭计算的工具

计算
任务

「计算两个数的最大公约数」

「解一个数独」

「判断一张图片是不是人脸」

「给出一个棋盘状态下的最优决策」

「写一首李白风格的关于火锅的现代诗」

「生成一张黑色柴犬在雪地的图片」

计算
实现

CPU

GPU

TPU

FPGA

.....

编程语言：驾驭计算的工具

计算
任务

「计算两个数的最大公约数」

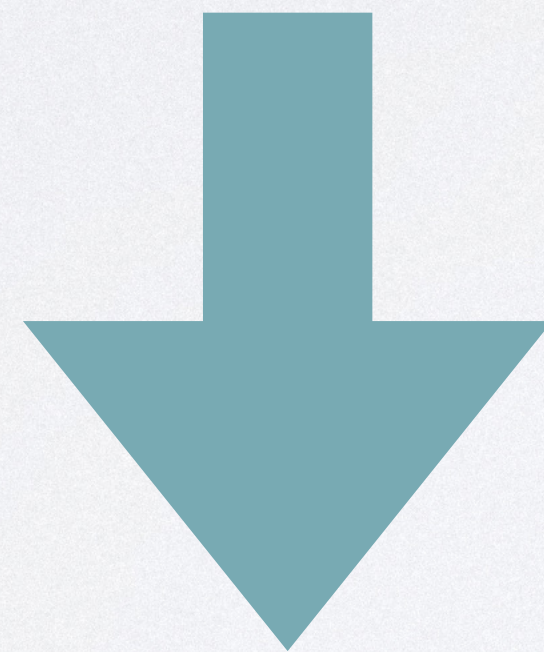
「解一个数独」

「判断一张图片是不是人脸」

「给出一个棋盘状态下的最优决策」

「写一首李白风格的关于火锅的现代诗」

「生成一张黑色柴犬在雪地的图片」



计算
实现

CPU

GPU

TPU

FPGA

.....

编程语言：驾驭计算的工具

计算
任务

「计算两个数的最大公约数」

「解一个数独」

「判断一张图片是不是人脸」

「给出一个棋盘状态下的最优决策」

「写一首李白风格的关于火锅的现代诗」

「生成一张黑色柴犬在雪地的图片」

用编程语言为计算任务写程序

计算
实现

CPU

GPU

TPU

FPGA

.....

编程语言：驾驭计算的工具

计算任务

「计算两个数的最大公约数」

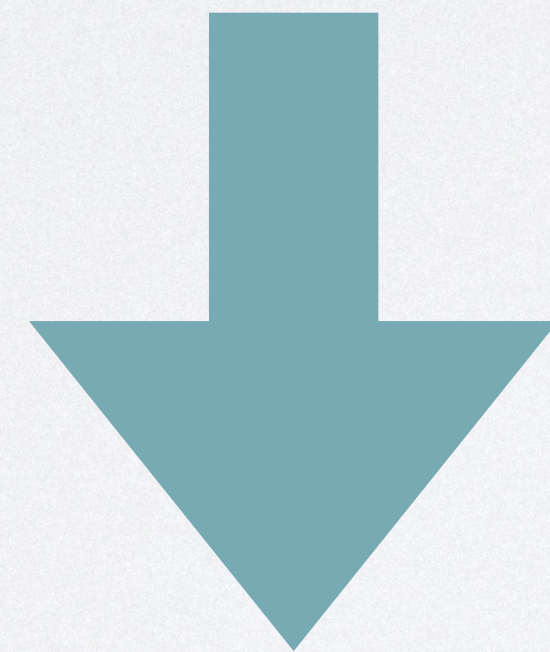
「解一个数独」

「判断一张图片是不是人脸」

「给出一个棋盘状态下的最优决策」

「写一首李白风格的关于火锅的现代诗」

「生成一张黑色柴犬在雪地的图片」



用编程语言为计算任务写程序
编译器自动把程序转换为计算实现

计算实现

CPU

GPU

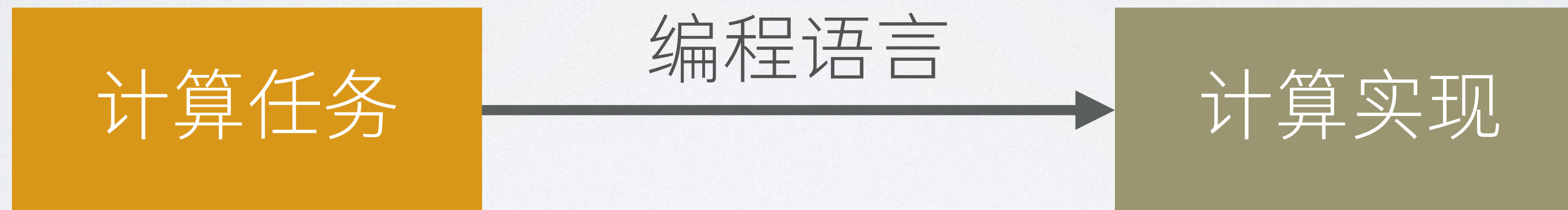
TPU

FPGA

.....

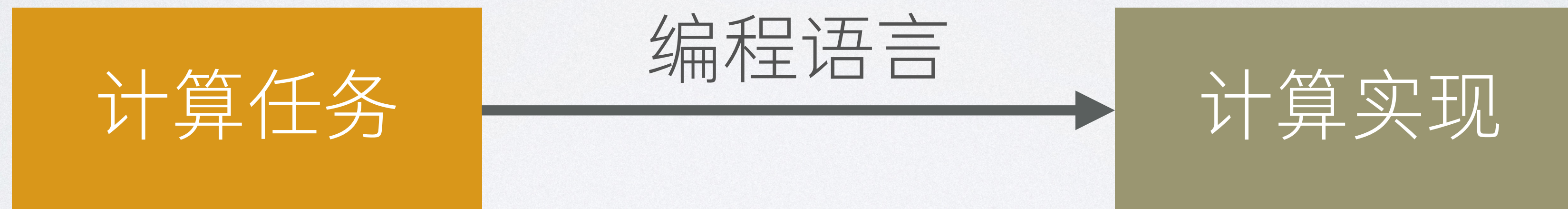


研究新范式的动机



研究新范式的动机

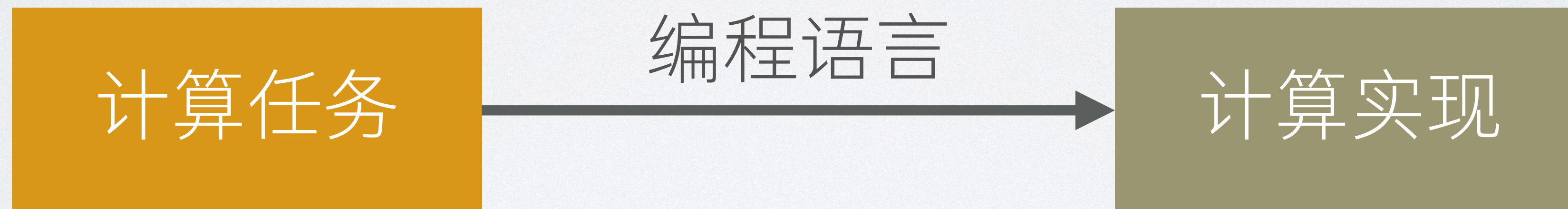
动机一：让程序更加安全
(保证正确高效地完成计算)



研究新范式的动机

动机一：让程序更加安全
(保证正确高效地完成计算)

动机二：驾驭智能的计算
(把智能系统视作抽象算力)





三个系统级编程语言

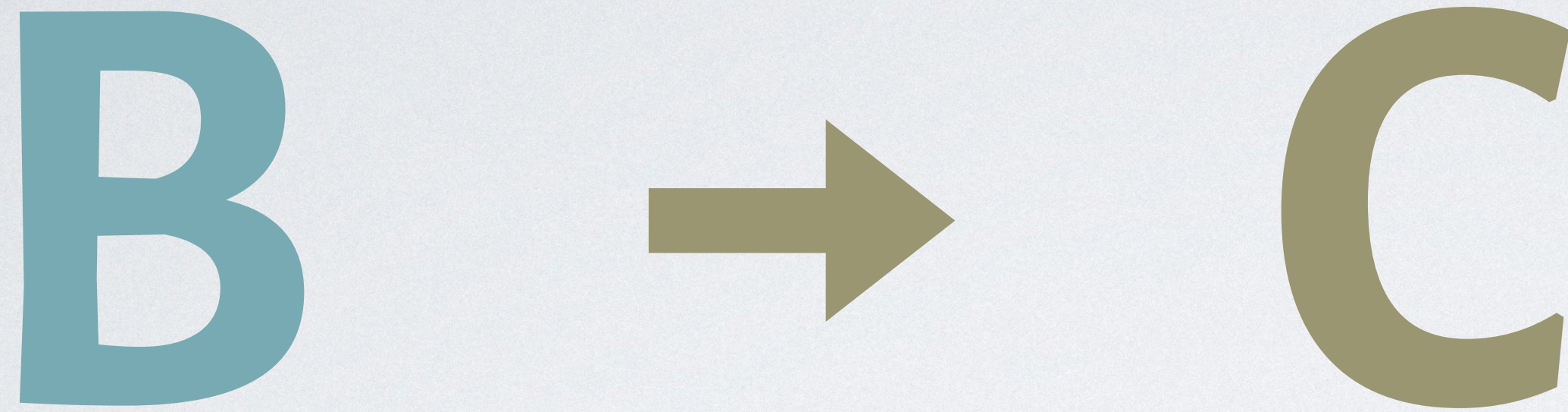


三个系统级编程语言

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!

三个系统级编程语言

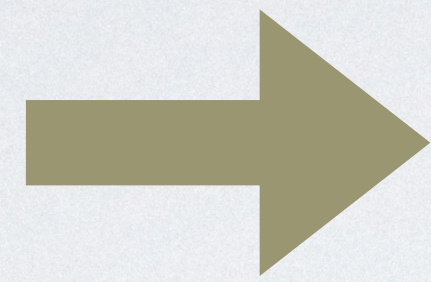


- ◎ 1969 年
- ◎ 贝尔实验室
- ◎ 为系统和编译器开发设计
- ◎ 无类型
- ◎ 非常慢!

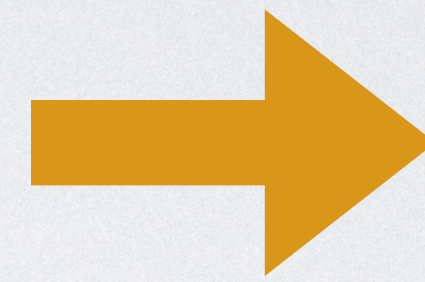
- ◎ 1972 年
- ◎ 贝尔实验室
- ◎ 通用编程语言
 - ◎ 长期被用来开发操作系统、设备驱动、协议栈等
- ◎ 静态类型系统
- ◎ 很快啊!

三个系统级编程语言

B



C



R_{ust}

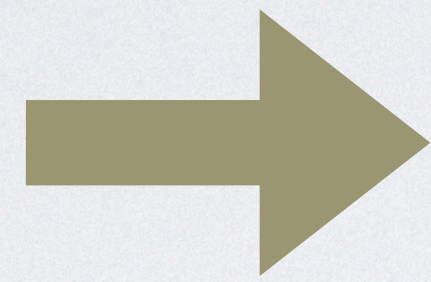
- 1969 年
- 贝尔实验室
- 为系统和编译器开发设计
- 无类型
- 非常慢!

- 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!

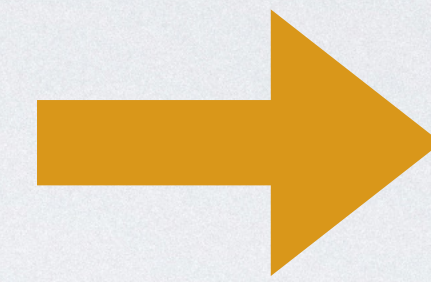
- 2015 年
- Mozilla
- 通用编程语言
 - 强调性能、安全以及并发
- 基于类型的内存安全
- 性能还不错!

三个系统级编程语言

B



C



R_{ust}

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

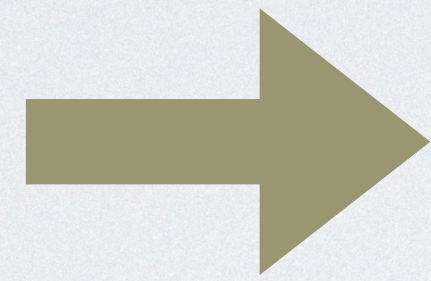
- 1972 年
- 贝尔实验室
- 通用编程语言
 - 长期被用来开发操作系统、设备驱动、协议栈等

- 静态类型系统
- 很快啊!

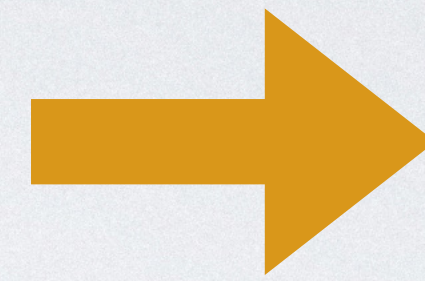
- 2015 年
- Mozilla
- 通用编程语言
 - 强调性能、安全以及并发
- 基于类型的内存安全
- 性能还不错!

三个系统级编程语言

B



C



R_{ust}

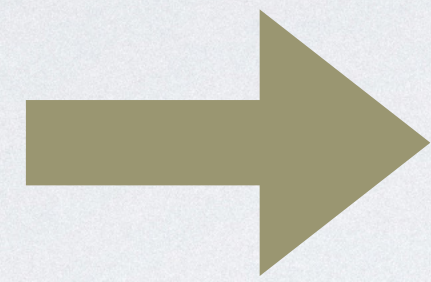
- 1969 年
- 贝尔实验室
- 为系统和编译器开发设计
- 无类型
- 非常慢!

- 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!

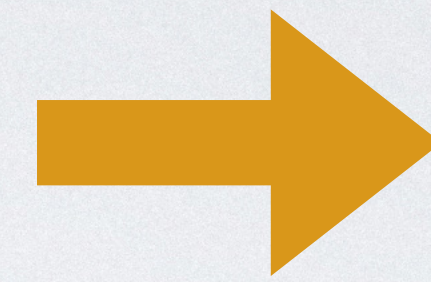
- 2015 年
- Mozilla
- 通用编程语言
 - 强调性能、安全以及并发
- 基于类型的内存安全
- 性能还不错!

趋势：语言的安全性越来越高

B



C



R_{ust}

无安全保障

非常慢

类型安全

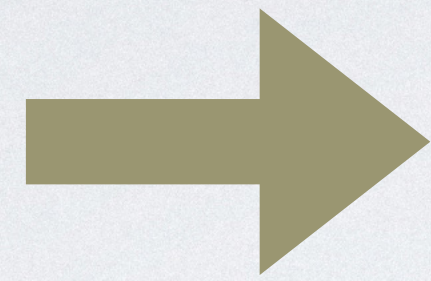
很快啊

类型、内存、并发安全

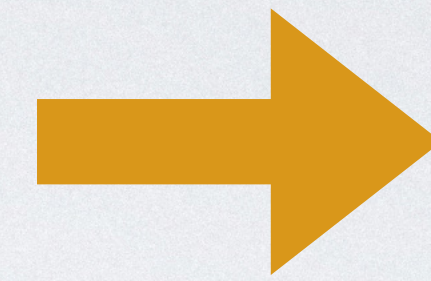
挺快的

趋势：语言的安全性越来越高

B



C



R_{ust}

无安全保障

非常慢

类型安全

很快啊

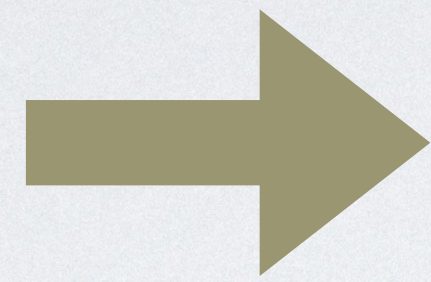
类型、内存、并发安全

挺快的

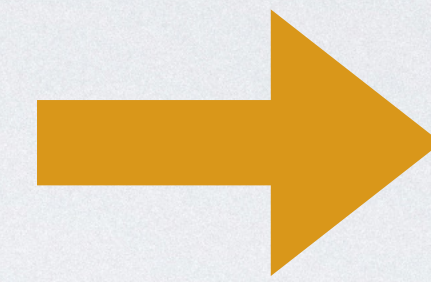
- 系统编程语言不仅要提供安全保障，还要提供性能保障

趋势：语言的安全性越来越高

B



C



R_{ust}

无安全保障

非常慢

类型安全

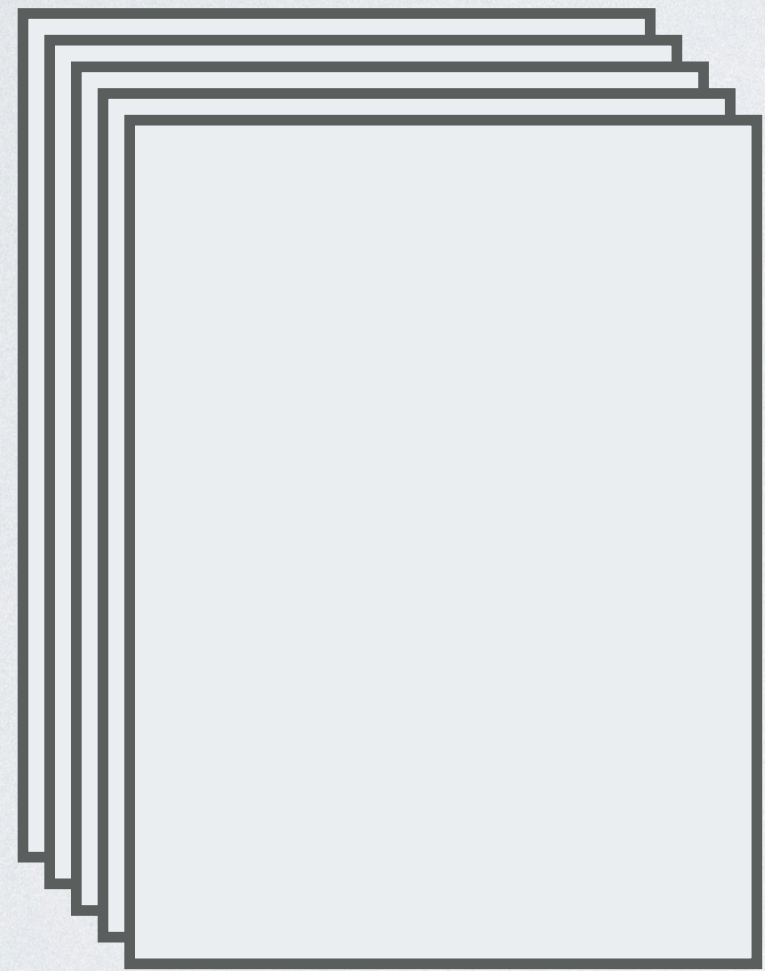
很快啊

类型、内存、并发安全

挺快的

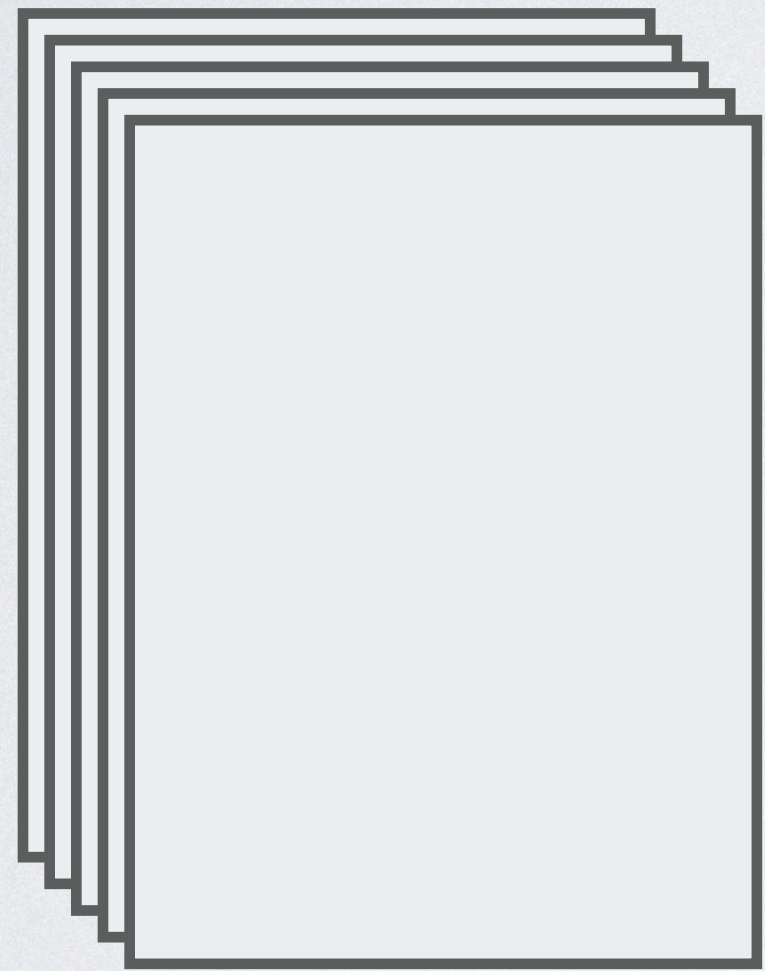
- 系统编程语言不仅要提供安全保障，还要提供性能保障
- 语言需要提供精细的对资源消耗的分析和管控

