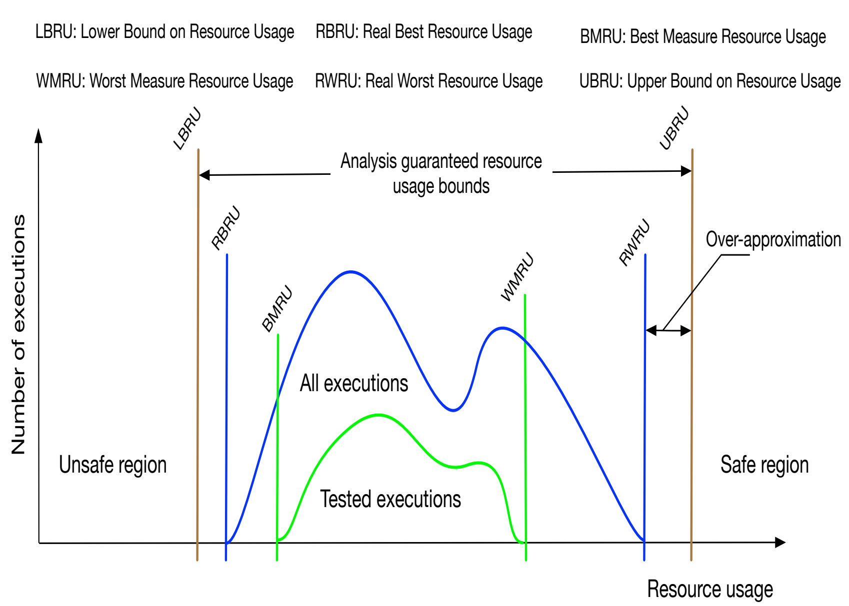
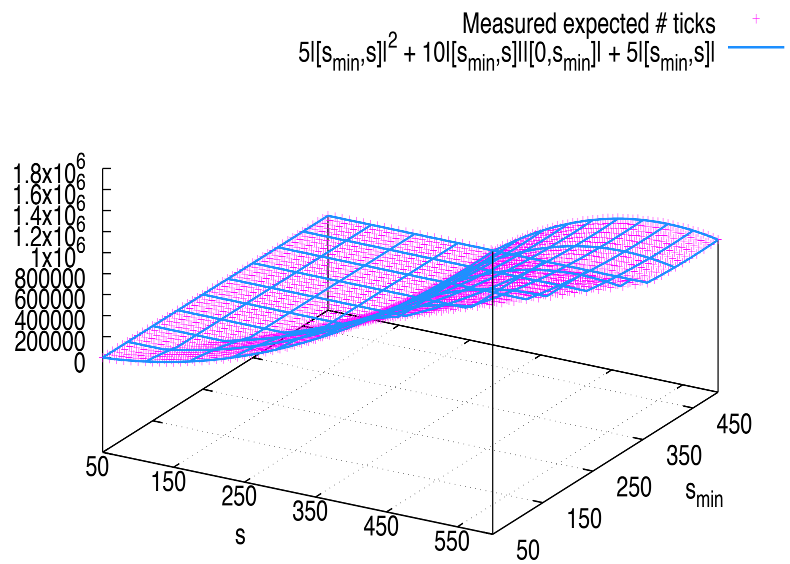
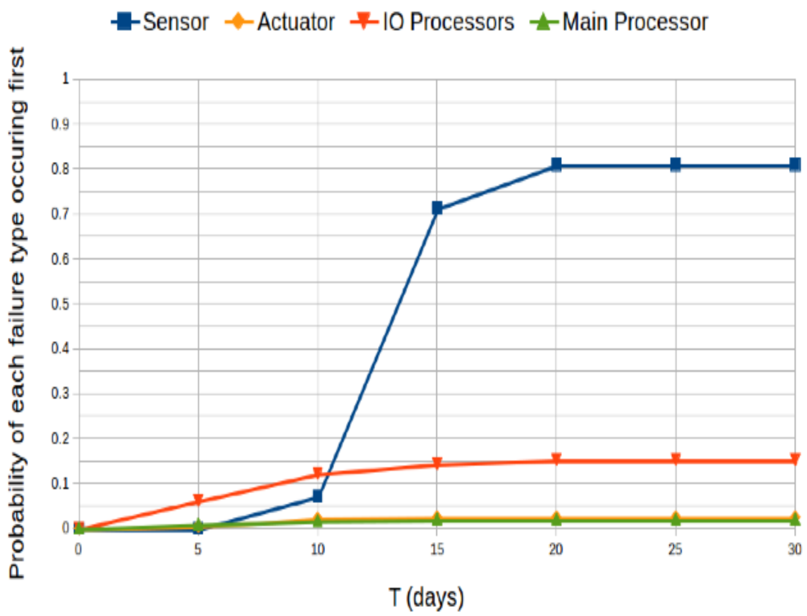
Quantitative Analysis for **Reliable** and **Secure Software**