软件的资源消耗



软件

软件的资源消耗

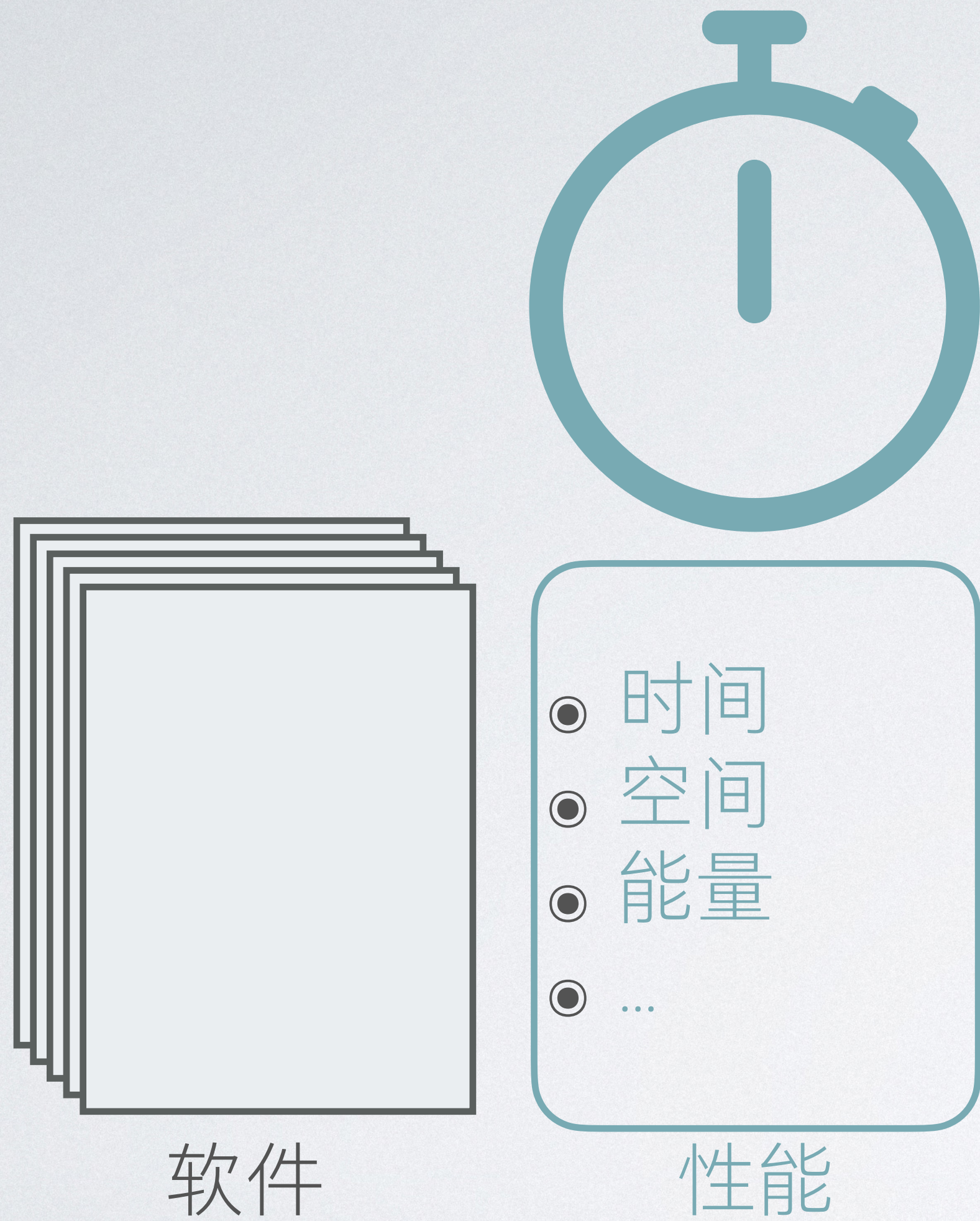


软件

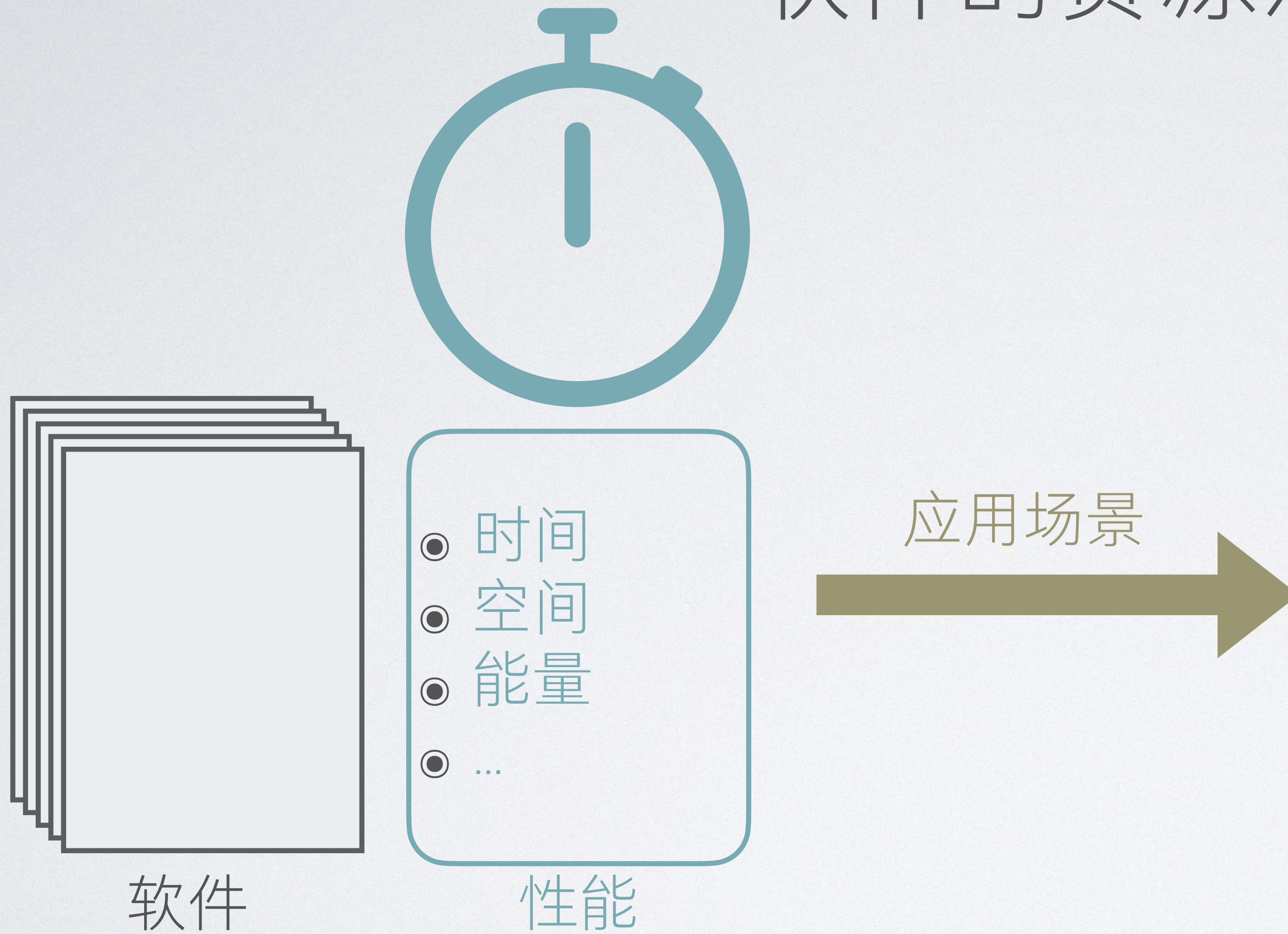


性能

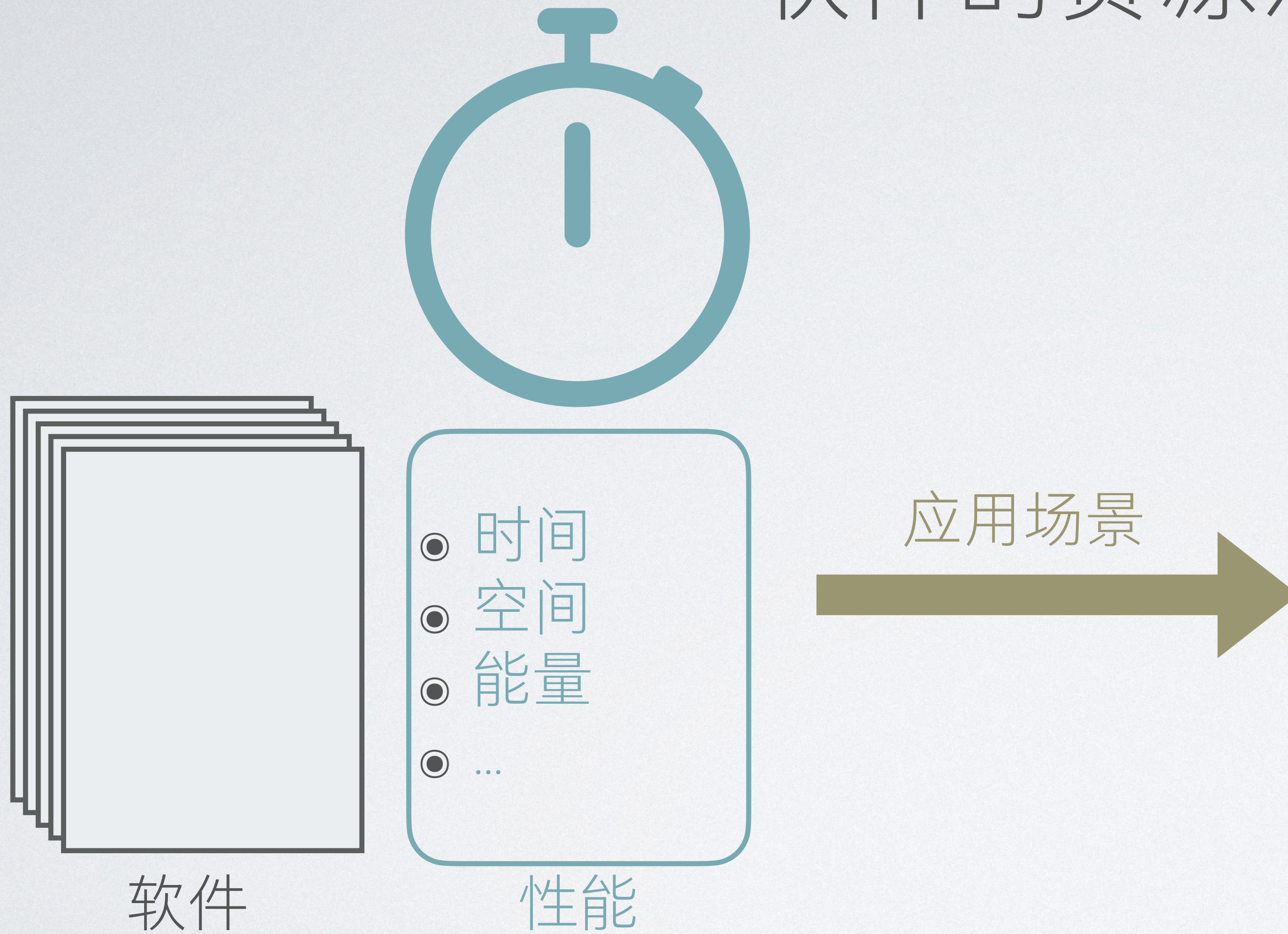
软件的资源消耗



软件的资源消耗

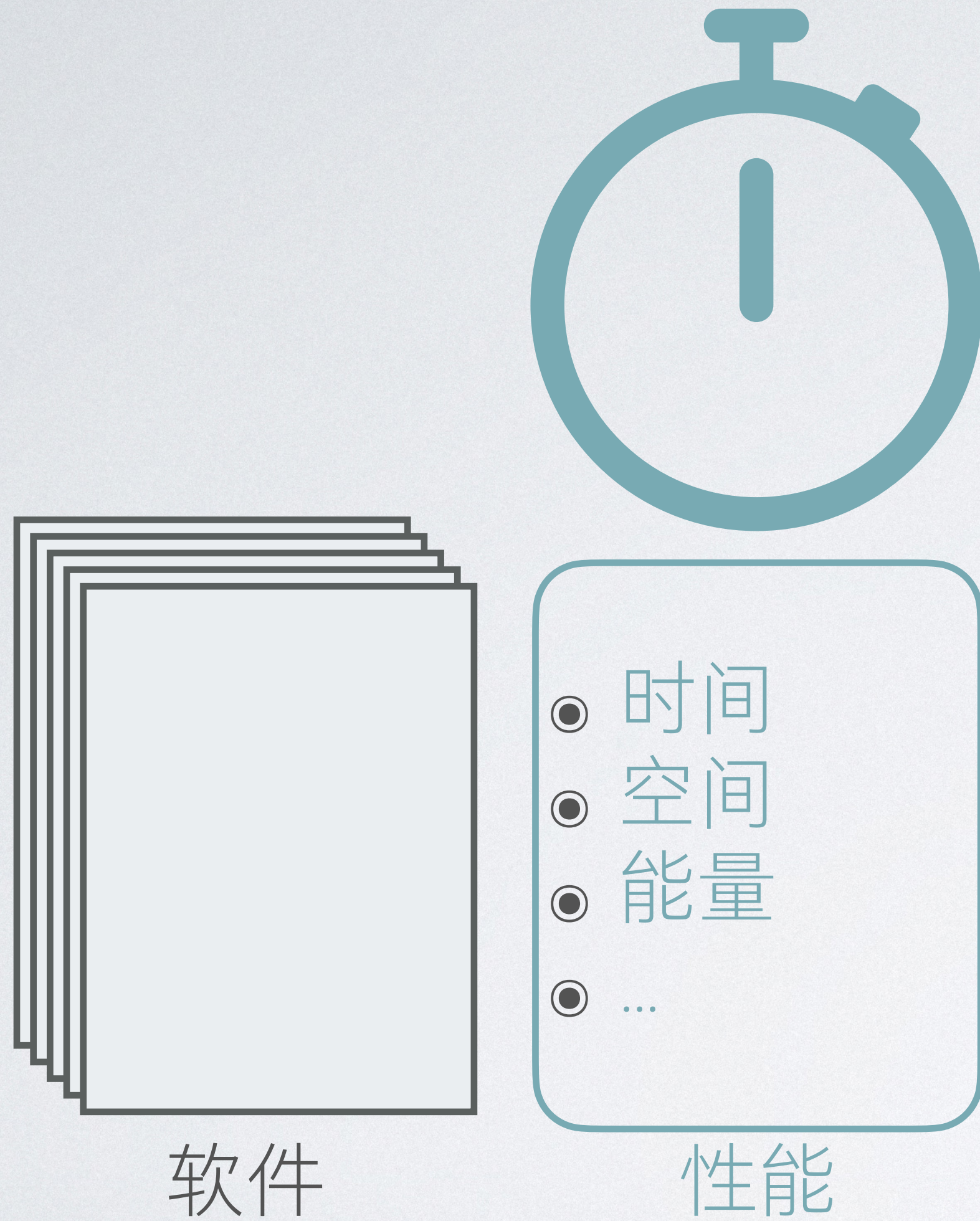


软件的资源消耗



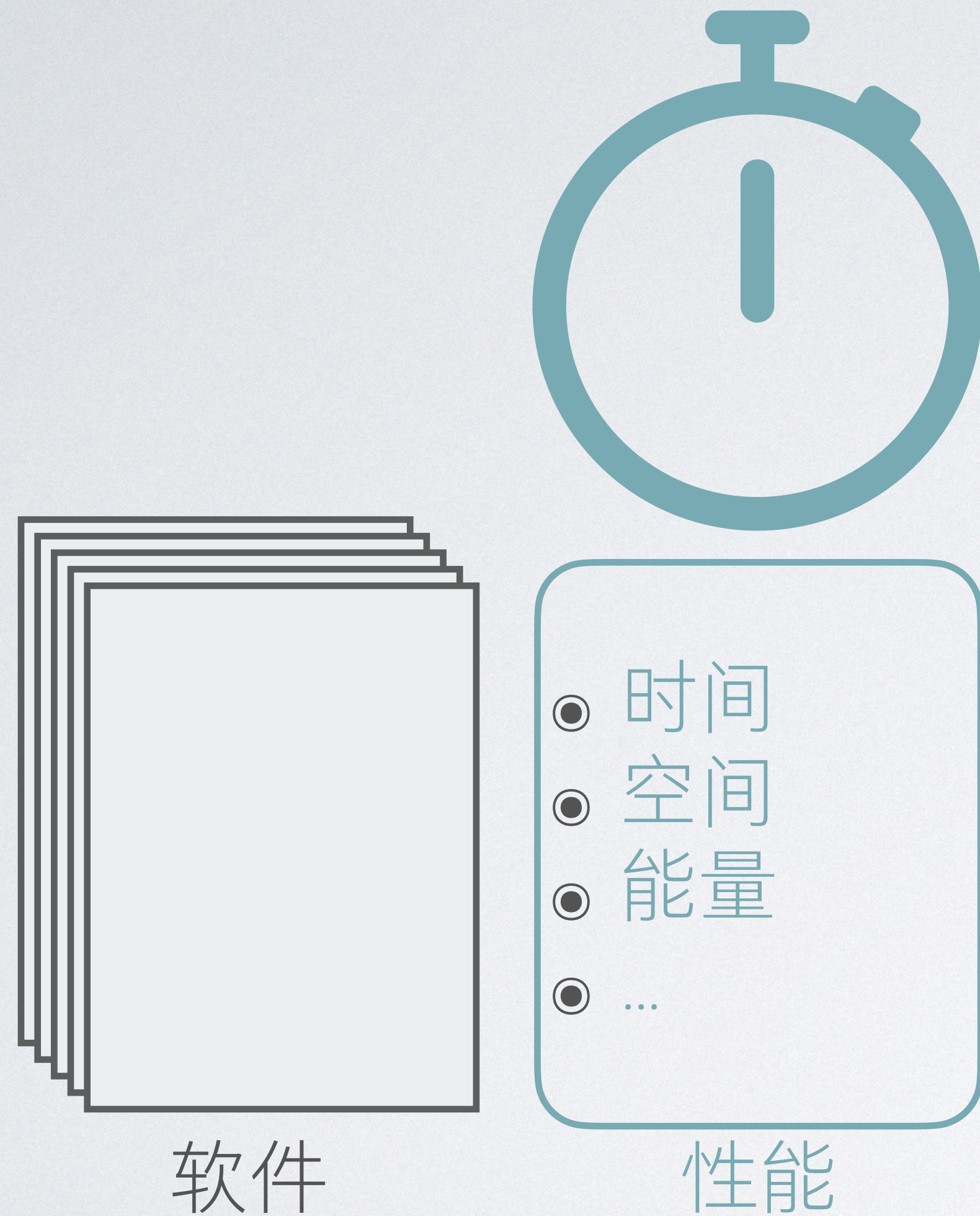
- 分析算法的时、空复杂度

软件的资源消耗



- 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



- 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



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

分析算法的时、空复杂度



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

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

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  
    }  
}
```

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  
    }  
}
```