WCET



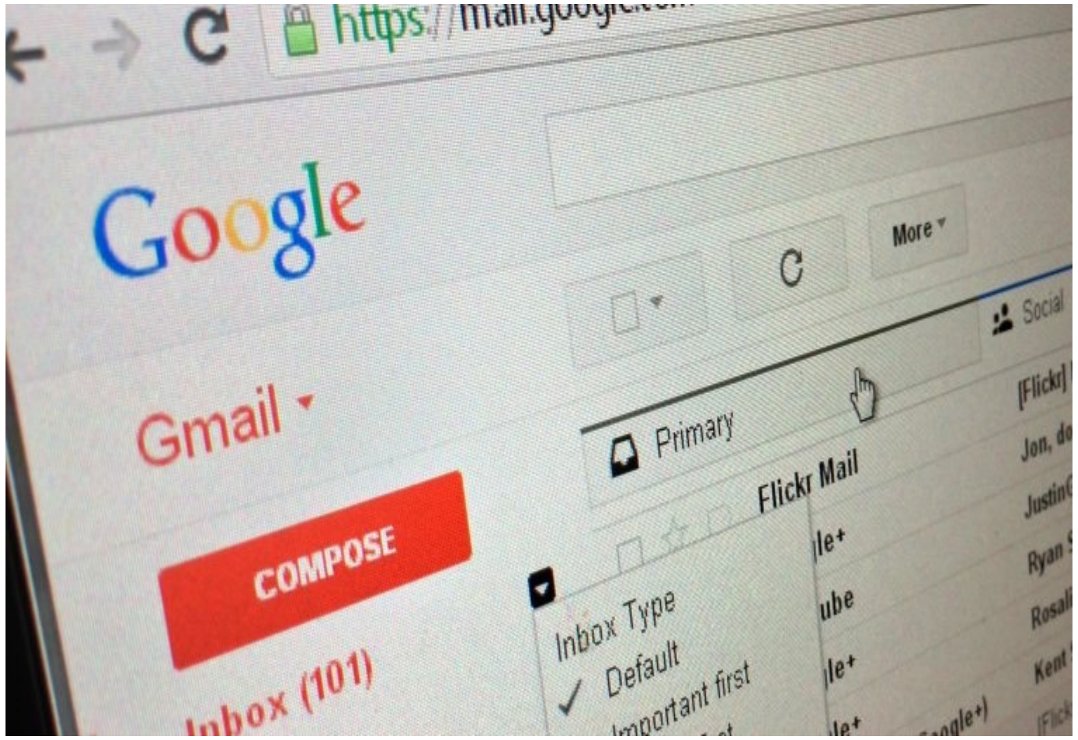
Expected WCET



Failure rates

Van Chan Ngo
channgo@cmu.edu

Software is everywhere and there is almost nothing that isn’t impacted by software


Google
Gmail
COMPOSE
Inbox (101)
More
Primary
Social
Flickr Mail
Inbox Type
Default
Important first


AIRBUS A380
A380

What if…

- Self-driving car crashes during maneuver
- Banking transactions are vulnerable to attacks
- Pacemaker fails due to running out of resources, e.g. memory, battery, …

What if…

- Self-driving car crashes during maneuver
- Banking transactions are vulnerable to attacks
- Pacemaker fails due to running out of resources, e.g. memory, battery, …

- Or Gmail is 2x slower
- According to Google, there are 1.2 billion users
- Each user demands 2.2kWh per year

What if…

- Self-driving car crashes during maneuver

- Banking transactions are vulnerable to attacks
- Pacemaker fails due to running out of resources, e.g. memory, battery

Generates 5 billion lbs of CO2 emissions more

- Or Gmail is 2x slo…
- According to Google, there are 1.2 billion users
- Each user demands 2.2kWh per year

Reliability & security flaws

Reliability & security flaws

2010: Recalls defective pacemaker devices because they cause health consequences or death, due to software defects

Reliability & security flaws

2010: Recalls defective pacemaker devices because they cause health consequences or death, due to software defects

2014: Toyota recalls 1.9 million cars due to software running out of stack space (stack overflow)

Reliability & security flaws

2010: Recalls defective pacemaker devices because they cause health consequences or death, due to software defects

2014: Toyota recalls 1.9 million cars due to software running out of stack space (stack overflow)

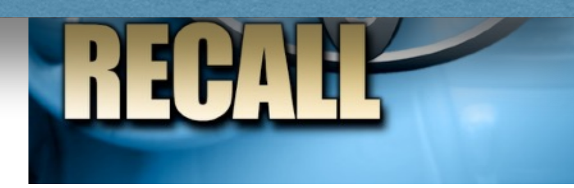
2015: Hackers remotely control a Jeep on the highway because of software security

Reliability & security flaws

2010: Recalls defective pacemaker devices because they cause health consequences or death, due to software defects

Our daily life and safety increasingly depend on software operating in a reliable and secure manner

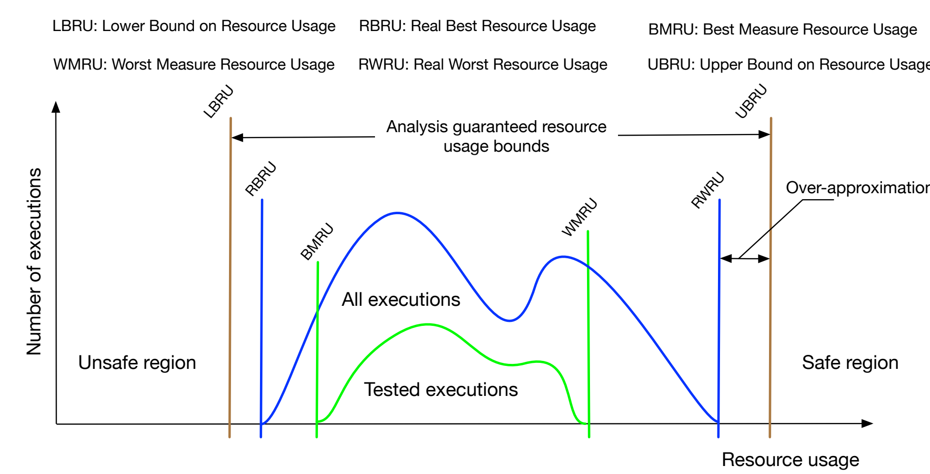
overflow)