$$O(|list|^2)$$

分析算法的时、空复杂度



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

```
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!

$$O(|list|^2)$$



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



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

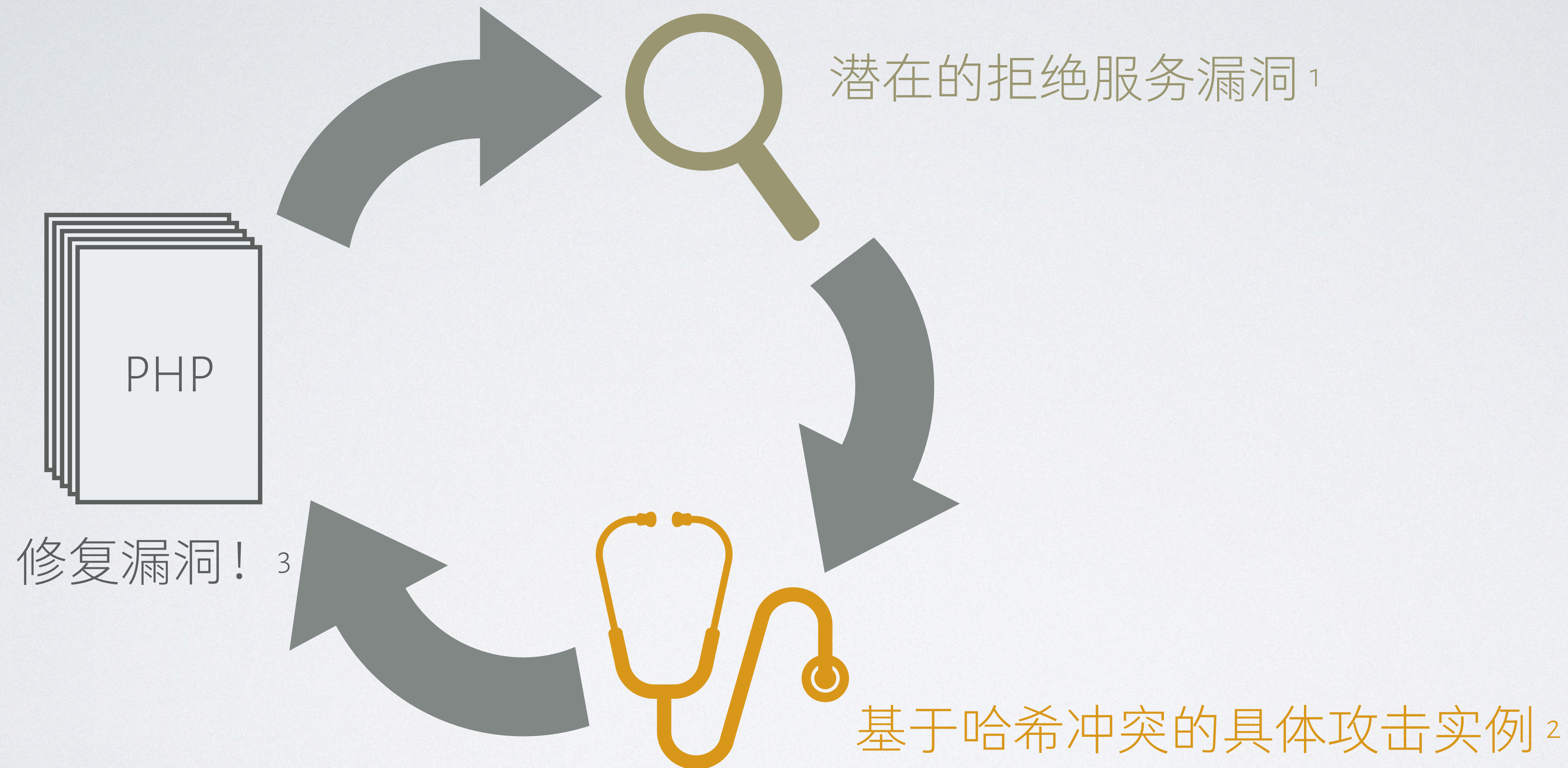


¹ CVE - CVE-2011-4885: <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-4885>.

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



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*.



工作 1: 资源制导最坏情况输入生成

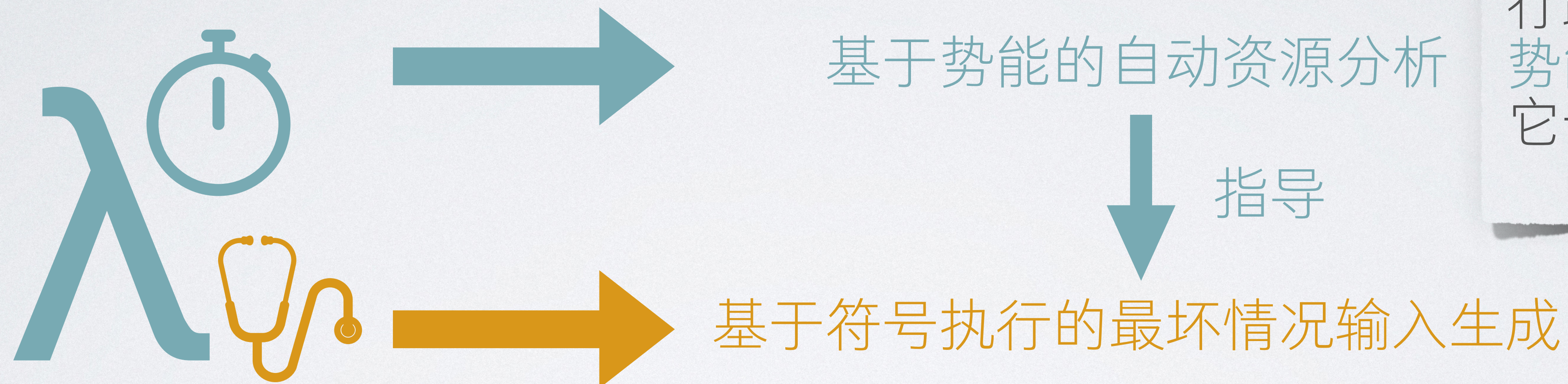
D. Wang and J. Hoffmann. Type-Guided Worst-Case Input Generation. In *POPL'19*.



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*.



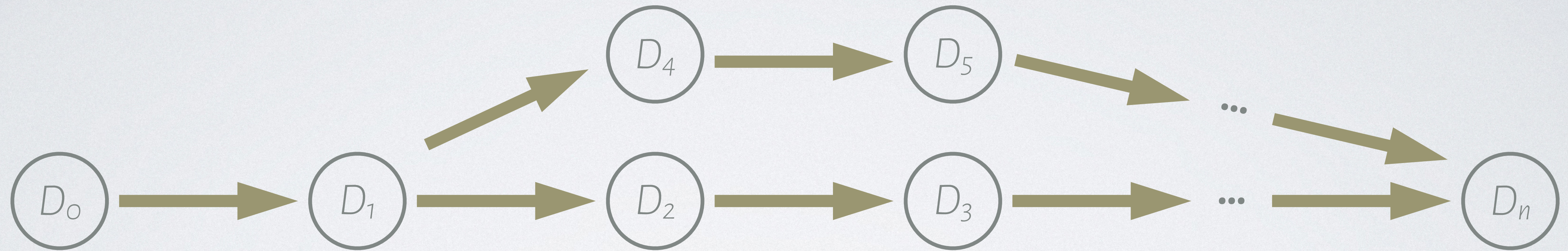
如果一条程序执行路径没有任何势能浪费，那么它一定具有最坏的资源消耗

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

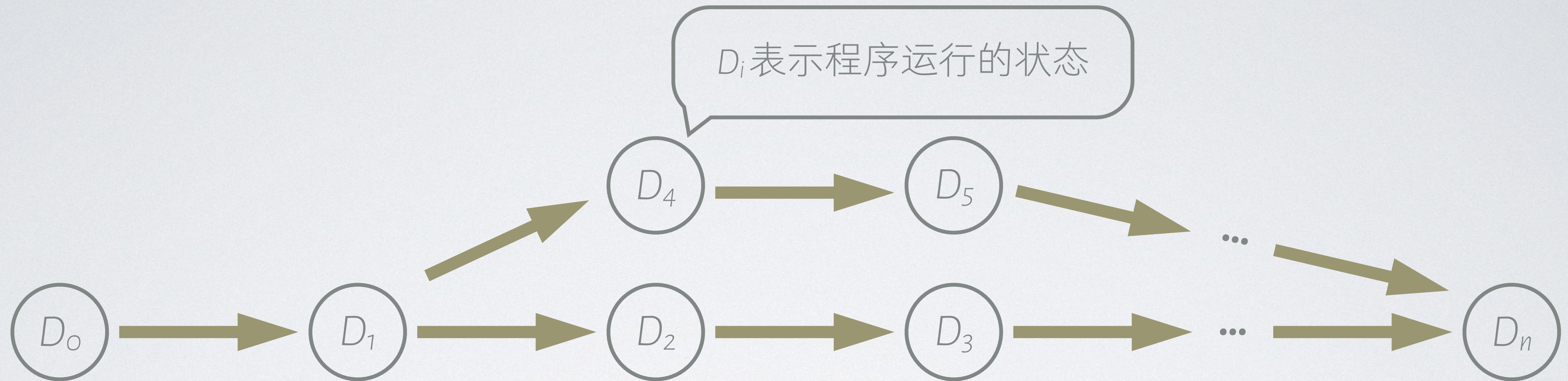


基于势能的自动资源分析

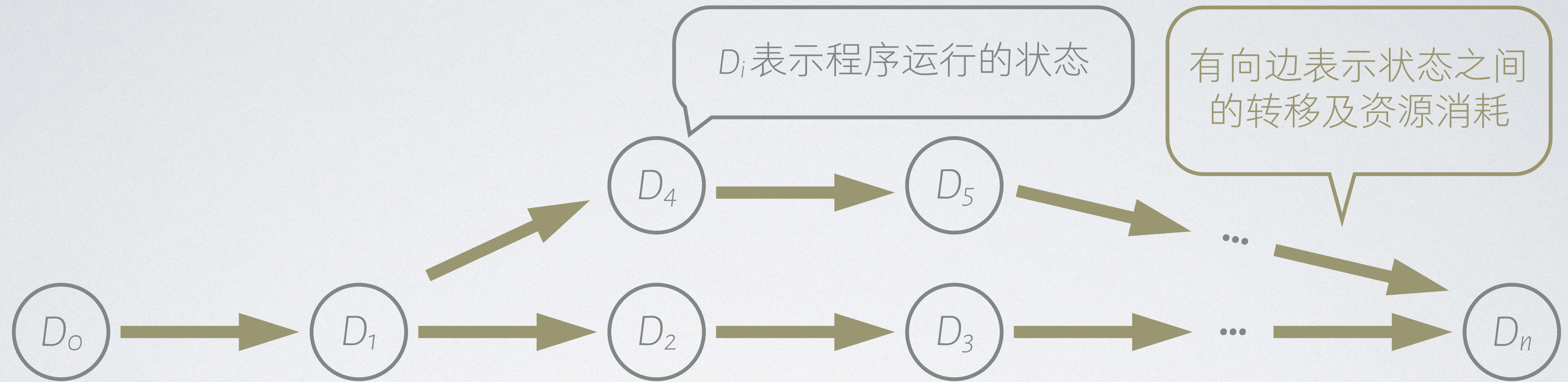
基于势能的自动资源分析



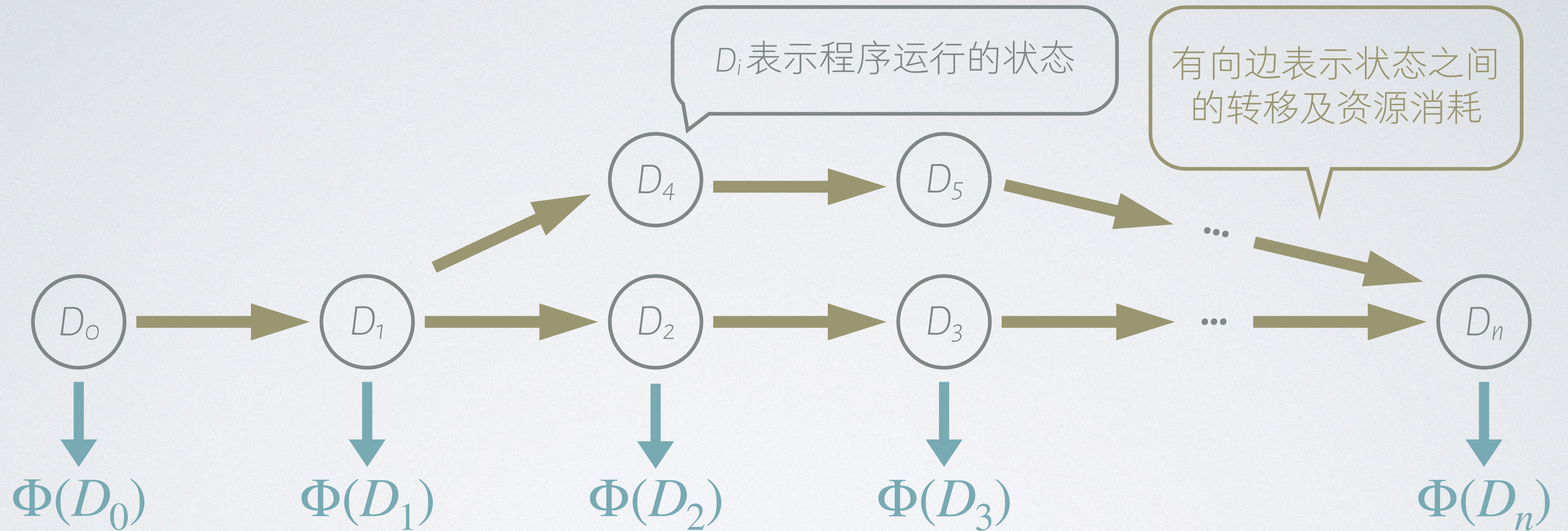
基于势能的自动资源分析



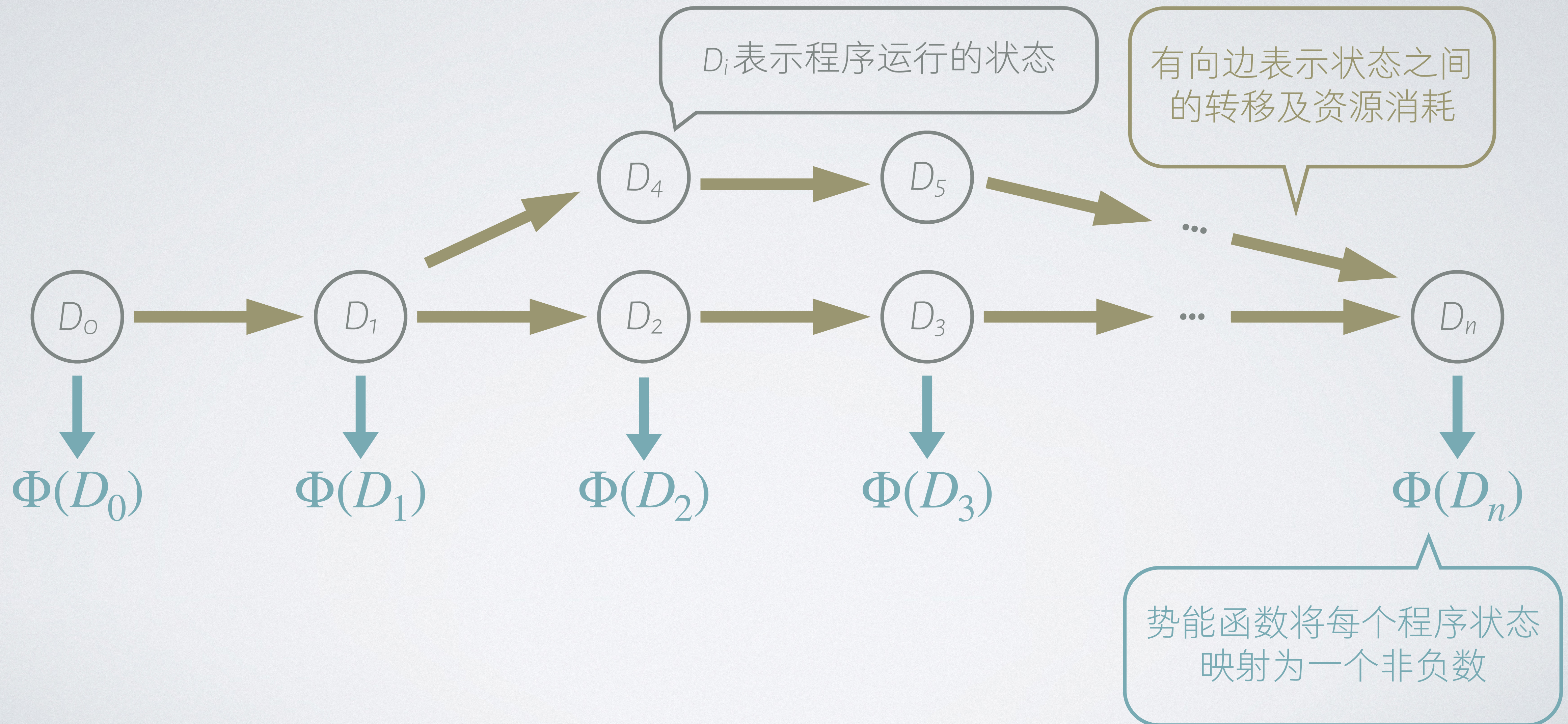
基于势能的自动资源分析



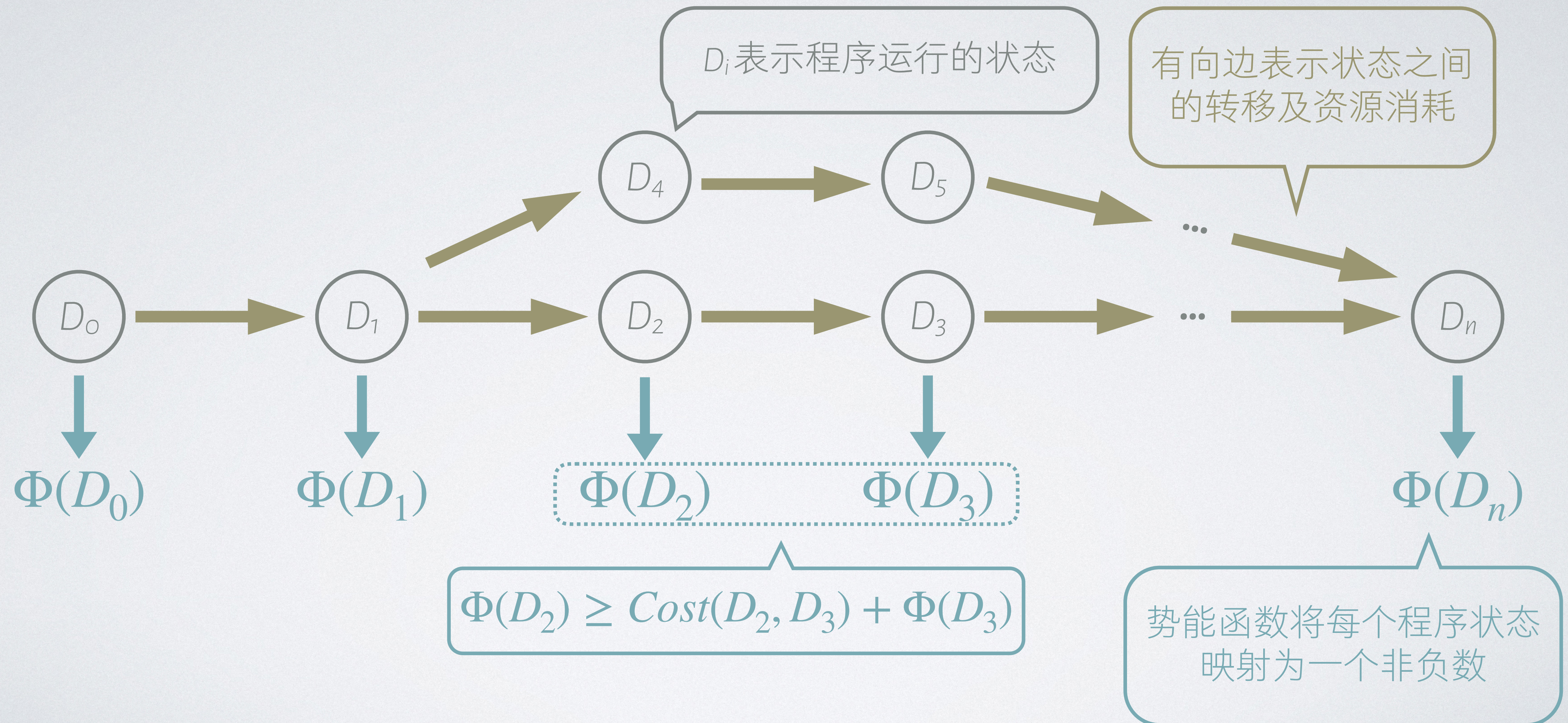
基于势能的自动资源分析



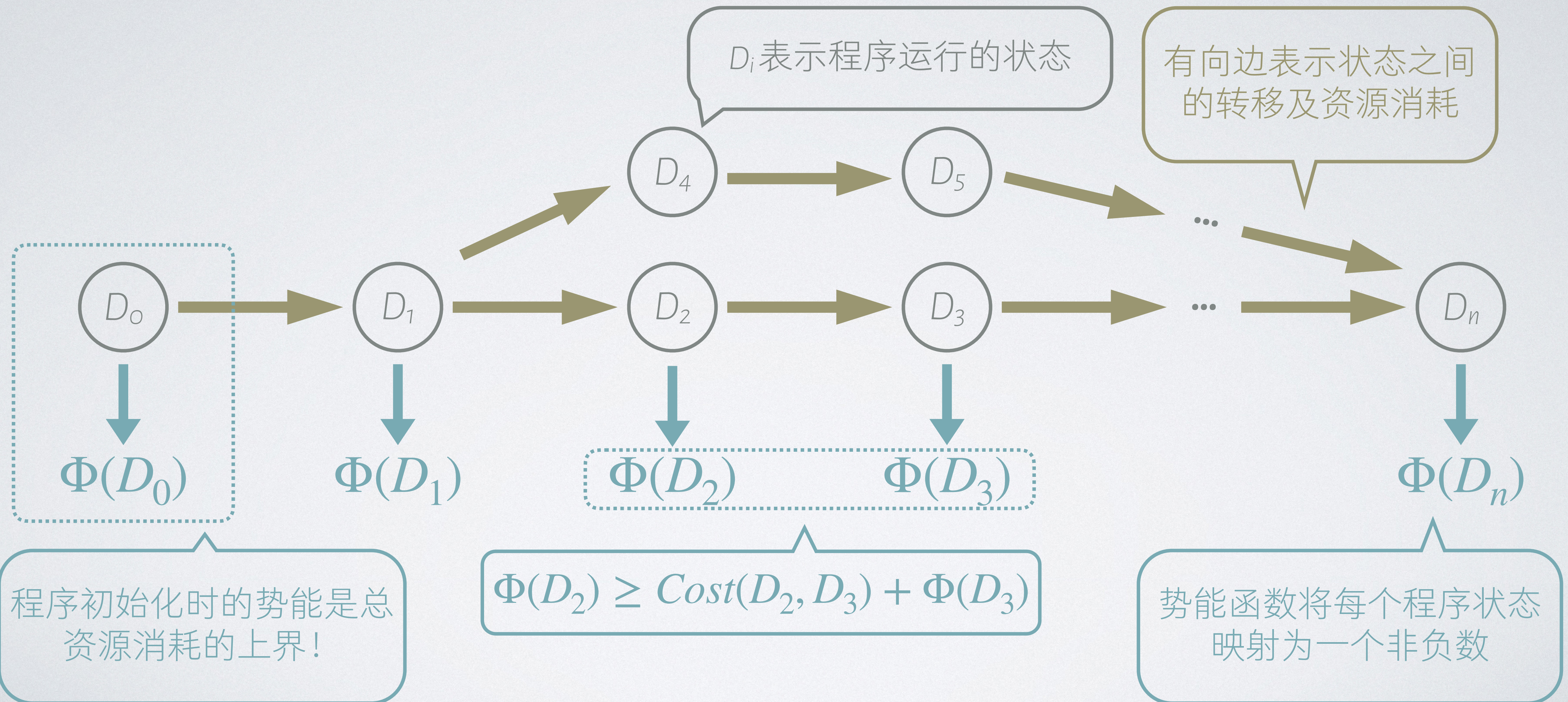
基于势能的自动资源分析



基于势能的自动资源分析

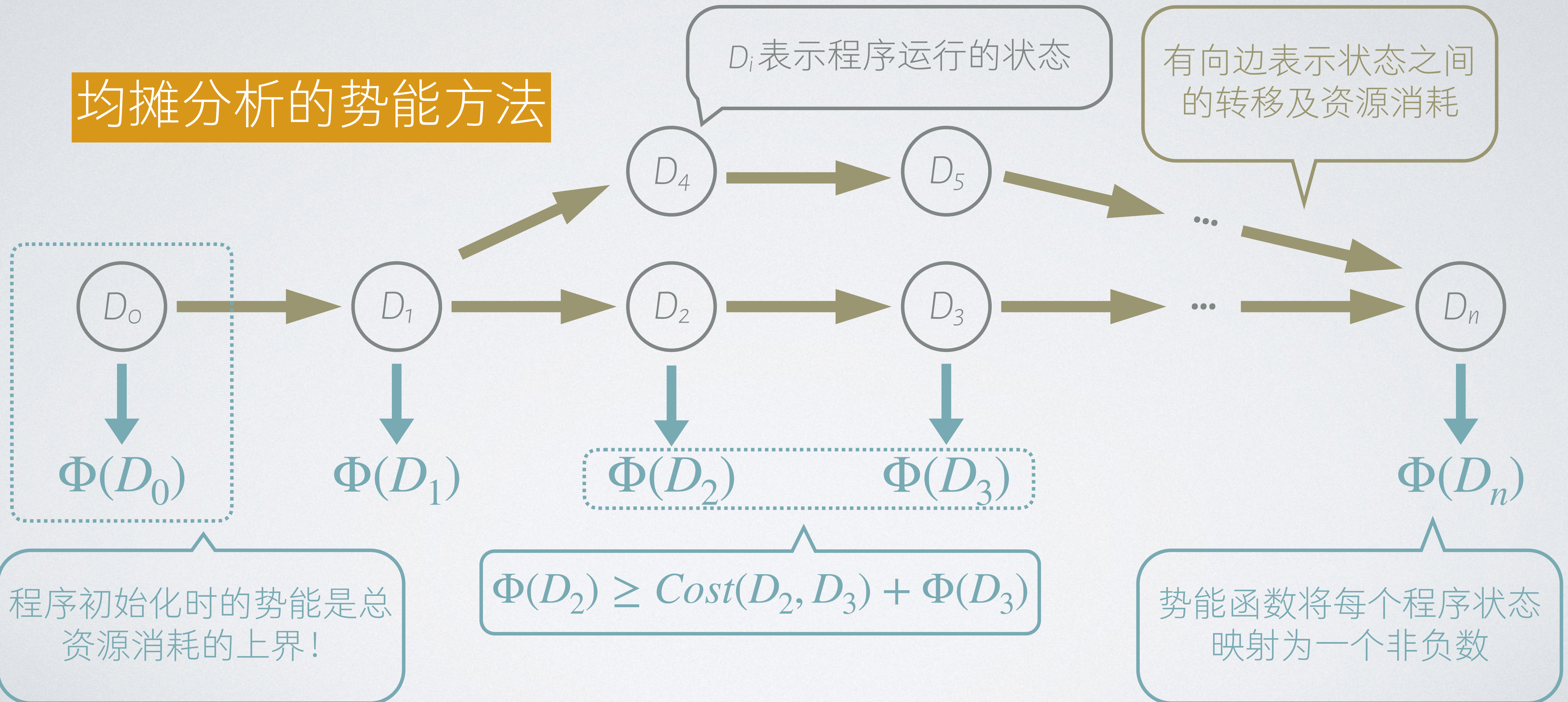


基于势能的自动资源分析



基于势能的自动资源分析

均摊分析的势能方法





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)

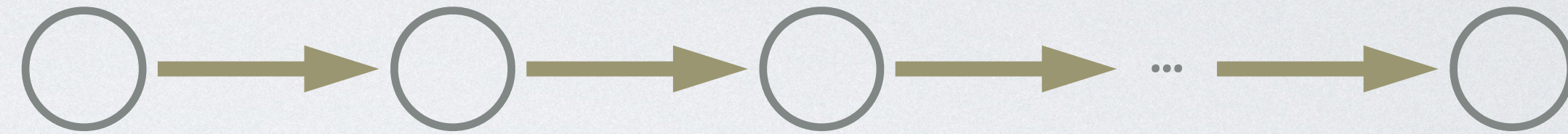


基于符号执行的最坏情况测试生成

输入的规约
(例如列表长度)



可能的执行路径 1

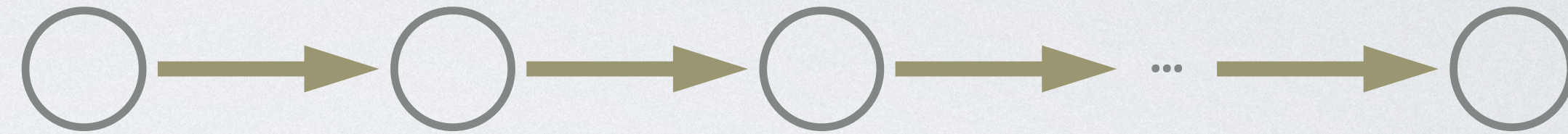


基于符号执行的最坏情况测试生成

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

输入的规约
(例如列表长度)



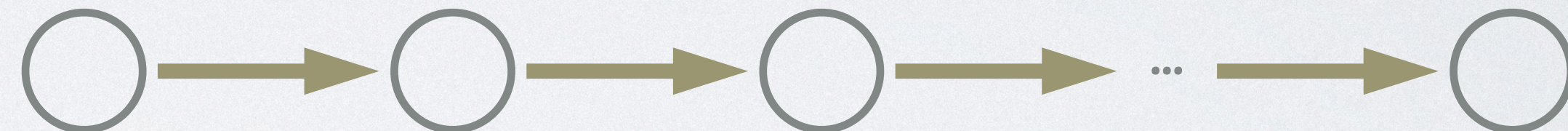