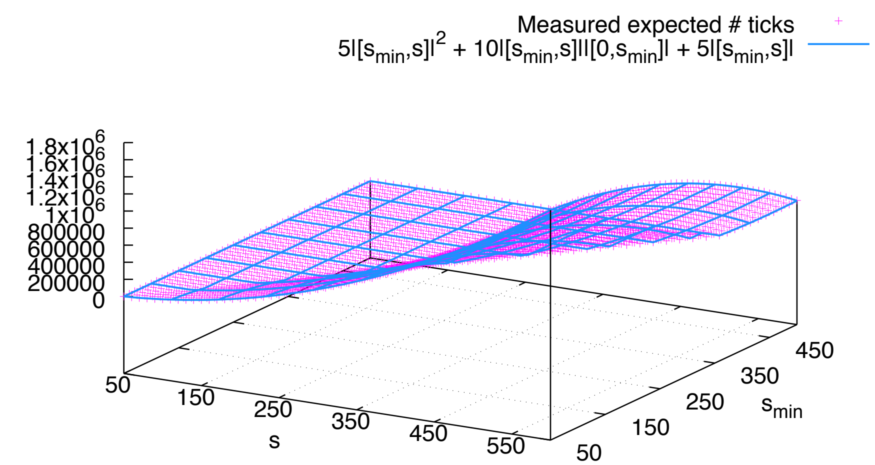


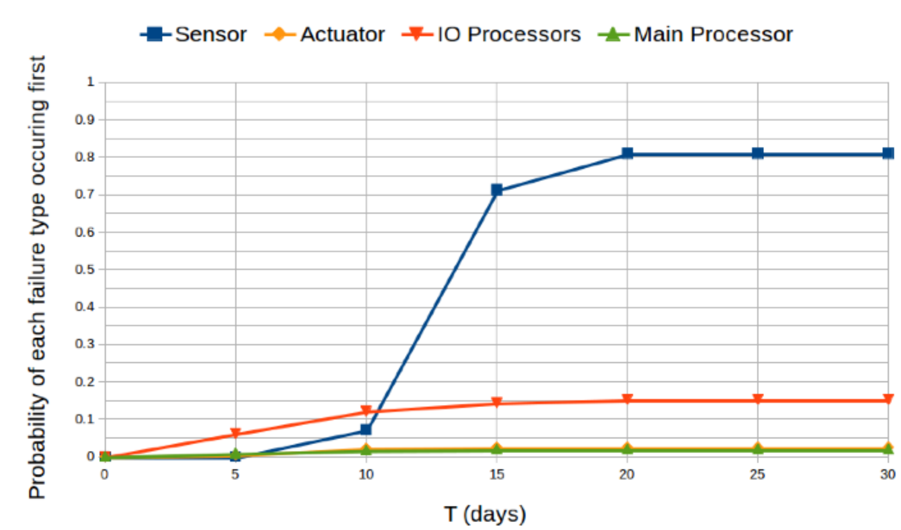
2015: Hackers remotely control a Jeep on the highway because of software security

Quantitative analysis

- Mathematically reasons about quantitative aspects of software
- Real-time properties: hard/soft deadlines
- Resource constraints: energy, memory allocation, buffer size
- Probabilistic aspects: random delays, failure rates, expected resource usage
- Security & privacy: leakage of secret data







Quantitative analysis, con't

- Is the worst-case execution time less than 10 ms?
- Is the program secure against timing side-channel attacks?
- Is the probability that a self-driving car makes a fatality caused by accident per hour less than 10^{-9}?
- What is the worst-case expected time taken for the algorithm to terminate?
- What is the probability of a failure causing the system to shut down within 4 hours?

Quantitative analysis, con't

Quantitative properties are correctness

- Is the worst-case execution time less than 10 ms?
- Is the program secure against timing side-channel attacks?
- Is the probability that a self-driving car makes a fatality caused by accident per hour less than 10^{-9}?
- What is the worst-case expected time taken for the algorithm to terminate?
- What is the probability of a failure causing the system to shut down within 4 hours?

Quantitative analysis, con't

Quantitative properties are correctness

- Is the worst-case execution time less than 10 ms?
- Is the program secure against timing side-channel attacks?
- Is the probability that a self-driving car makes a fatality caused by accident per hour less than 10^{-9}?
- What is the worst-case expected time taken for the algorithm to terminate?
- What is the probability of a failure causing the system to shut down within 4 hours?