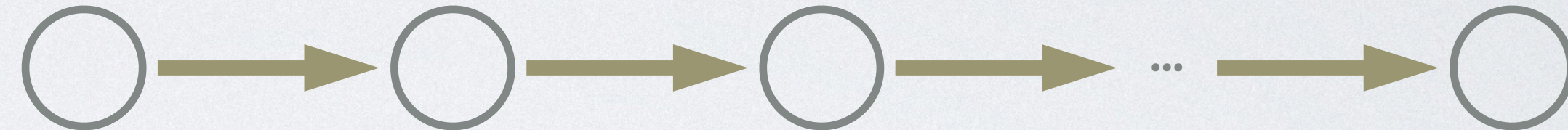
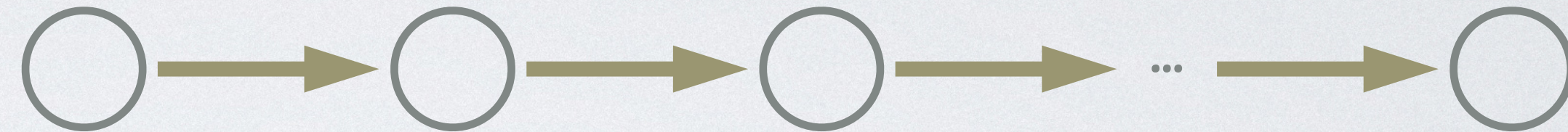
可能的执行路径 1

可能的执行路径 2

可能的执行路径 3

...

可能的执行路径 n



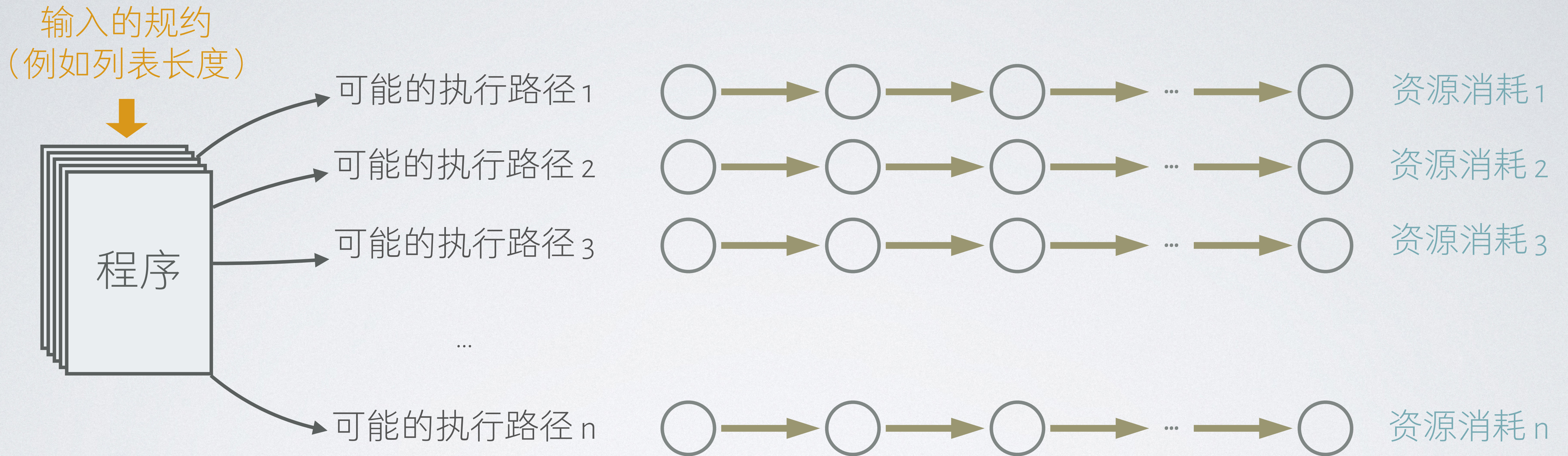
资源消耗 1

资源消耗 2

资源消耗 3

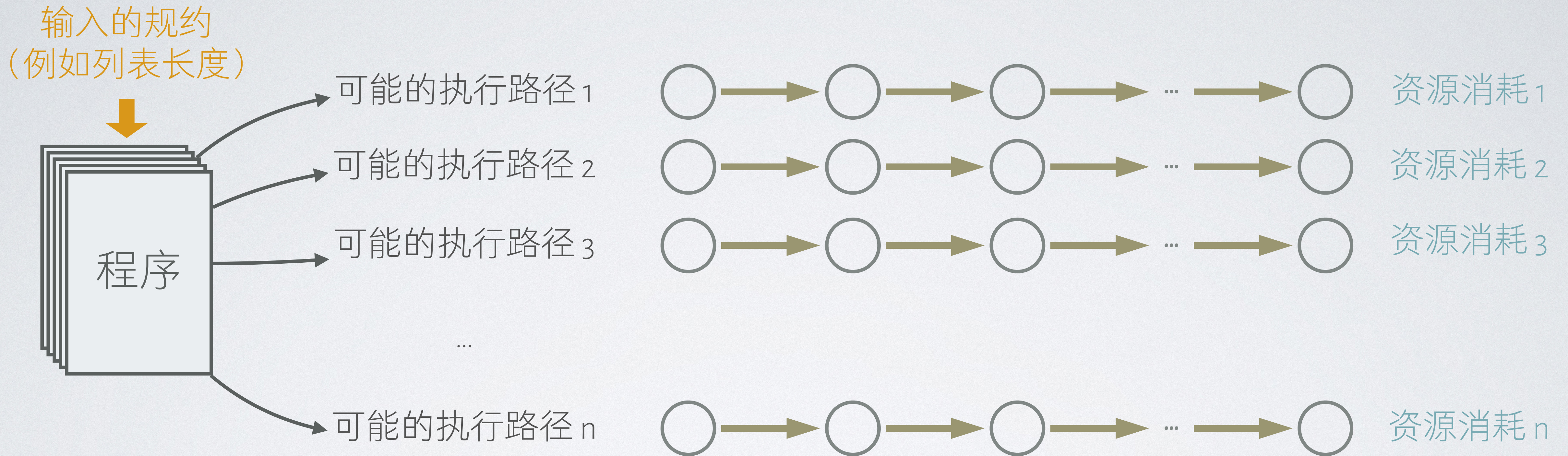
资源消耗 n

基于符号执行的最坏情况测试生成



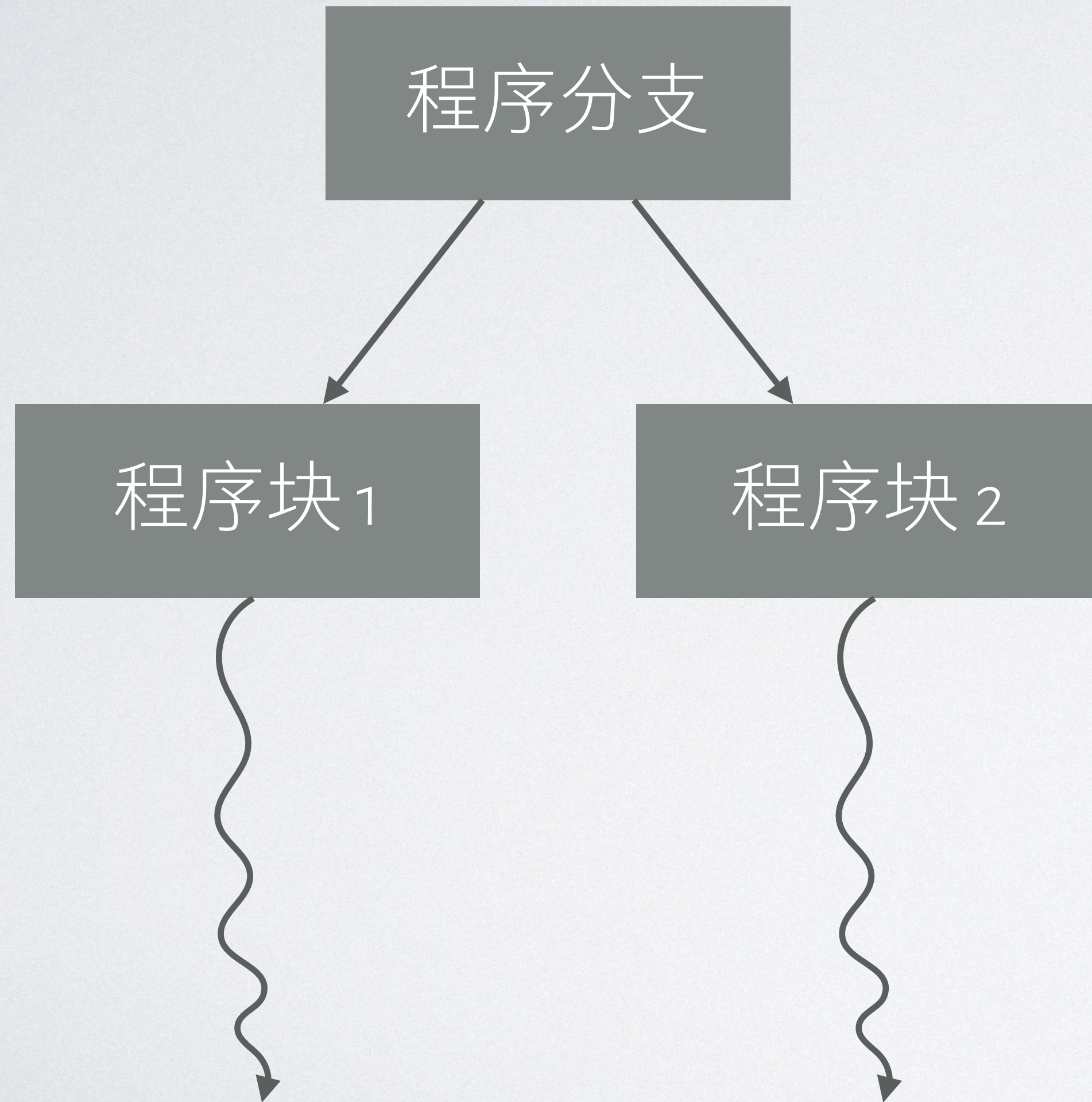
- 给定一个输入的规约，**搜索**一条具有**最大资源消耗**的程序执行路径

基于符号执行的最坏情况测试生成

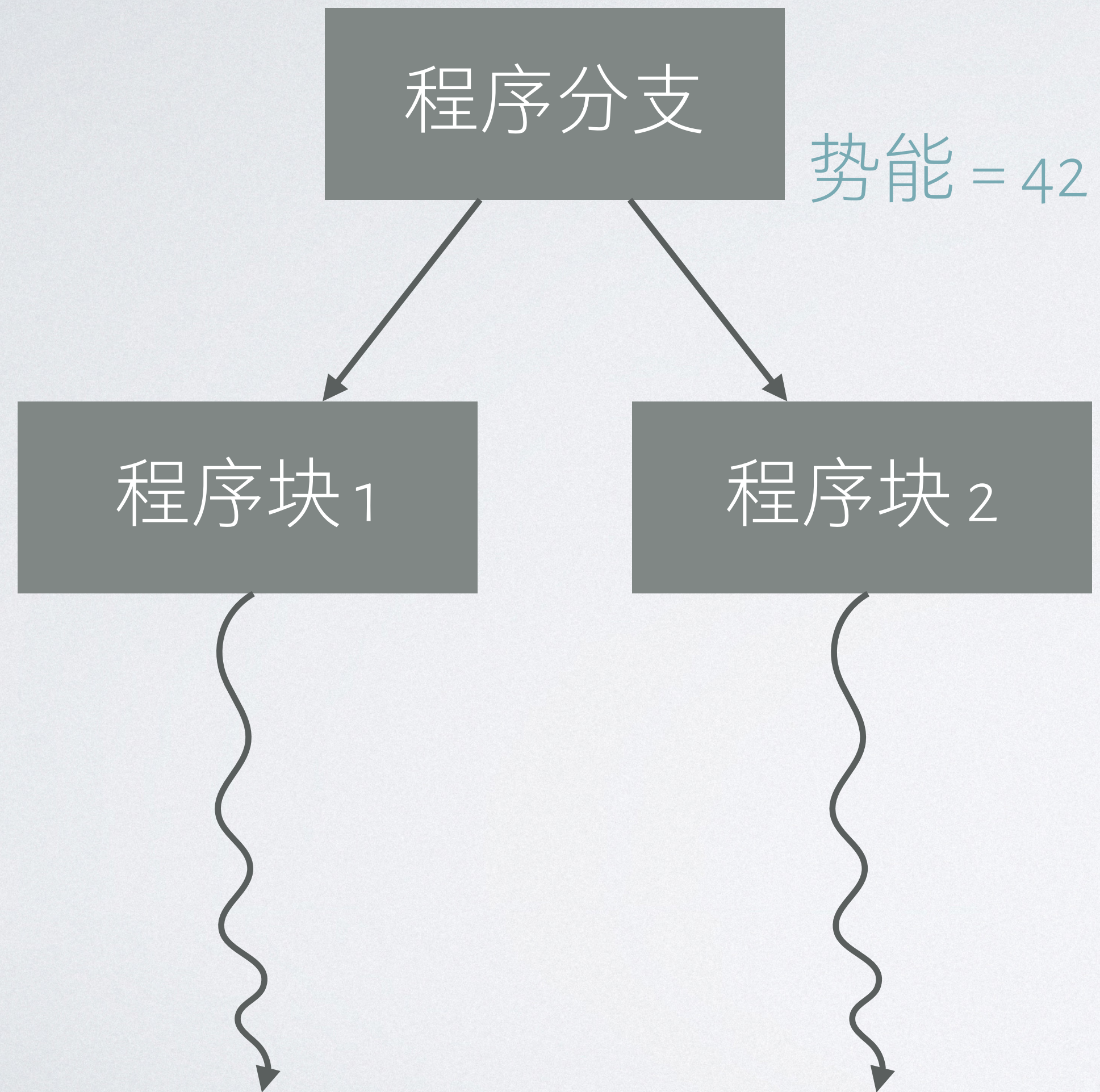


- 给定一个输入的规约，**搜索**一条具有**最大资源消耗**的程序执行路径
- 是否可以利用**静态资源分析**的结果来**指导**符号执行的**搜索**过程呢？

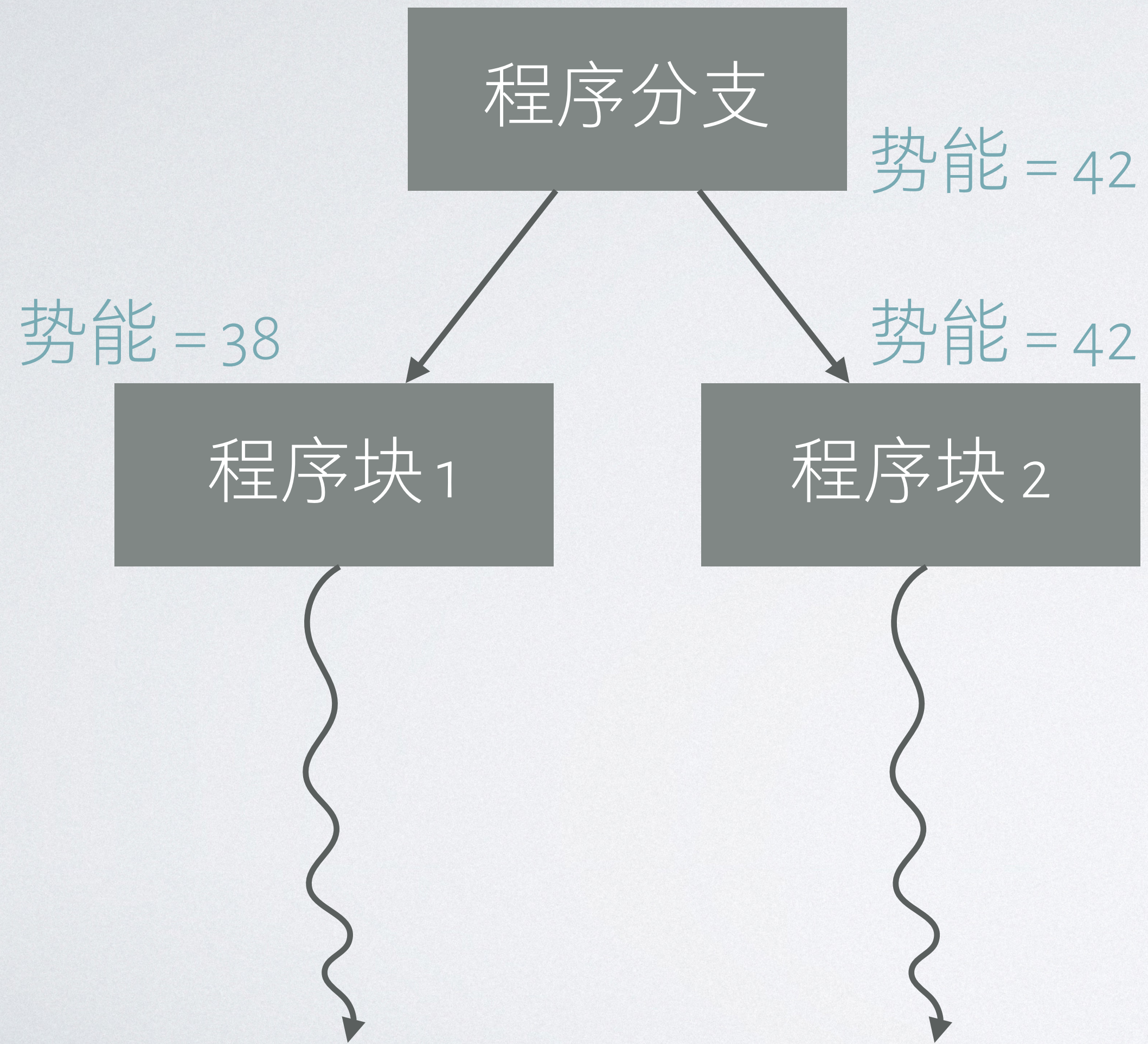
资源制导的符号执行



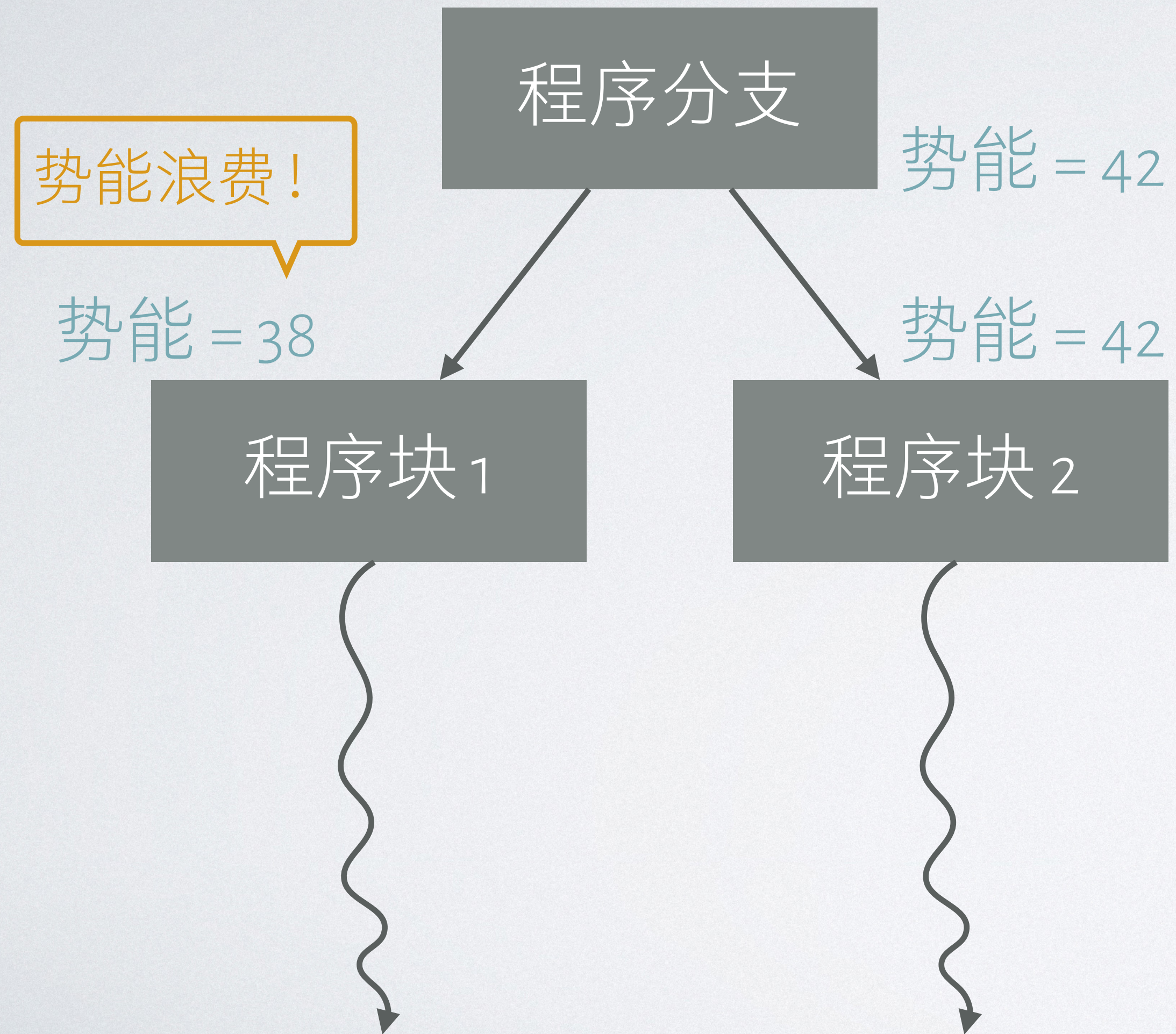
资源制导的符号执行



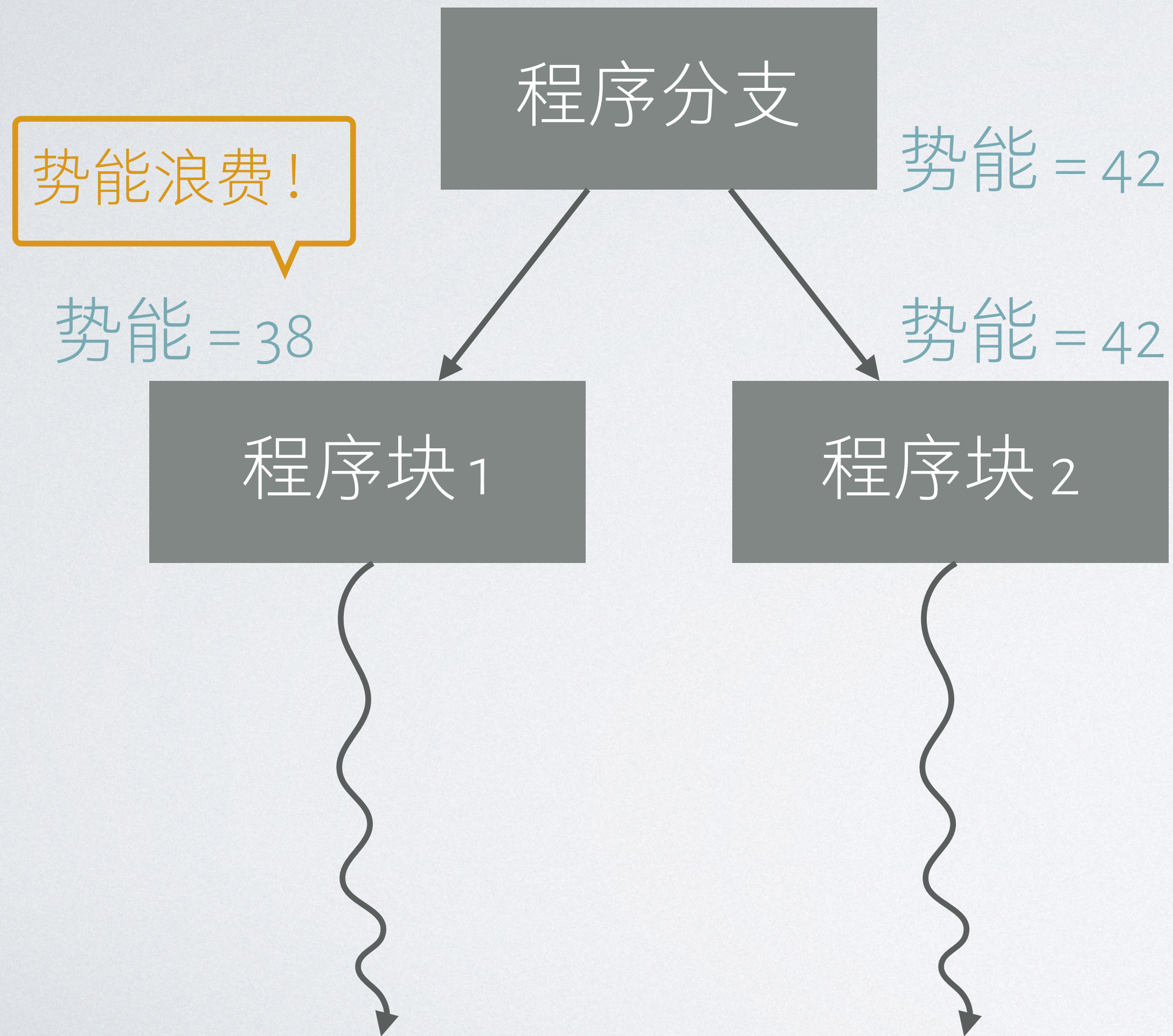
资源制导的符号执行



资源制导的符号执行

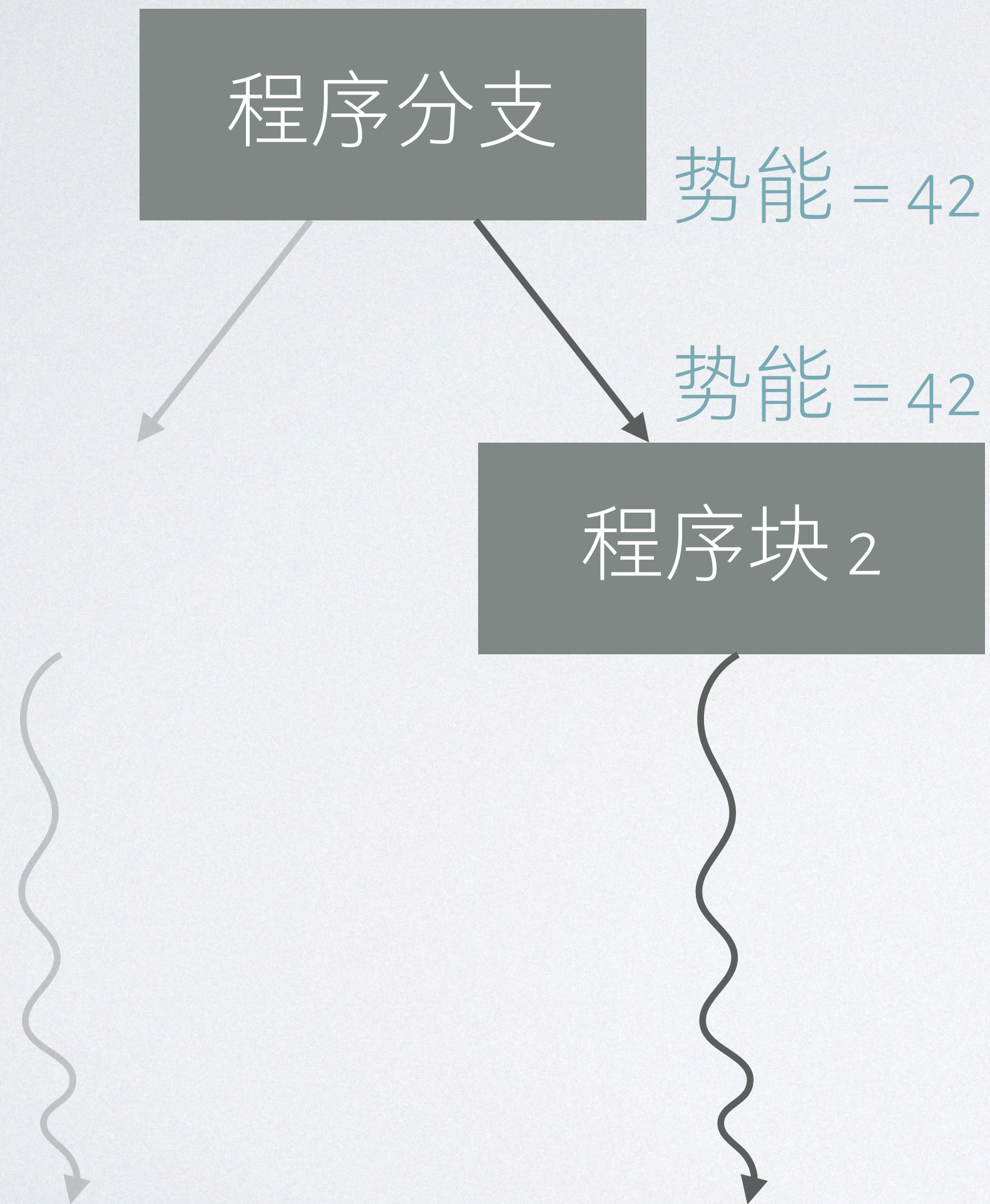


资源制导的符号执行



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



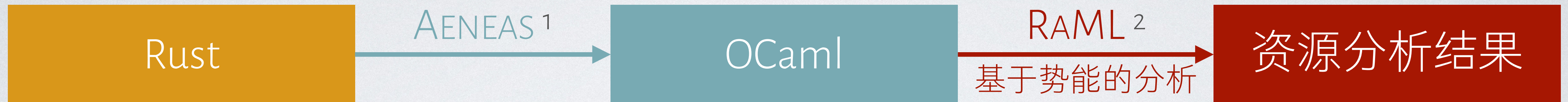
如果一条程序执行路径**没有**任何**势能浪费**，那么它一定具有**最坏的资源消耗**

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*.



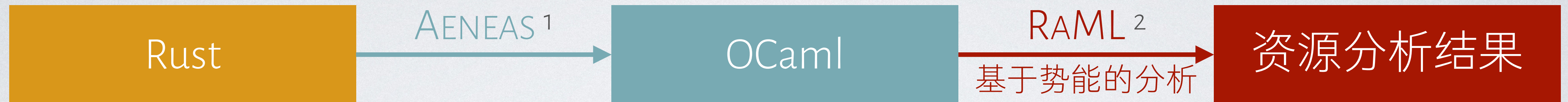
¹ S. Ho and J. Protzenko. 2022. AENEAS: Rust Verification by Functional Translation. In ICFP'22.

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.

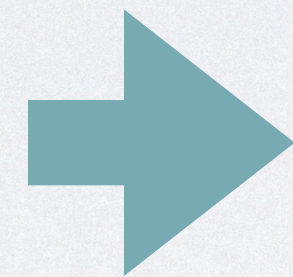
² J. Hoffmann, A. Das, and S.-C. Weng. 2017. Towards Automatic Resource Bound Analysis for OCaml. In POPL'17.

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.

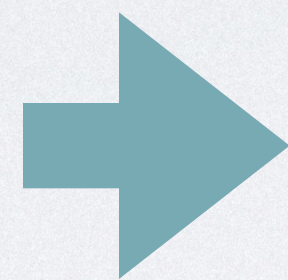
² J. Hoffmann, A. Das, and S.-C. Weng. 2017. Towards Automatic Resource Bound Analysis for OCaml. In POPL'17.

工作 2：通过转译分析 Rust 程序的资源消耗

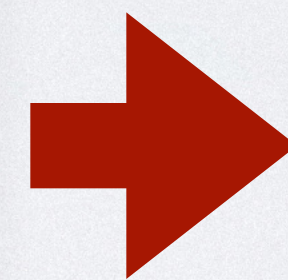
X. Peng and D. Wang. Rust Resource Analysis by Functional Translation. *Working Paper*.



```
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
```



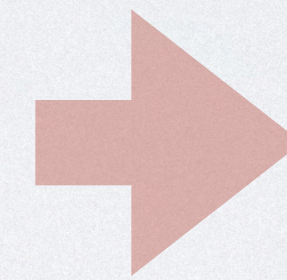
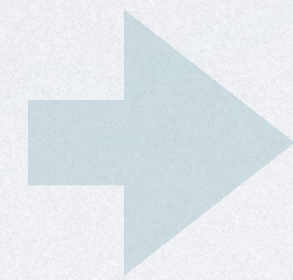
```
== list_consume_twice_fwd_back :  
  
Simplified bound:  
1*M  
where  
M is the number of ::-nodes of the argument
```

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*.



通过 Rust 类型中的所有
权信息提升分析精度

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.



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*.

T. Knoth, D. Wang, A. Reynolds, J. Hoffmann, and N. Polikarpova. Liquid Resource Types. In *ICFP'20*.



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*.

T. Knoth, D. Wang, A. Reynolds, J. Hoffmann, and N. Polikarpova. Liquid Resource Types. In *ICFP'20*.

- 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*.

T. Knoth, D. Wang, A. Reynolds, J. Hoffmann, and N. Polikarpova. Liquid Resource Types. In *ICFP'20*.

- 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*.

T. Knoth, D. Wang, A. Reynolds, J. Hoffmann, and N. Polikarpova. Liquid Resource Types. In *ICFP'20*.

- 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

```
rep : (n: int) -> a ->  
{ v: List a | len(v) = n }
```



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*.

T. Knoth, D. Wang, A. Reynolds, J. Hoffmann, and N. Polikarpova. Liquid Resource Types. In *ICFP'20*.

- 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

```
rep : (n: int) -> a ->  
{ v: List a | len(v) = n }
```

```
let rec rep n x =  
  if n <= 0  
  then []  
  else x :: (rep (n - 1) x)
```



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*.

T. Knoth, D. Wang, A. Reynolds, J. Hoffmann, and N. Polikarpova. Liquid Resource Types. In *ICFP'20*.

- 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

```
rep : (n: int) -> a ->  
{ v: List a | len(v) = n }
```

```
let rec rep n x =  
  if n <= 0  
  then []  
  else x :: (rep (n - 1) x)
```

将程序合成问题转化为
搜索一个符合目标类型的程序



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

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

一个满足 Ψ 的具有类型 B 的值 v

$\{ v: \text{Int} \mid v \geq 0 \}$

一个非负整数



以 Refinement Types 为规约

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

一个满足 Ψ 的具有类型 B 的值 v

$\{ v: \text{Int} \mid v \geq 0 \}$

一个非负整数

$(xs: \text{List } a) \rightarrow \{ v: \text{List } a \mid \text{len}(v) = \text{len}(xs) + 1 \}$

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

势能：数值型标注



Φ

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

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



约束：布尔型标注

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

势能：数值型标注



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

约束：布尔型标注



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

一个非负整数，携带了5倍于其自身值的势能

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

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

势能：数值型标注

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

约束：布尔型标注

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

一个非负整数，携带了5倍于其自身值的势能

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

一个列表，其中的每个非负元素
携带了1单位的势能

$\text{ite} = \text{if-then-else}$

设计了将两种标注整合的类型系统和
基于该类型系统的程序合成算法



资源制导的程序合成

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

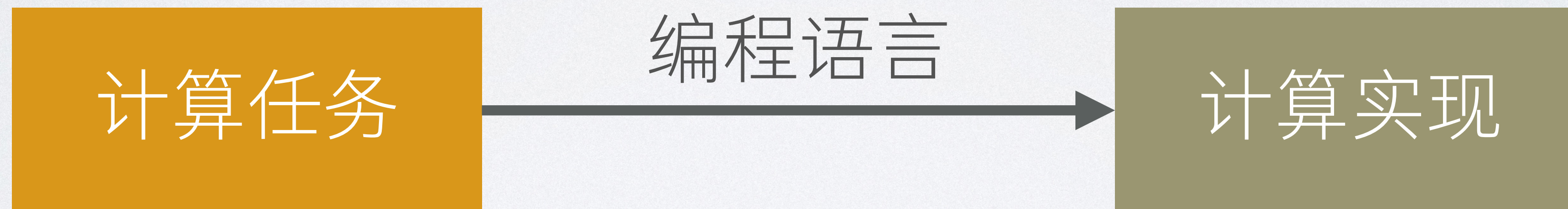
```
let rec common xs ys =  
  match xs with  
  | [] -> []  
  | x::xt ->  
    match ys with  
    | [] -> []  
    | y::yt ->  
      if x < y then common xt ys  
      else if y < x then common xs yt  
      else x::(common xt yt)
```

RESYN: 线性时间复杂度

研究新范式的动机

动机一：让程序更加安全
(保证正确高效地完成计算)

动机二：驾驭智能的计算
(把智能系统视作抽象算力)





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

机器语言



汇编语言

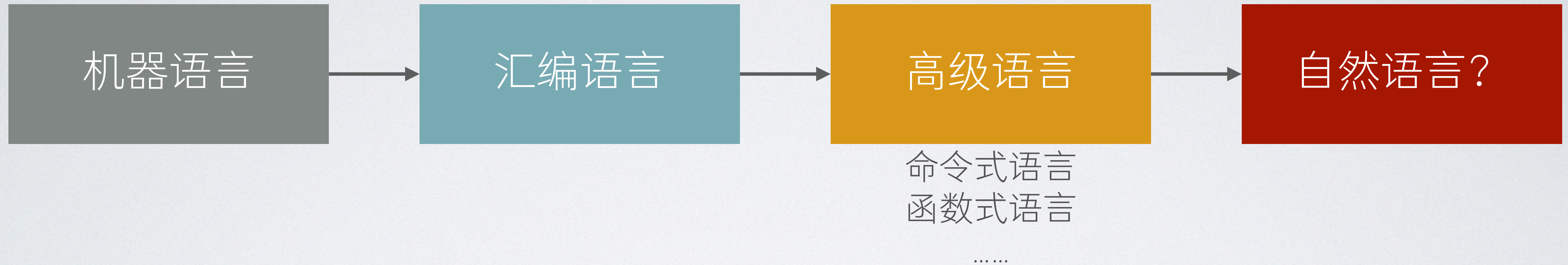


趋势：语言的抽象层次越来越高

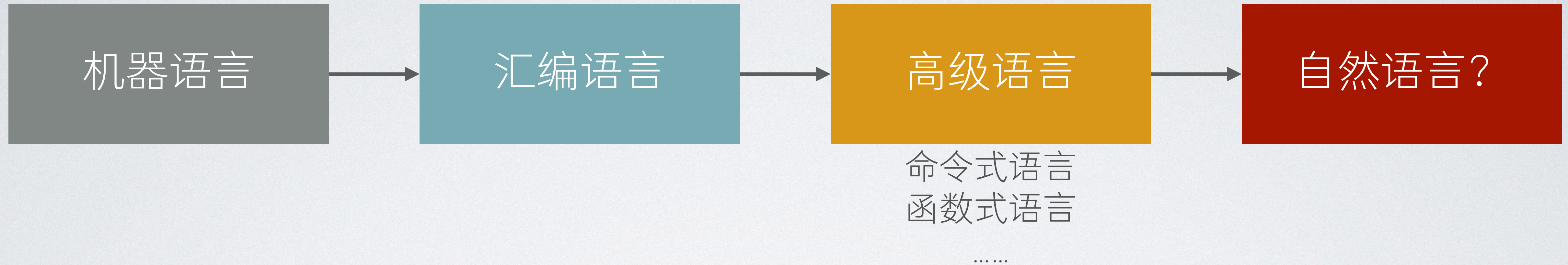




趋势：语言的抽象层次越来越高

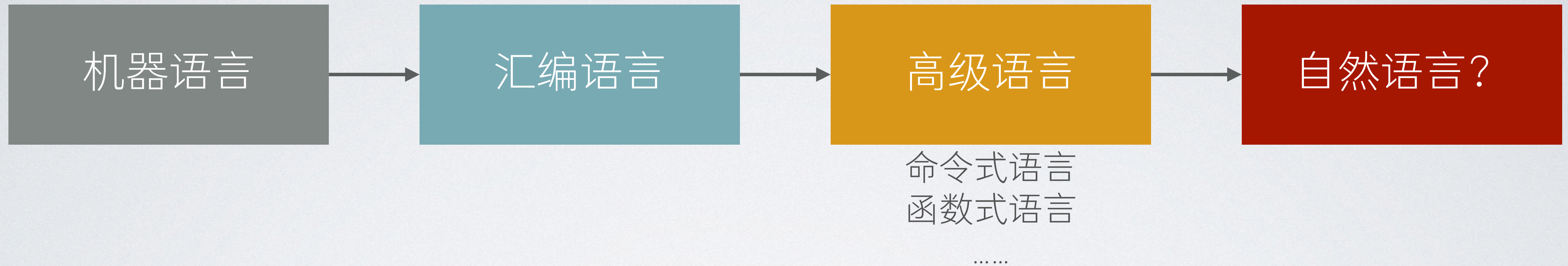


趋势：语言的抽象层次越来越高



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

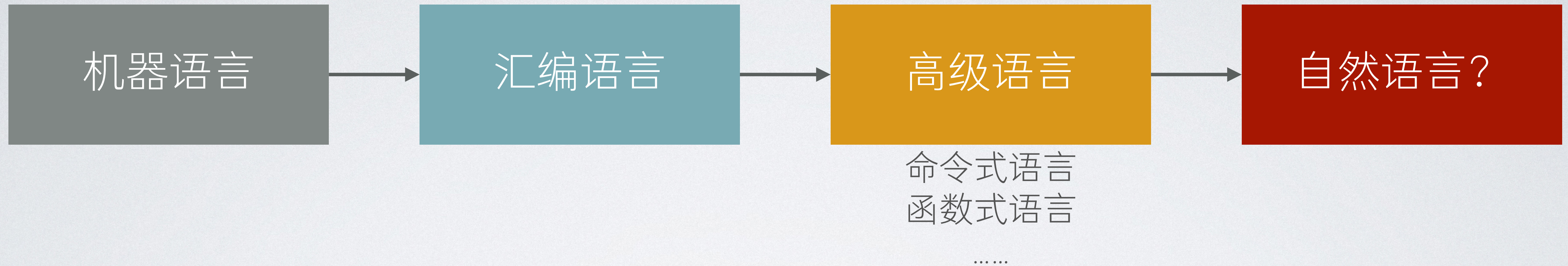
趋势：语言的抽象层次越来越高



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



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？

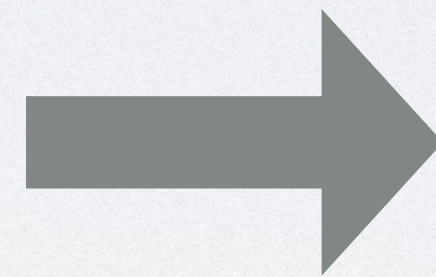


SCENIC: 驾驶场景描述语言

```
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
```

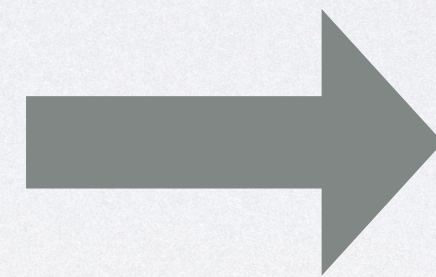
SCENIC: 驾驶场景描述语言

```
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
```



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



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)
```