Quantitative properties are probabilistic and performance

Testing and simulation

- Testing and simulation is not enough for quantitative analysis
- Probability of a fatality caused by accident per hour of human driving is less than 10^{-6}
- Fatality rate of autonomous vehicles should be less than 10^{-9}
- Testing needs at least 10^{9} hours of driving to collect data
- Order of 30 billion miles of data

Testing and simulation

- Testing and simulation is not enough for quantitative analysis
- Probability of a fatality caused by accident per hour of human driving is less than 10^{-6}
- Fatality rate of autonomous vehicles should be less than 10^{-9}
- Testing needs at least 10^{9} hours of driving to collect data
- Order of 30 billion miles of data

Almost impossible in practice

Goals and contributions

- Static and automatic quantitative analysis at
  - programing language-level with type system, program logic, …
  - formal model-level with model checking, theorem proving, …
- My contributions
  - Formally verified compiler for designing safety-critical systems
  - Statistical model checking for timed and probabilistic SystemC
  - Static analysis for probabilistic programming
  - Type system for preventing side-channel attacks
- Published at top conferences in formal verification, programming language and compiler, and security including CAV, PLDI, Oakland

Goals and contributions

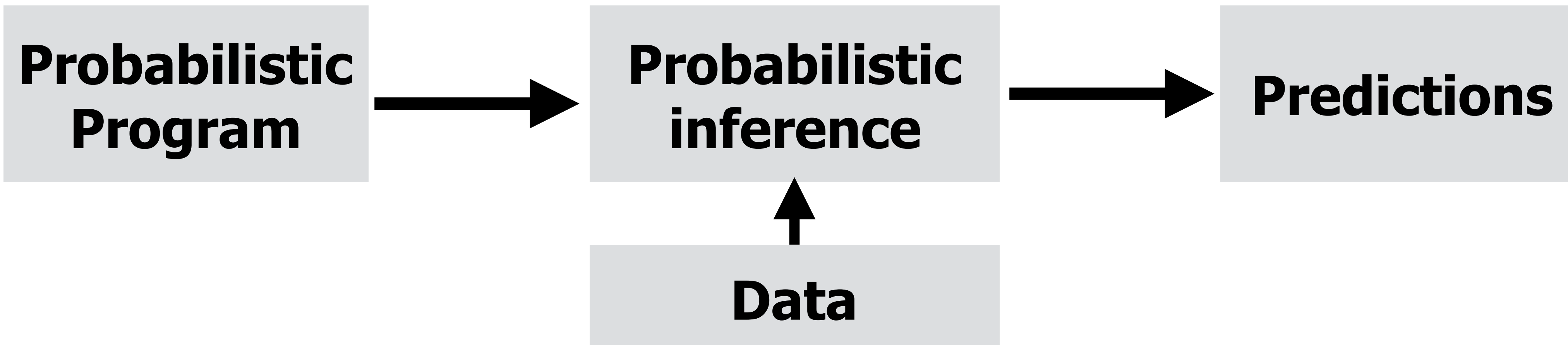
- Static and automatic quantitative analysis at
  - programing language-level with type system, program logic, …
  - formal model-level with model checking, theorem proving, …
- My contributions
  - Formally verified compiler for designi
  - Statistical model checking for timed a
  - Static analysis for probabilistic programming
  - Type system for preventing side-channel attacks
- Published at top conferences in formal verification, programming language and compiler, and security including CAV, PLDI, Oakland

Focus of this talk

Outline

- **Why quantitative analysis?**
- **Static analysis for probabilistic programming [PLDI ’18]**
- **Type system for enforcing security**
- **Future research directions**

Probabilistic programming



- Standard programming language like C or ML with two additional constructs
- Sampling assignment "x = Dist" draws a sample from a distribution
- Probabilistic branching "c1 [1/2] c2" controls flow by observations
- Goal is probabilistic inference that computes distribution implicitly specified by a probabilistic program
- Desired output can be expected values or probabilities of variables

Applications

- Machine learning
  - Specify prior distributions as probabilistic programs and rely on a compiler to perform inference and make predictions
- Security
  - Cryptography e.g., probabilistic encryptions as randomize algorithms
- Modeling probabilistic systems
  - System performance and reliability e.g., failure rates, reliability of communication channels

Example

```
tick = 0
while (x < n) {
   x = x + 1
   [3/4]
   x = x - 1;
   tick = tick + 1;
}
```

Simulates a random walk that ends when walker reaches the boundary

Each time unit:

- Goes forward 1 step with p = 3/4
- Goes backward 1 step with = 1/4

What is the expected value of # ticks?

Example

Current position

```
tick = 0
while (x < n) {
   x = x + 1
   [3/4]
   x = x - 1;
   tick = tick + 1;
}
```

Simulates a random walk that ends when walker reaches the boundary

Each time unit:

- Goes forward 1 step with p = 3/4
- Goes backward 1 step with = 1/4

What is the expected value of # ticks?

Example

Current position

Boundary condition

```
tick = 0
while (x < n) {
   x = x + 1
   [3/4]
   x = x - 1;
   tick = tick + 1;
}
```

Simulates a random walk that ends when walker reaches the boundary

Each time unit:

- Goes forward 1 step with p = 3/4
- Goes backward 1 step with = 1/4

What is the expected value of # ticks?

Example

Current position

Boundary condition

```
tick = 0
while (x < n) {
   x = x + 1
   [3/4]
   x = x - 1;
   tick = tick + 1;
}
```

Each iterator costs 1 time unit

Simulates a random walk that ends when walker reaches the boundary

Each time unit:

- Goes forward 1 step with p = 3/4
- Goes backward 1 step with = 1/4

What is the expected value of # ticks?