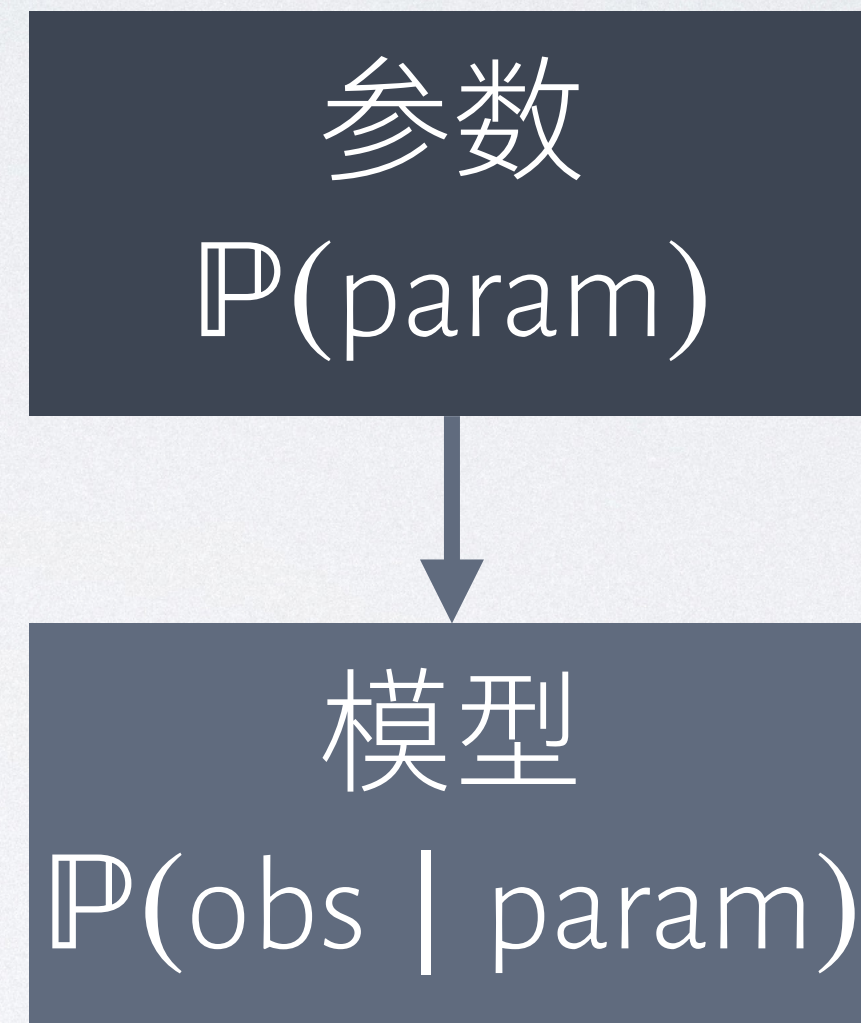
参数
 $\mathbb{P}(\text{param})$

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:  
    ...
```





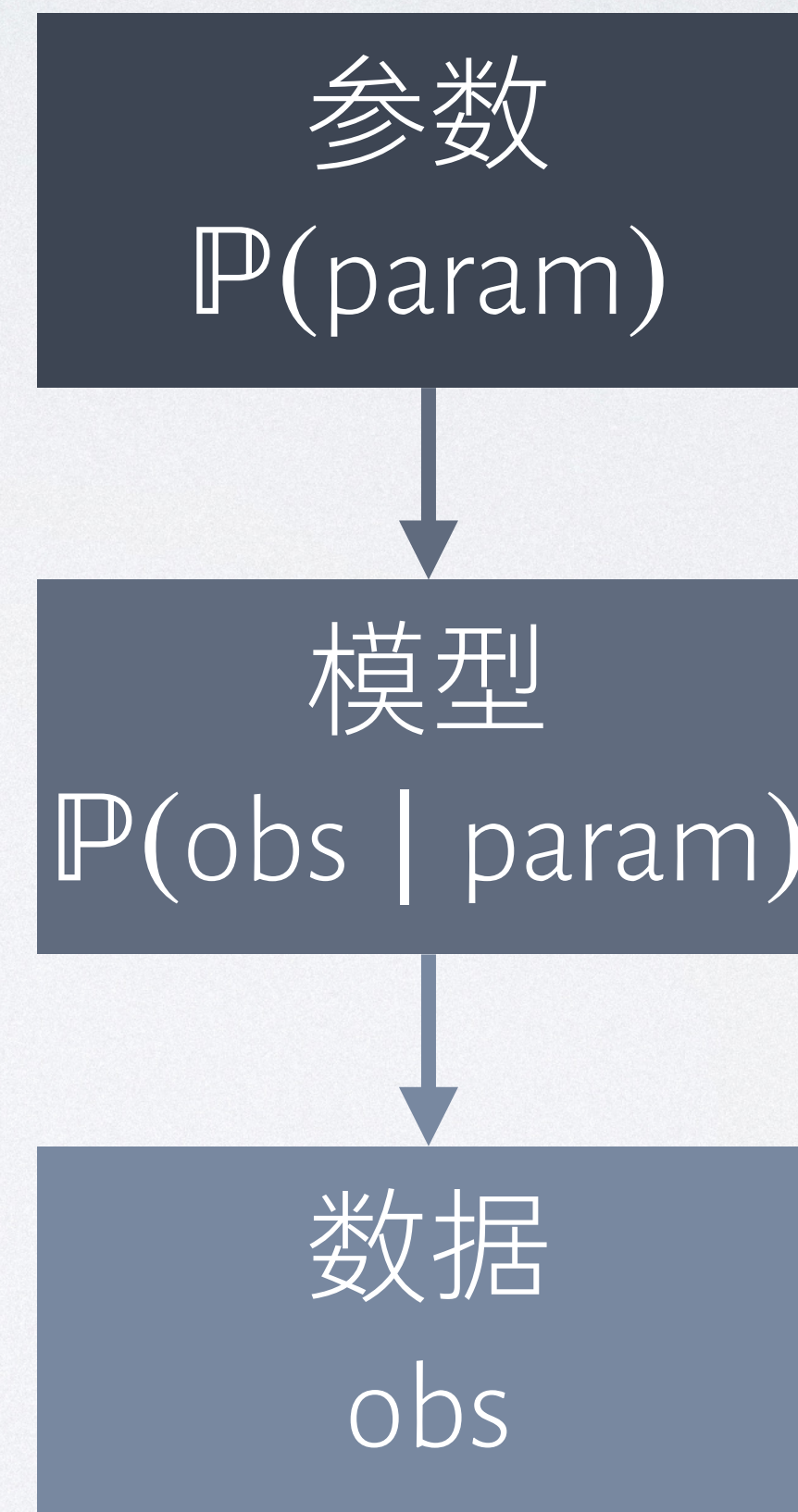
概率编程：通过程序描述概率模型

```
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)
```



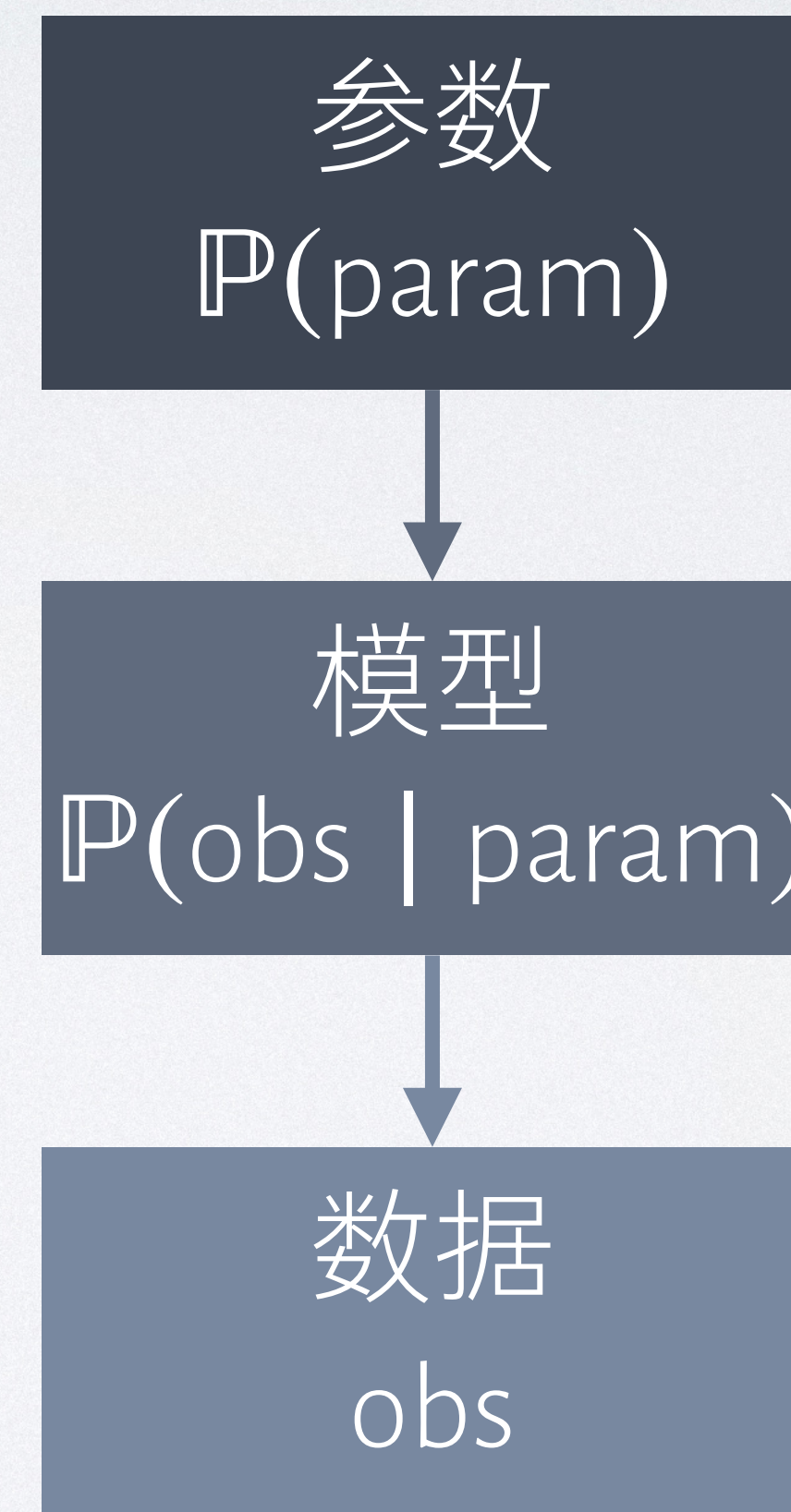
概率编程：通过程序描述概率模型

```
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)
```



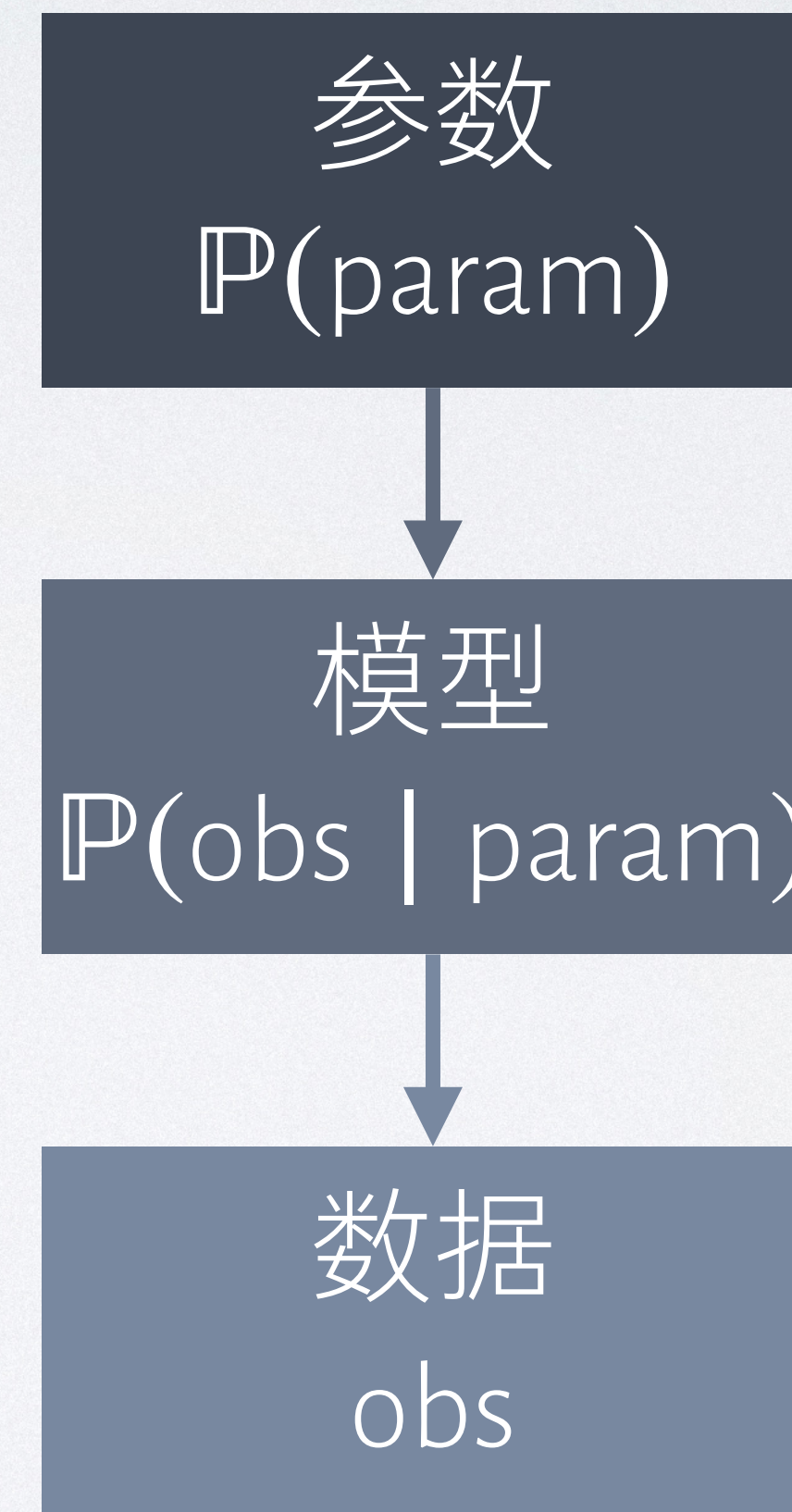
贝叶斯推断
 $\mathbb{P}(\text{param} \mid \text{obs})$

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)
```



听众喜欢我在概率编程上的工作的后验概率是多少？

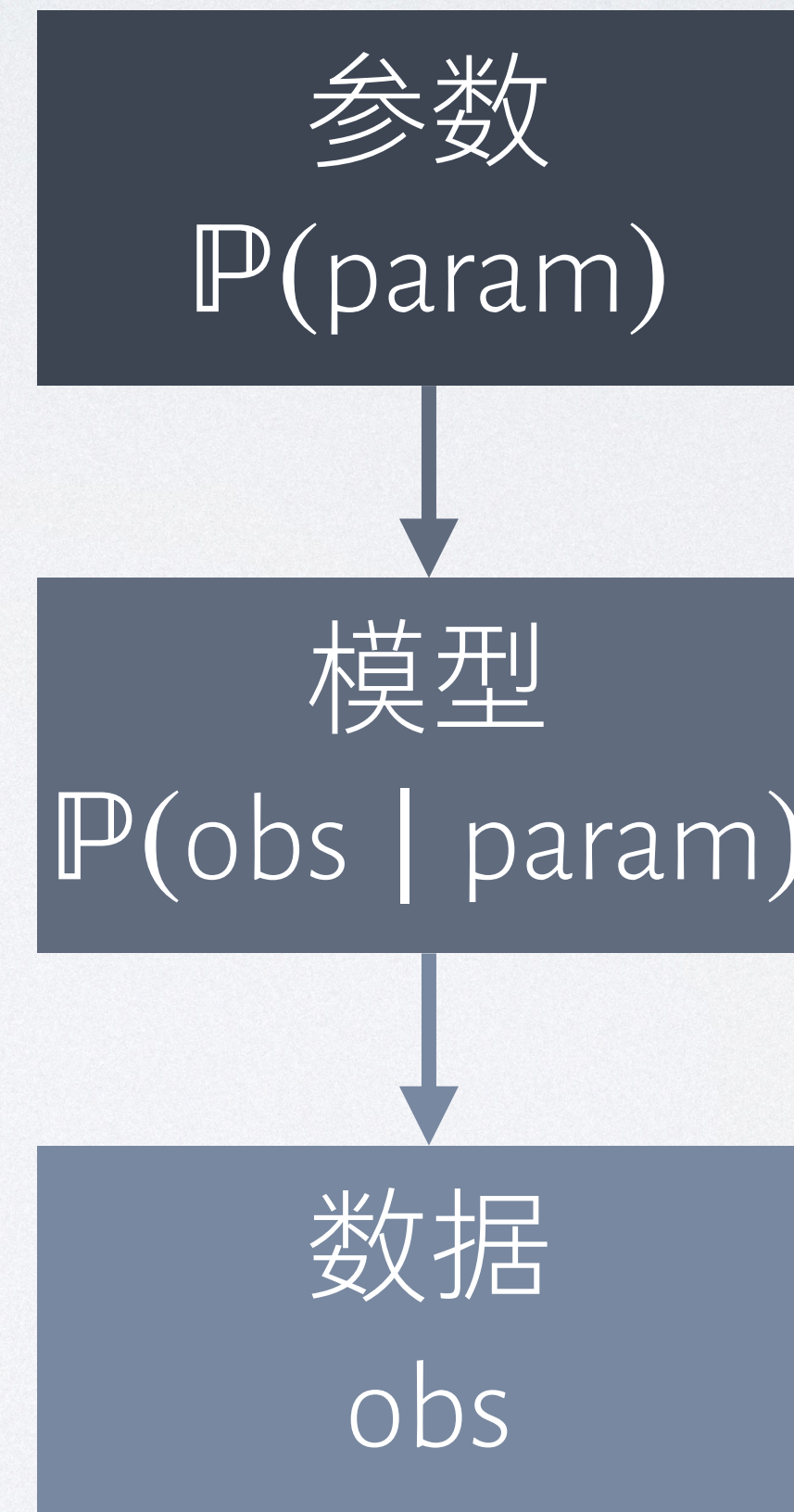
贝叶斯推断
 $\mathbb{P}(\text{param} | \text{obs})$

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)
```



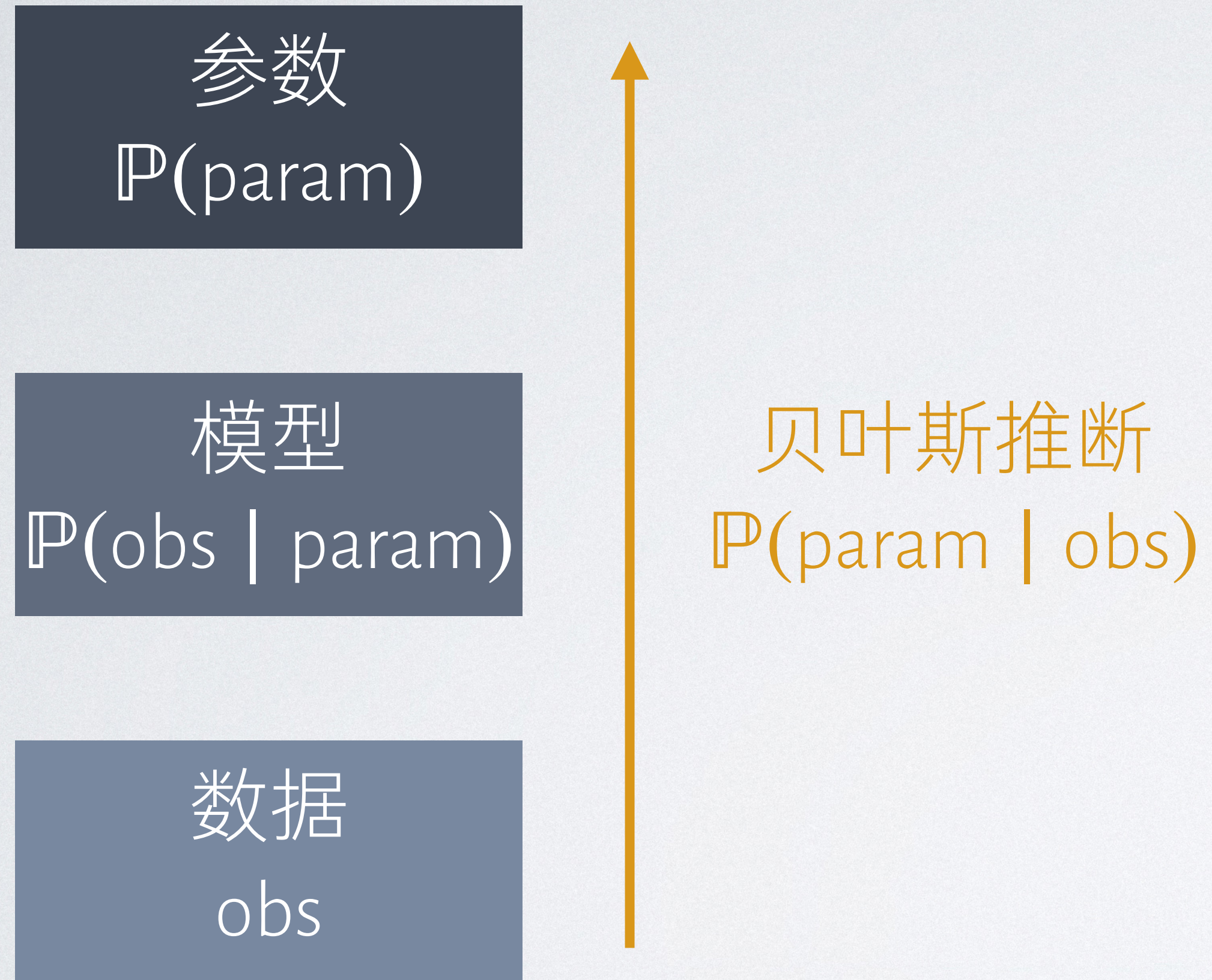
听众喜欢我在概率编程上的工作的后验概率是多少？

贝叶斯推断
 $\mathbb{P}(\text{param} | \text{obs})$

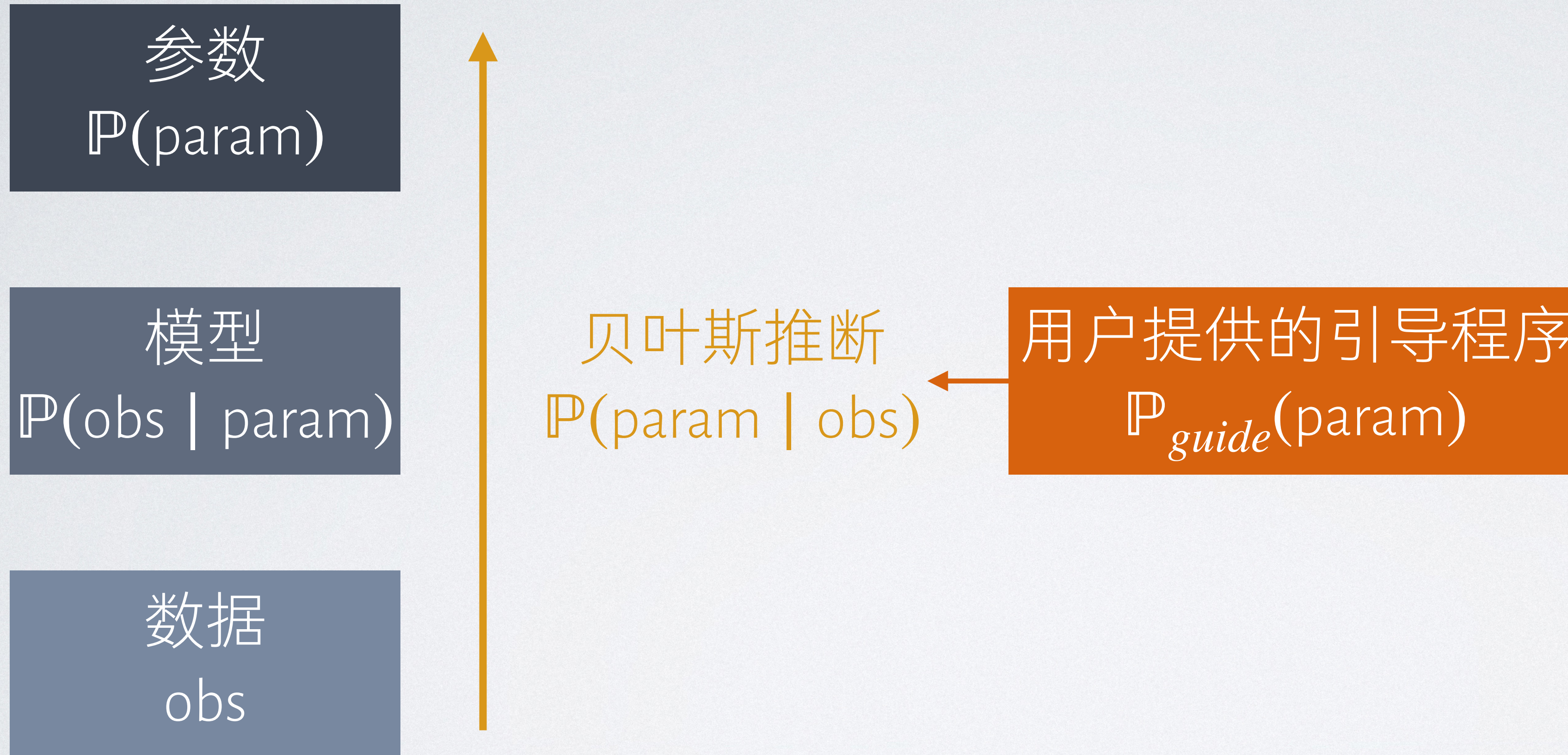
一个困难的问题



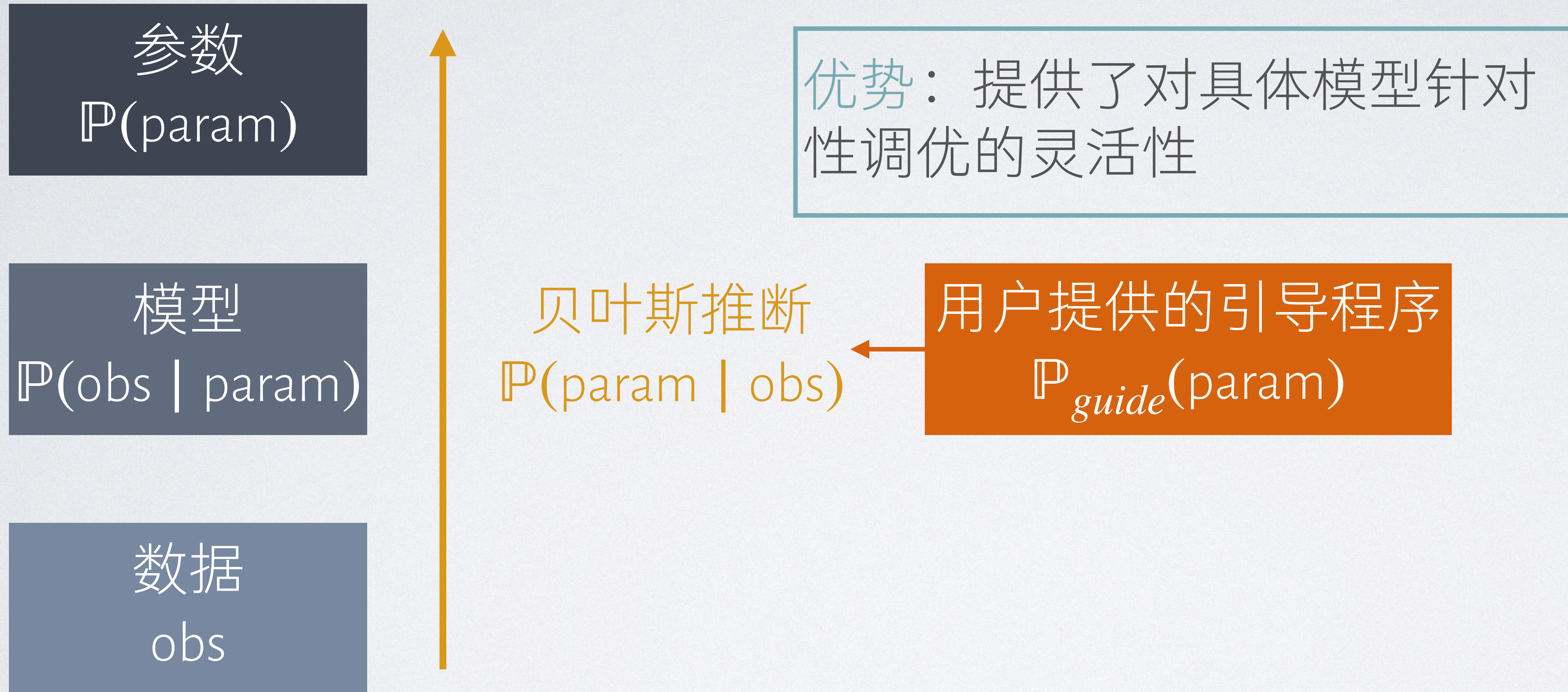
可编程概率推断



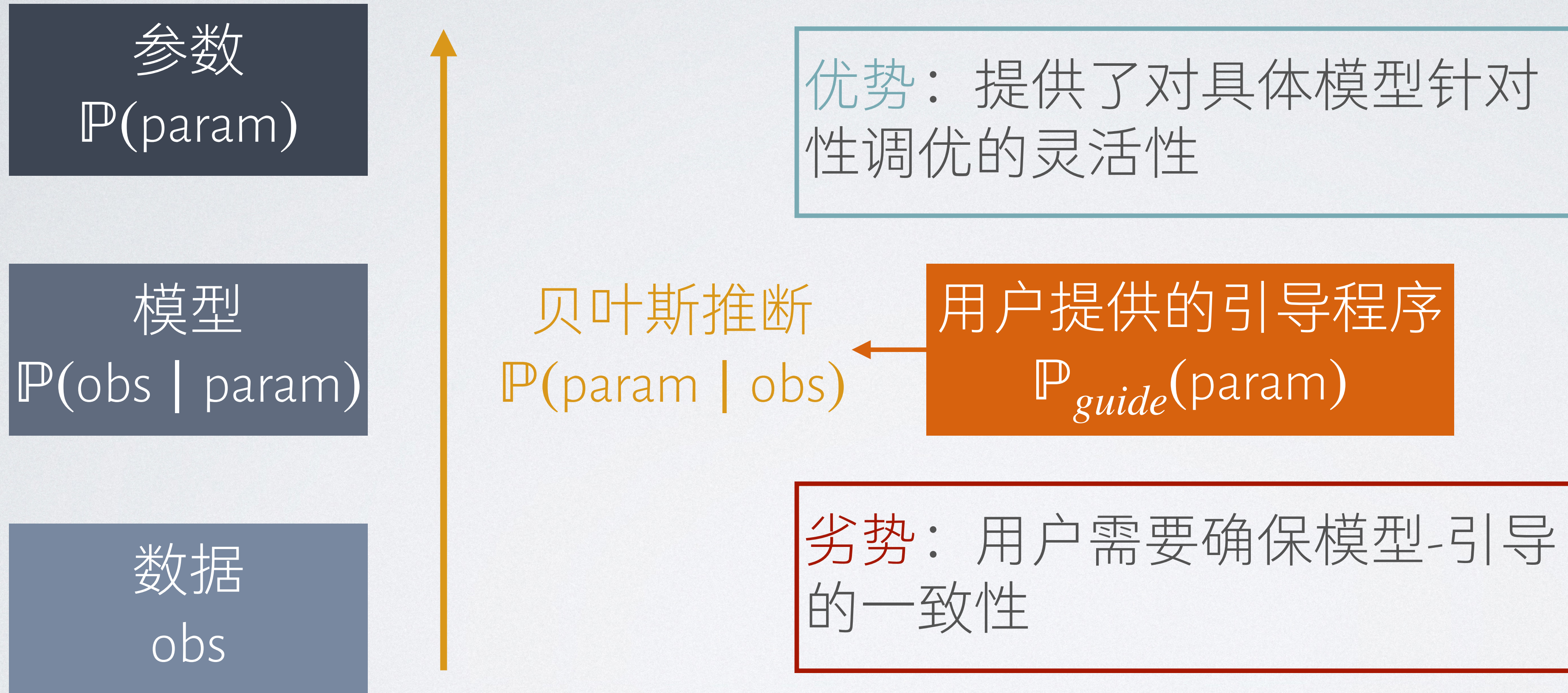
可编程概率推断



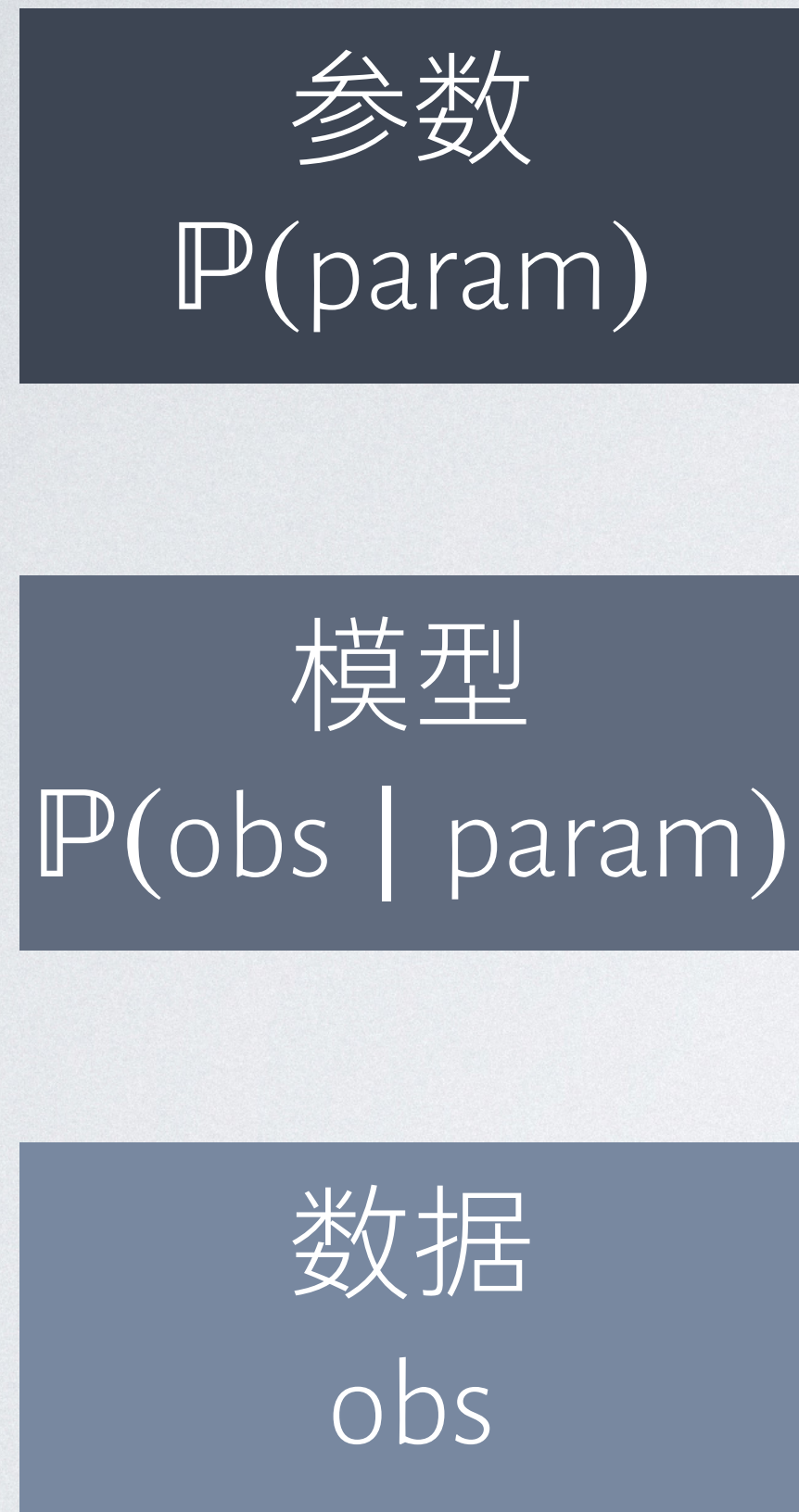
可编程概率推断



可编程概率推断



可编程概率推断



优势：提供了对具体模型针对性调优的灵活性

贝叶斯推断
 $\mathbb{P}(\text{param} \mid \text{obs})$

用户提供的引导程序
 $\mathbb{P}_{\text{guide}}(\text{param})$

劣势：用户需要确保模型-引导的一致性

一个复杂的任务



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. Programmable MCMC with Soundly Composed Guide Programs. *Working Paper*.

模型

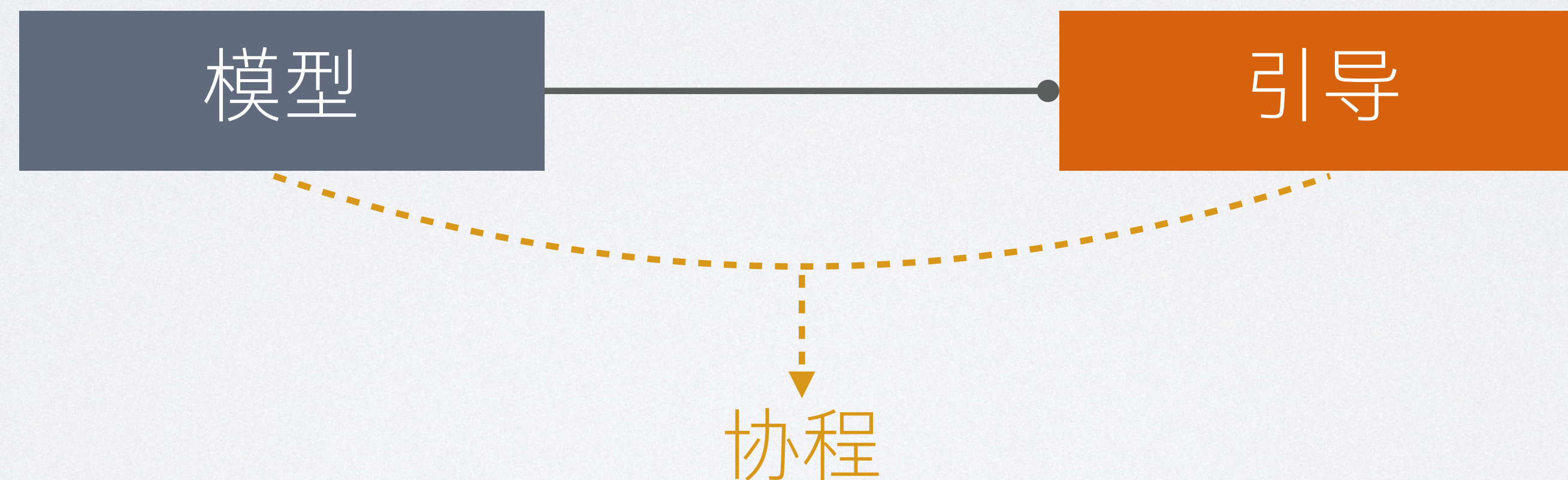
引导

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. Programmable MCMC with Soundly Composed Guide Programs. *Working Paper*.

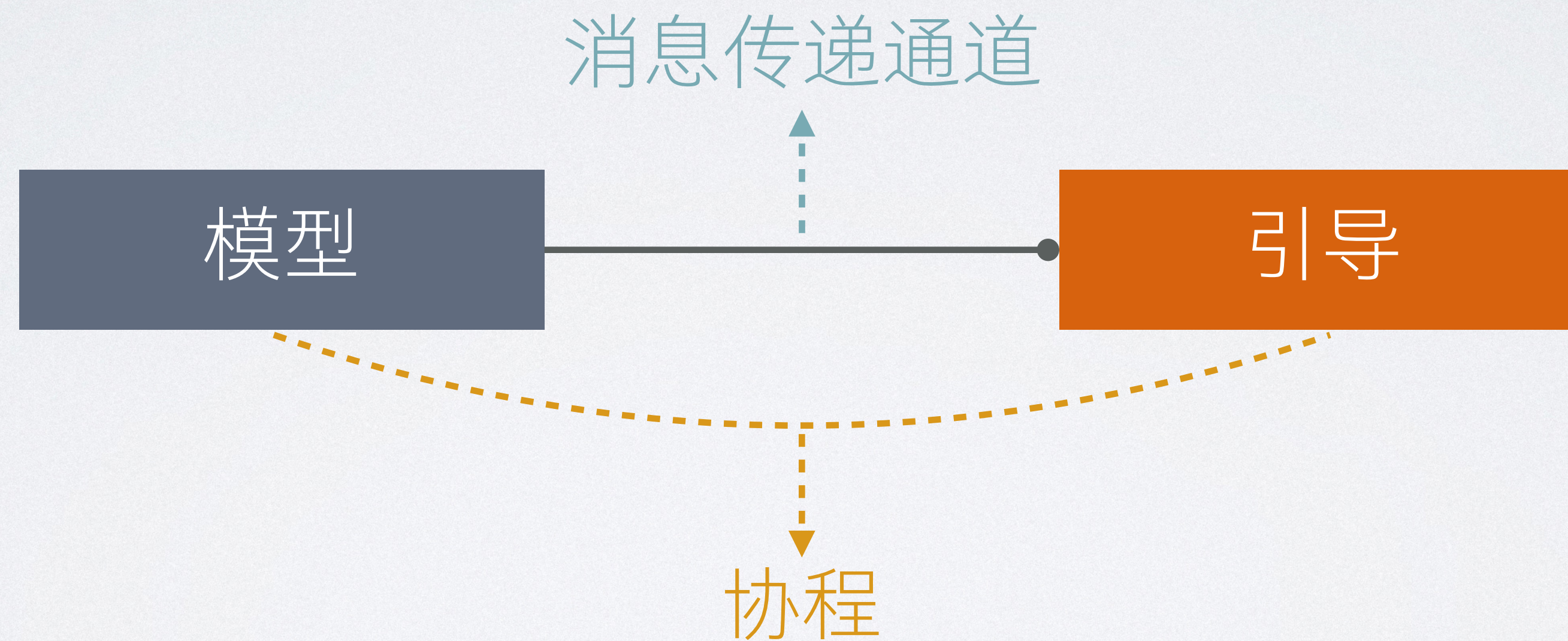


首个使用**并发编程**技术来确保可编程贝叶斯推断中的模型-引导一致性

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. Programmable MCMC with Soundly Composed Guide Programs. *Working Paper*.

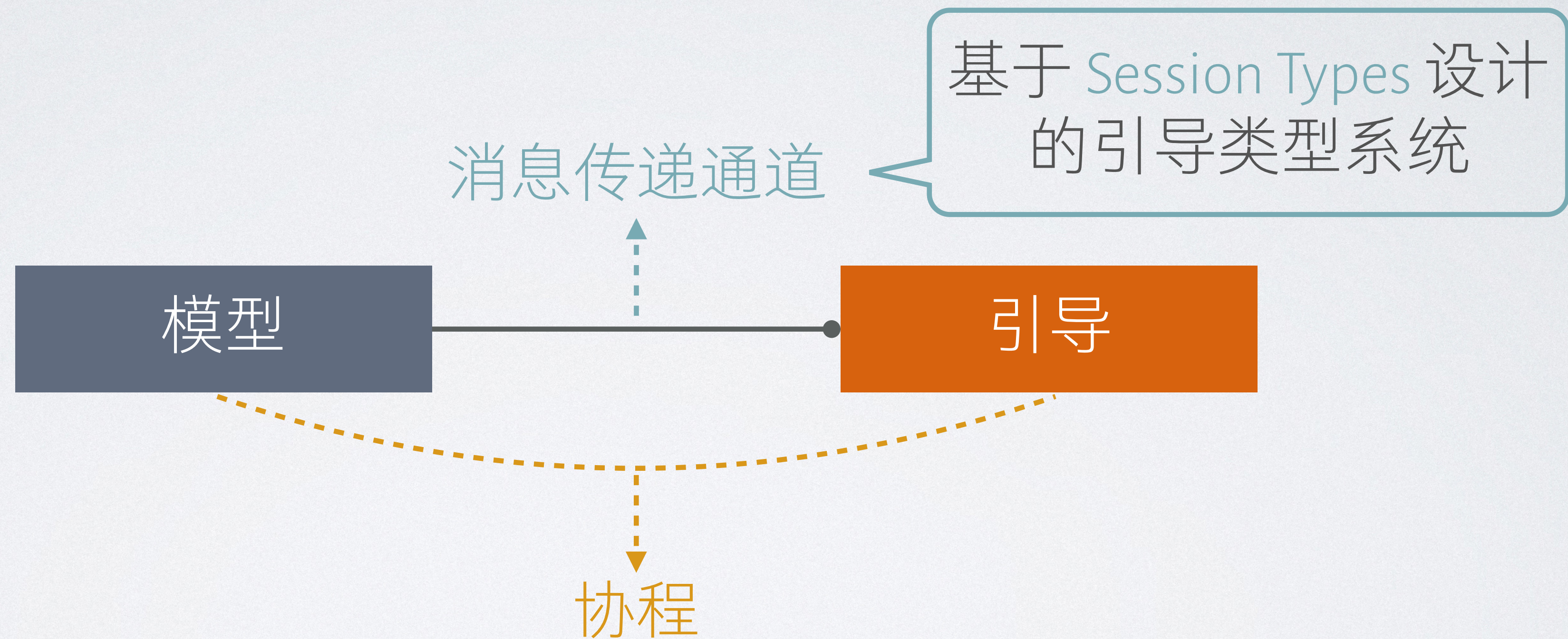


首个使用**并发编程**技术来确保可编程贝叶斯推断中的模型-引导一致性

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. Programmable MCMC with Soundly Composed Guide Programs. *Working Paper*.



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



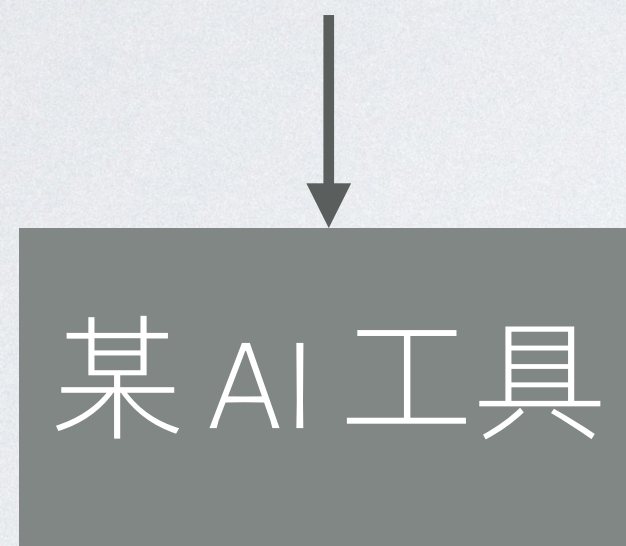
让 AI 来做引导?

- 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

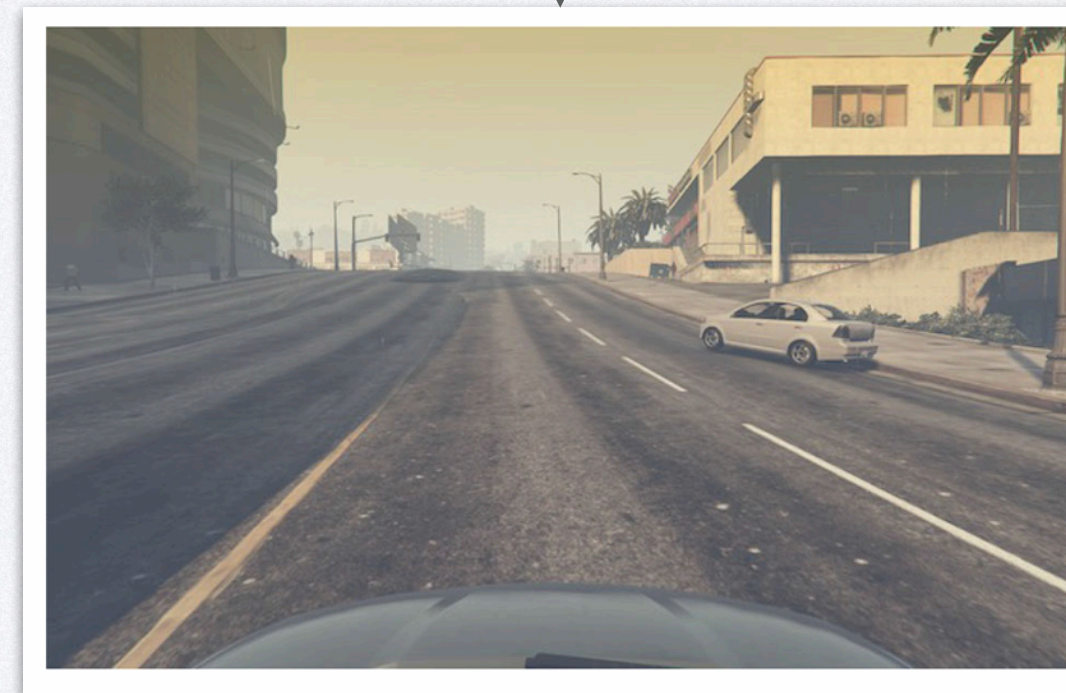
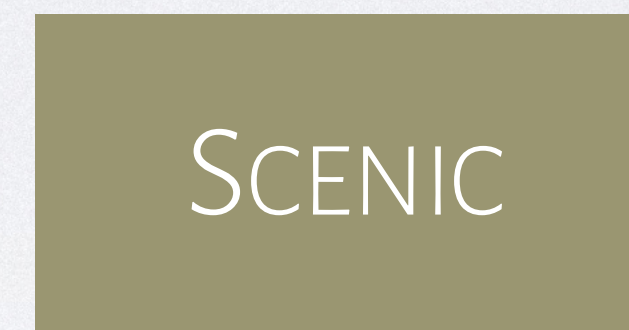
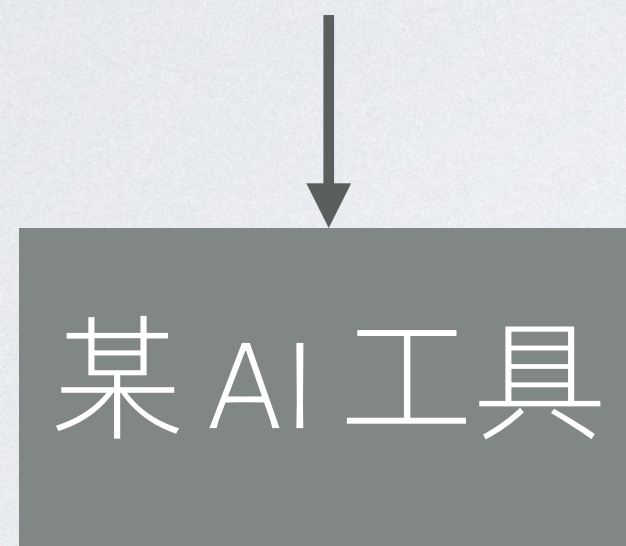
“A scene of a badly-parked car”



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?

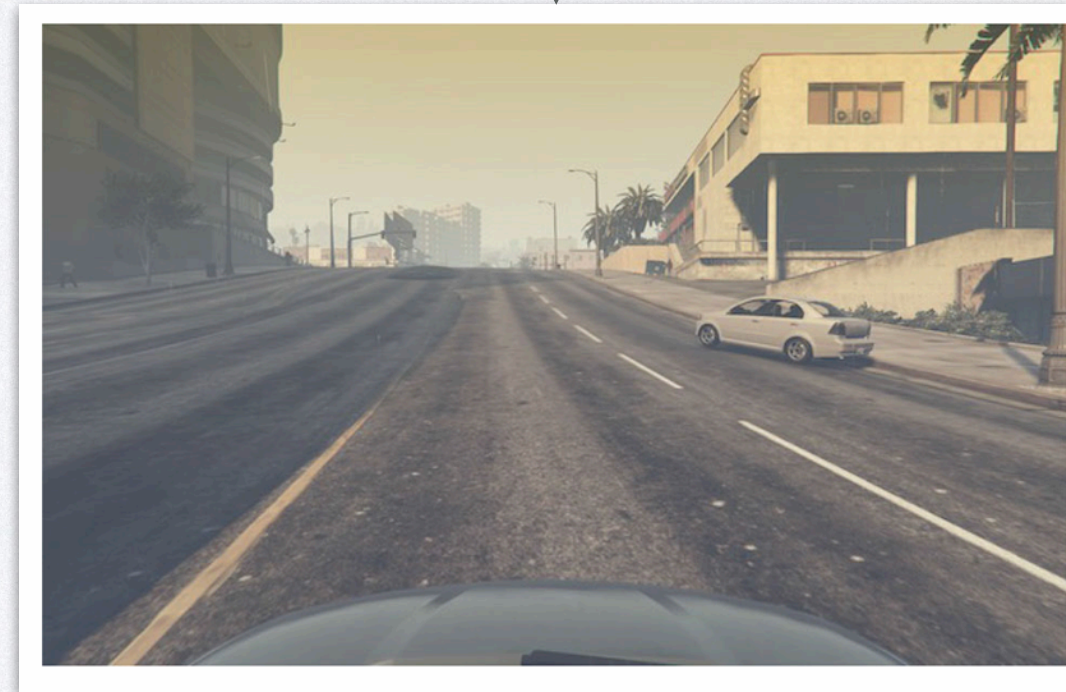
- 回顾「驾驶场景生成」问题：典型的 AI 生成内容（AIGC）的应用

“A scene of a badly-parked car”

某 AI 工具



SCENIC

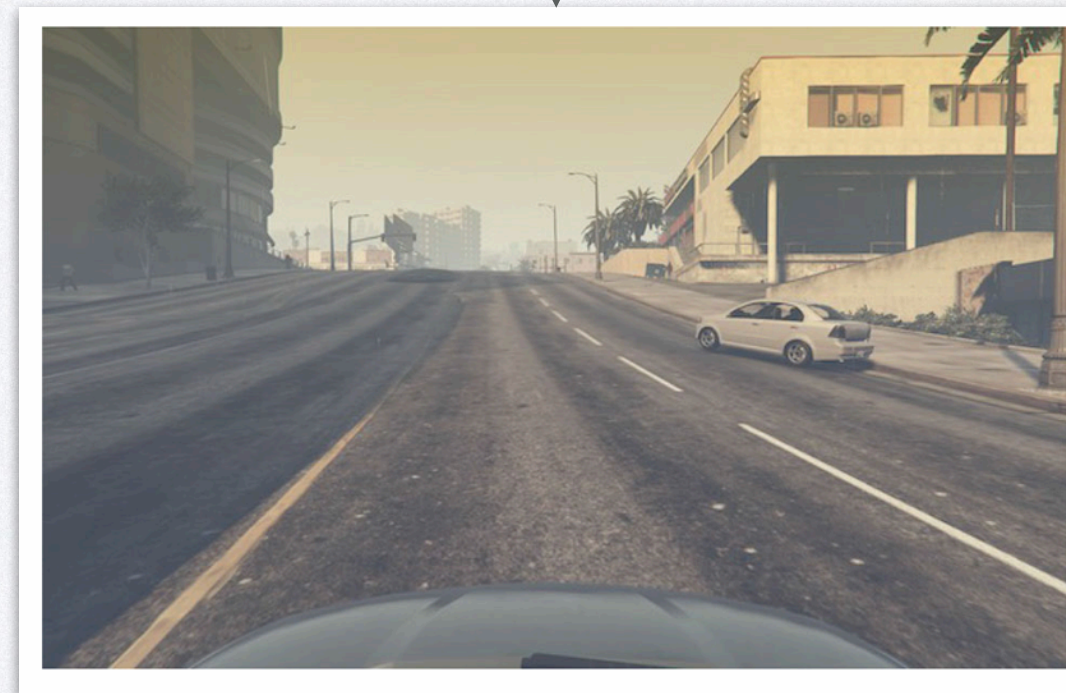
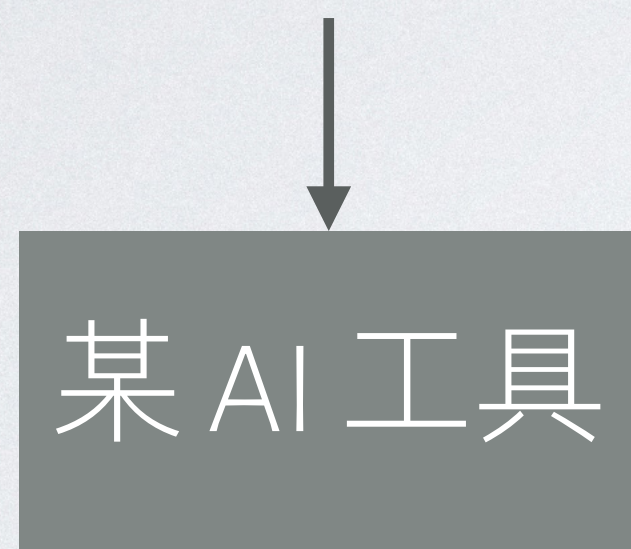


- AIGC 的优势：内容丰富、擅长处理模糊性

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

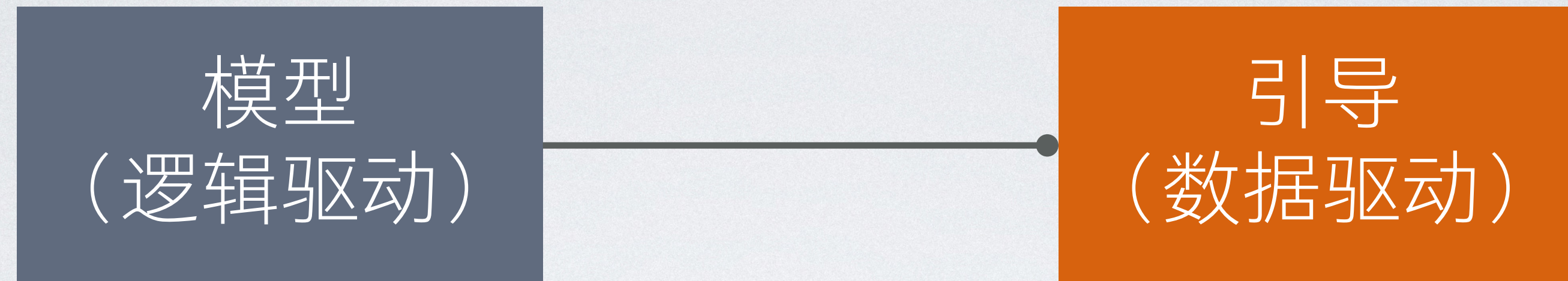
“A scene of a badly-parked car”



- 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



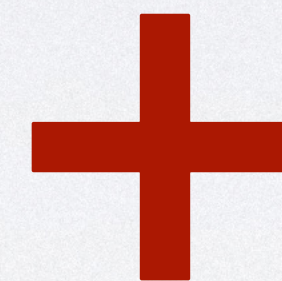
生成式「模糊」编程语言

模型
(逻辑驱动)

引导
(数据驱动)

智能系统（如语言模型）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”

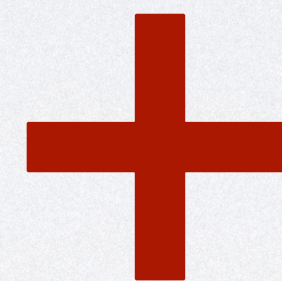
生成式「模糊」编程语言

模型
(逻辑驱动)

引导
(数据驱动)

智能系统（如语言模型）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：

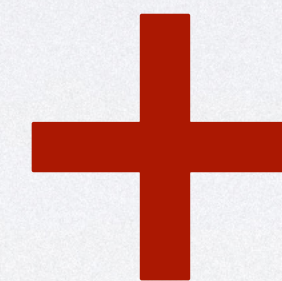
生成式「模糊」编程语言

模型
(逻辑驱动)

引导
(数据驱动)

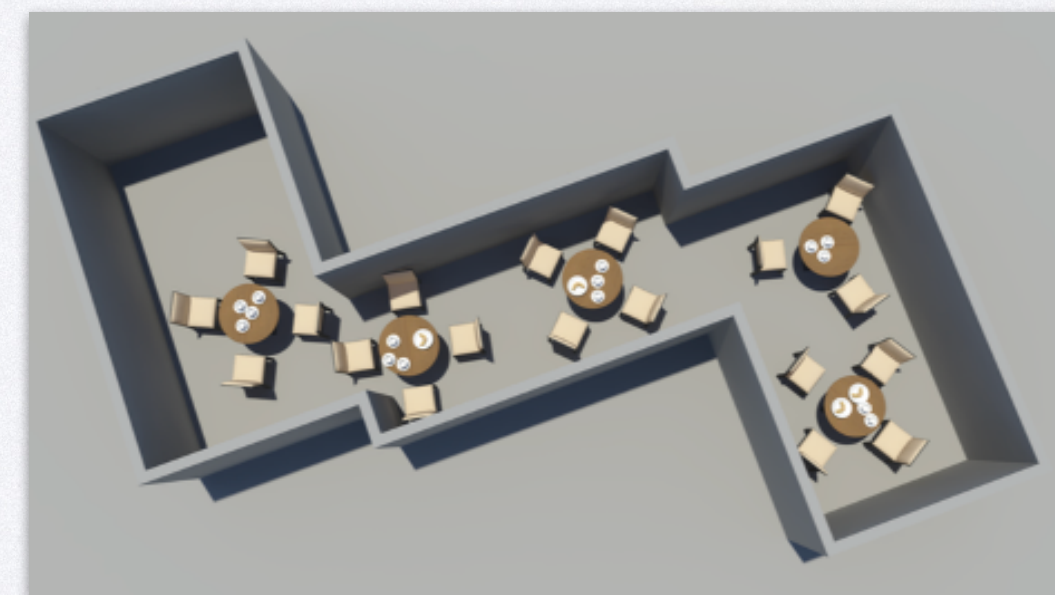
智能系统（如语言模型）提供采样能力