Example

Current position

Boundary condition

```
tick = 0
while (x < n) {
   x = x + 1
   [3/4]
   x = x - 1;
   tick = tick + 1;
}
```

Each iterator costs 1 time unit

Simulates a random walk that ends when walker reaches the boundary

Each time unit:

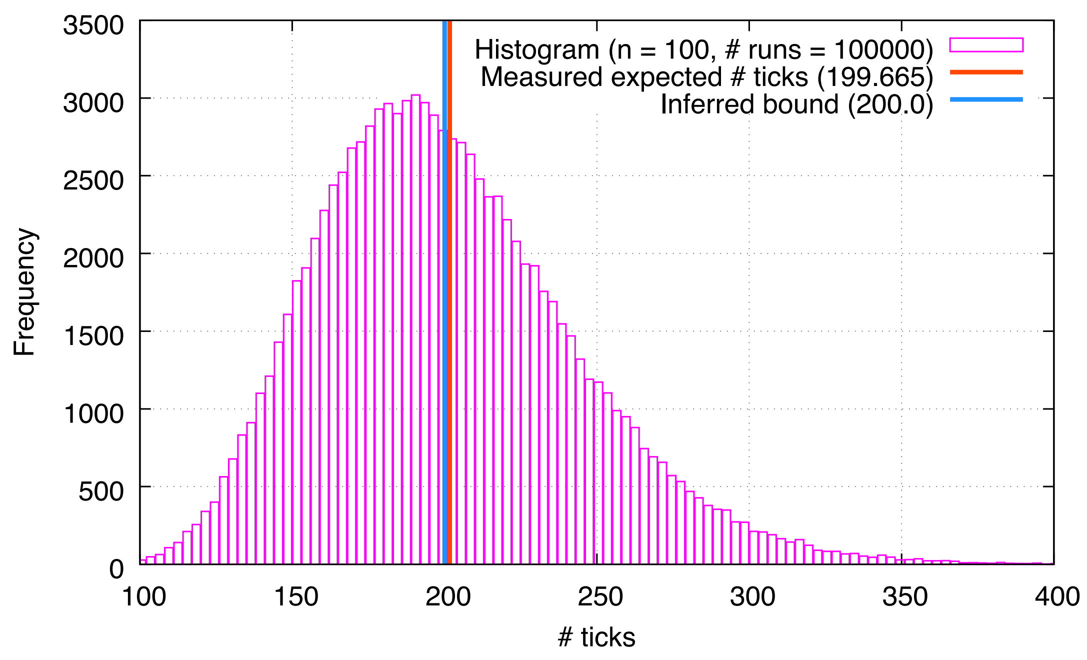
- Goes forward 1 step with p = 3/4

A goal of the probabilistic compilation

What is the expected value of # ticks?

Simulation-based technique

- Current probabilistic compilers use simulation to estimate the expected value
- With x = 0, n = 100, the estimated value is 199.665 with # runs = 100000
- However, simulation-based techniques have many drawbacks
  - Only give a concrete value
  - Run program many times again for other parameter values
  - Require an efficient compiled code (e.g., parallel executable code)



Expected bound analysis

- A static analysis that infers automatically upper-bounds on the expected resource consumption
- Bounds are multivariate symbolic polynomial and tight with precise constant factors
- A practical implementation working on imperative probabilistic programs
- For example, our tool gives the linear bound 2max(0,n-x) on the expected # ticks

Derivation

Cost model

```
tick = 0
while (x < n) {
  x = x + 1
  [3/4]
  x = x - 1;
  tick = tick + 1;
}
```

Static analyzer

Error

Bound

Derivation

Cost model

Specifies resource usage of primitive constructs

```
tick = 0
while (x < n) {
   x = x + 1
   [3/4]
   x = x - 1;
   tick = tick + 1;
}
```

Static analyzer

Error

Bound

Derivation

Cost model

Specifies resource usage of primitive constructs

```
tick = 0
while (x < n) {
   x = x + 1
   [3/4]
   x = x - 1;
   tick = tick + 1;
}
```

Static analyzer

Error

Bound

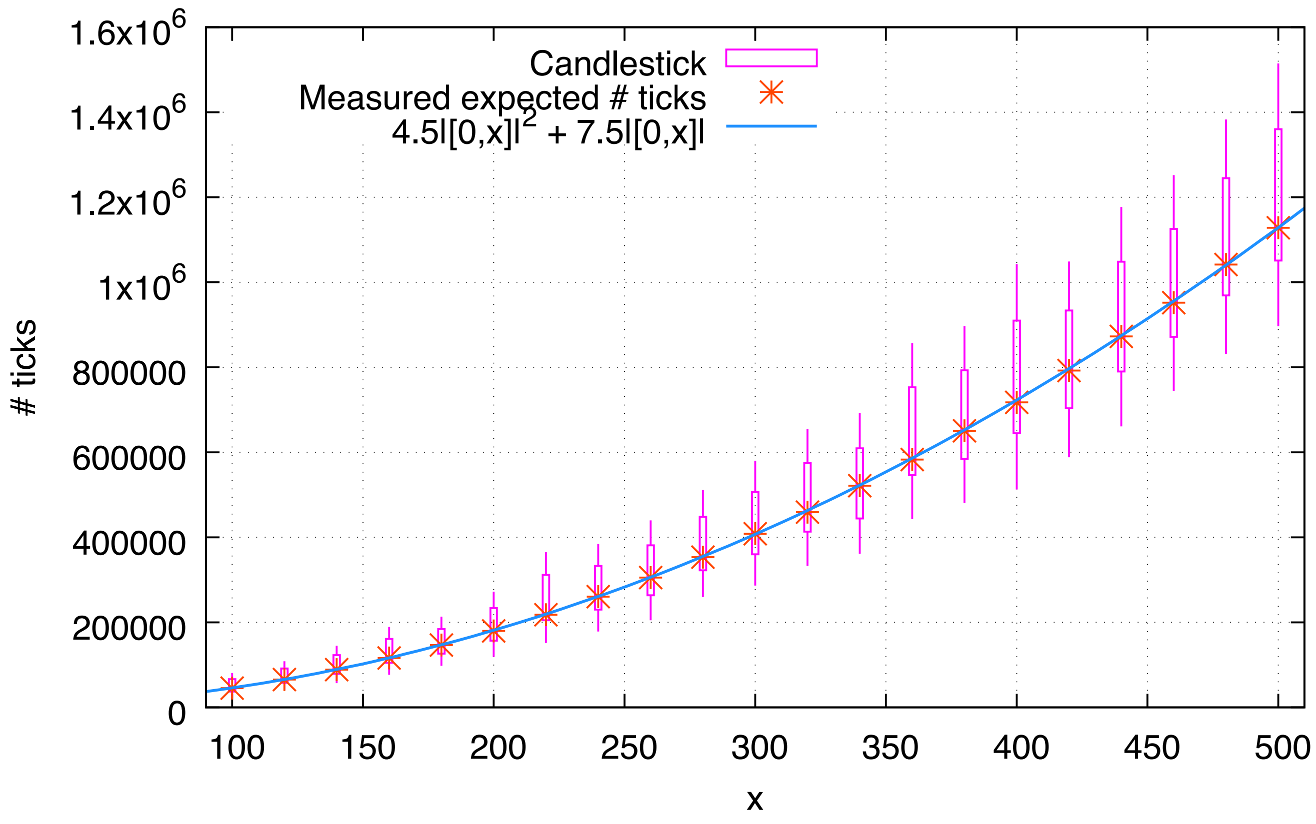
Expected resource usage might be infinite

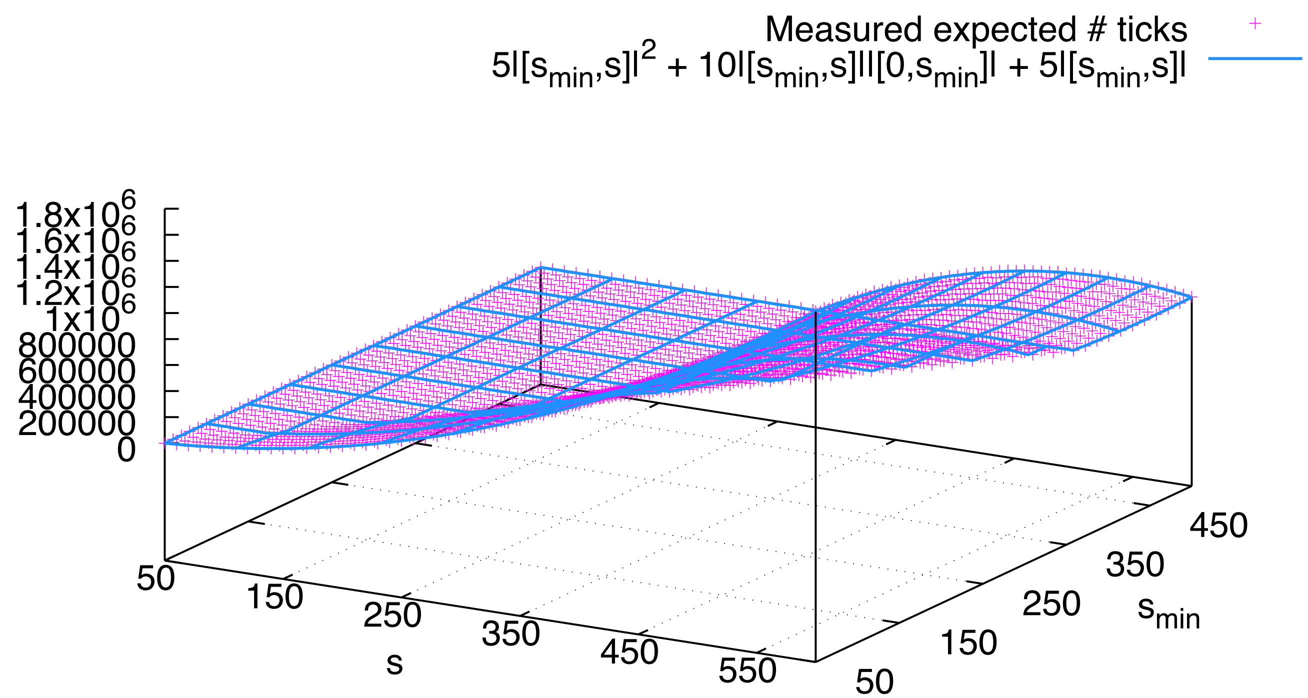
Absynth - Automatic bound synthesizer

- Accepts imperative probabilistic programs
- Infers multivariate polynomial bounds on the expected resource consumption
- Automatically analyzes 40 challenging probabilistic programs and randomized algorithms with different looping patterns
- Statically derived bounds are compared with simulation-based expectations to show that constant factors are very precise