```
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（如语言模型）提供采样能力

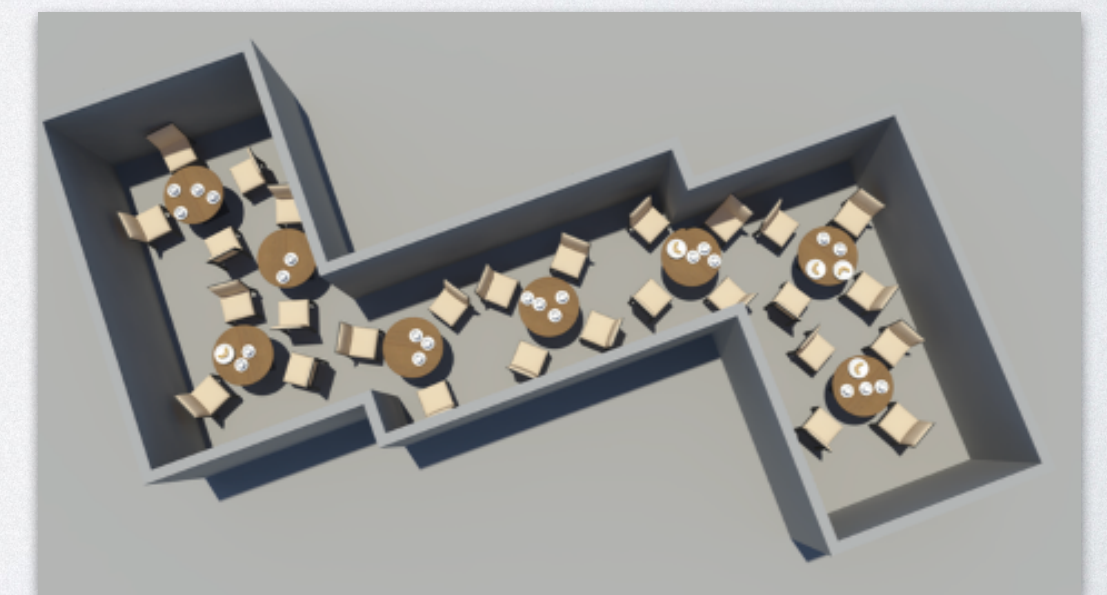
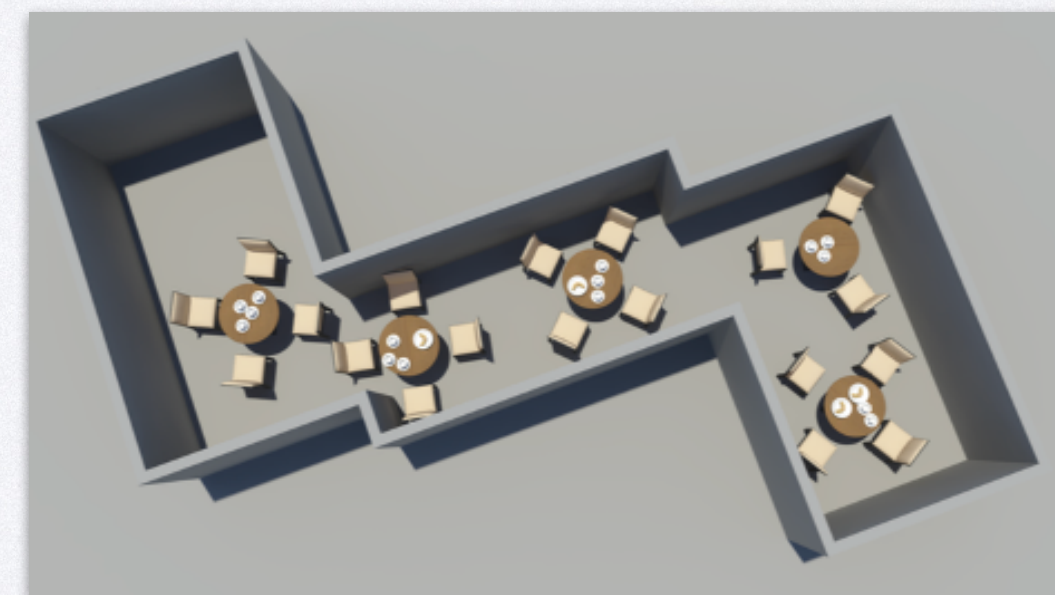
```
spot = OrientedPoint on visible curb  
badAngle = Uniform(1.0, -1.0) *  
            Range(10, 20) deg  
Car left of spot by 0.5,  
facing badAngle relative to roadDirection
```



“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



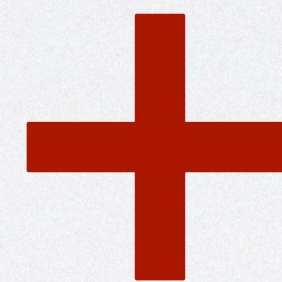
生成式「模糊」编程语言

模型
(逻辑驱动)

引导
(数据驱动)

智能系统（如语言模型）提供采样能力

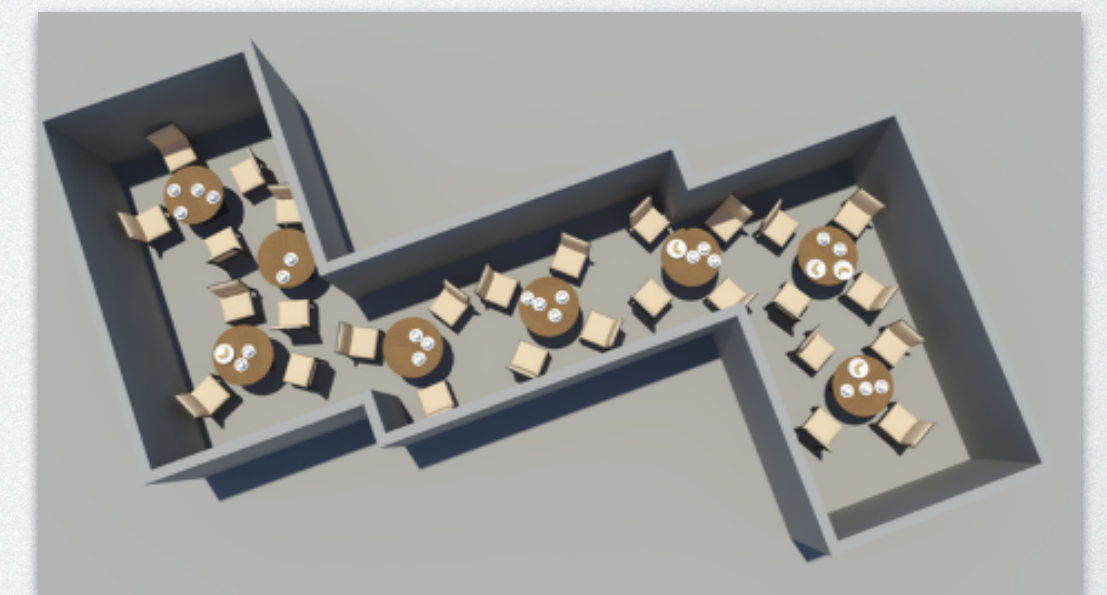
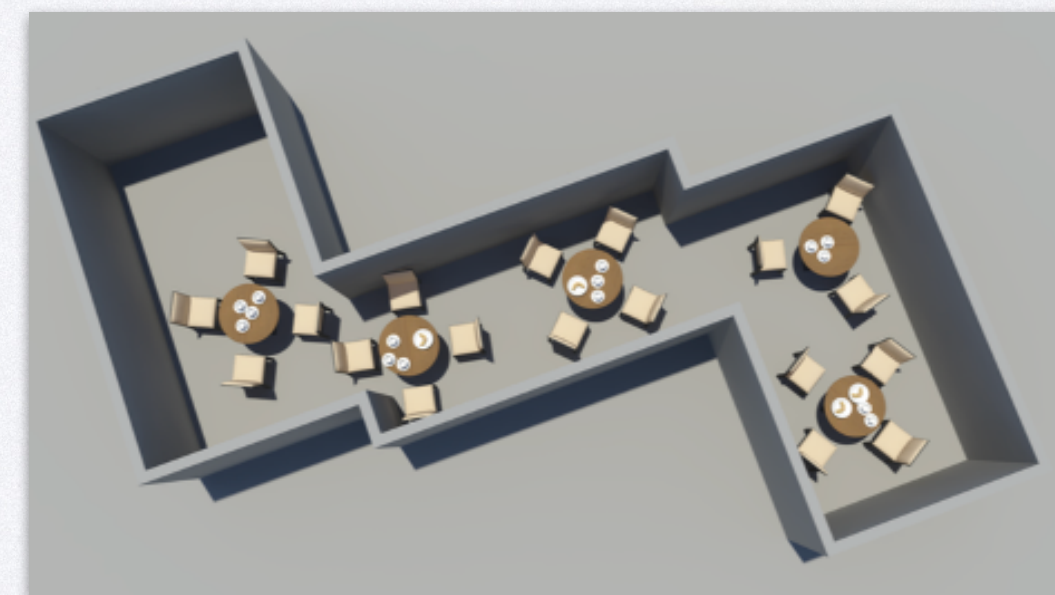
```
spot = OrientedPoint on visible curb  
badAngle = Uniform(1.0, -1.0) *  
            Range(10, 20) deg  
Car left of spot by 0.5,  
facing badAngle relative to roadDirection
```



“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





编程语言新范式初探

让程序更加安全：资源安全的编程语言

驾驭智能的计算：生成式模糊编程语言