Precise constant factors

- Percentage errors between statically derived bounds and simulation-based values are less than 1% for almost programs
- For example, figures show the constant factors in derived bounds for polynomial programs are very precise

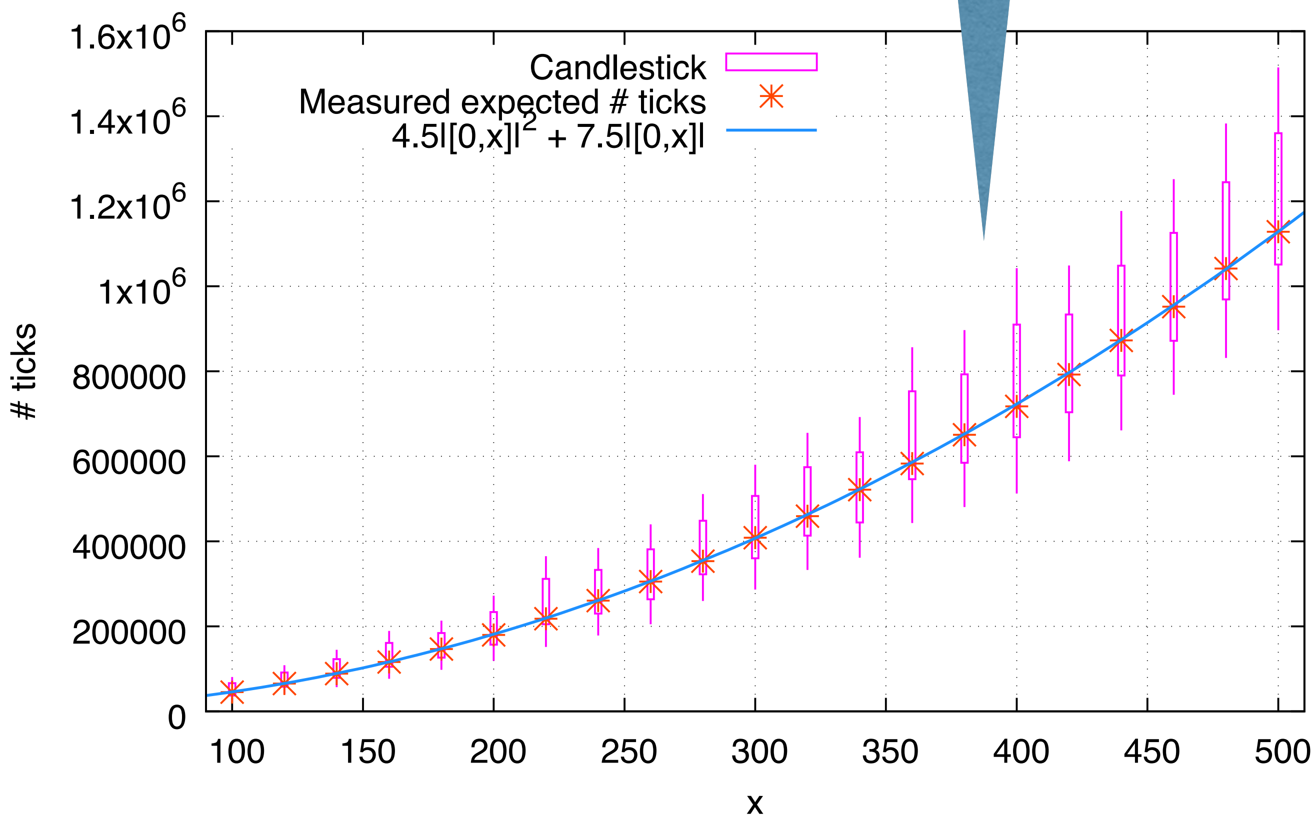


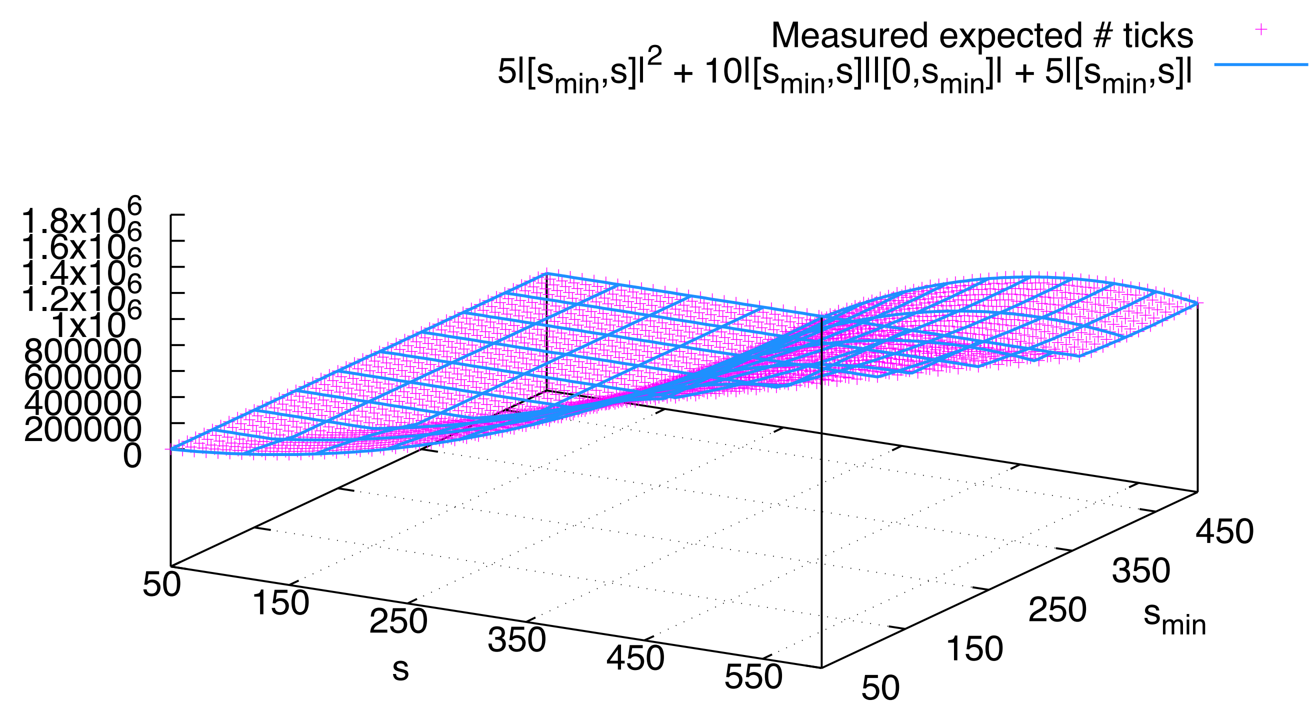


Precise constant factors

- Percentage errors between statically derived bounds and simulation-based values are less than 1% for almost programs
- For example, figures show the constant factors in derived bounds for polynomial programs are very precise

Blue lines are plotting of derived bounds



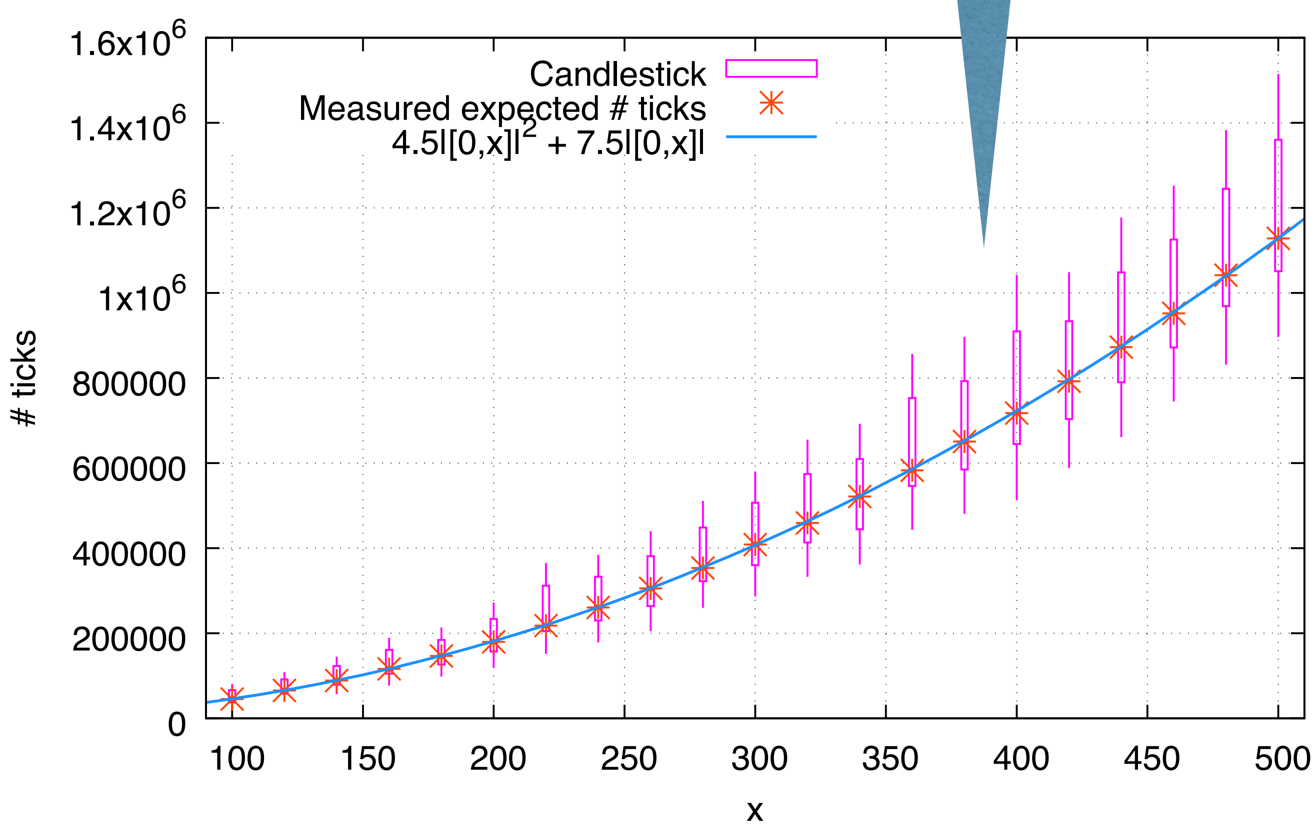


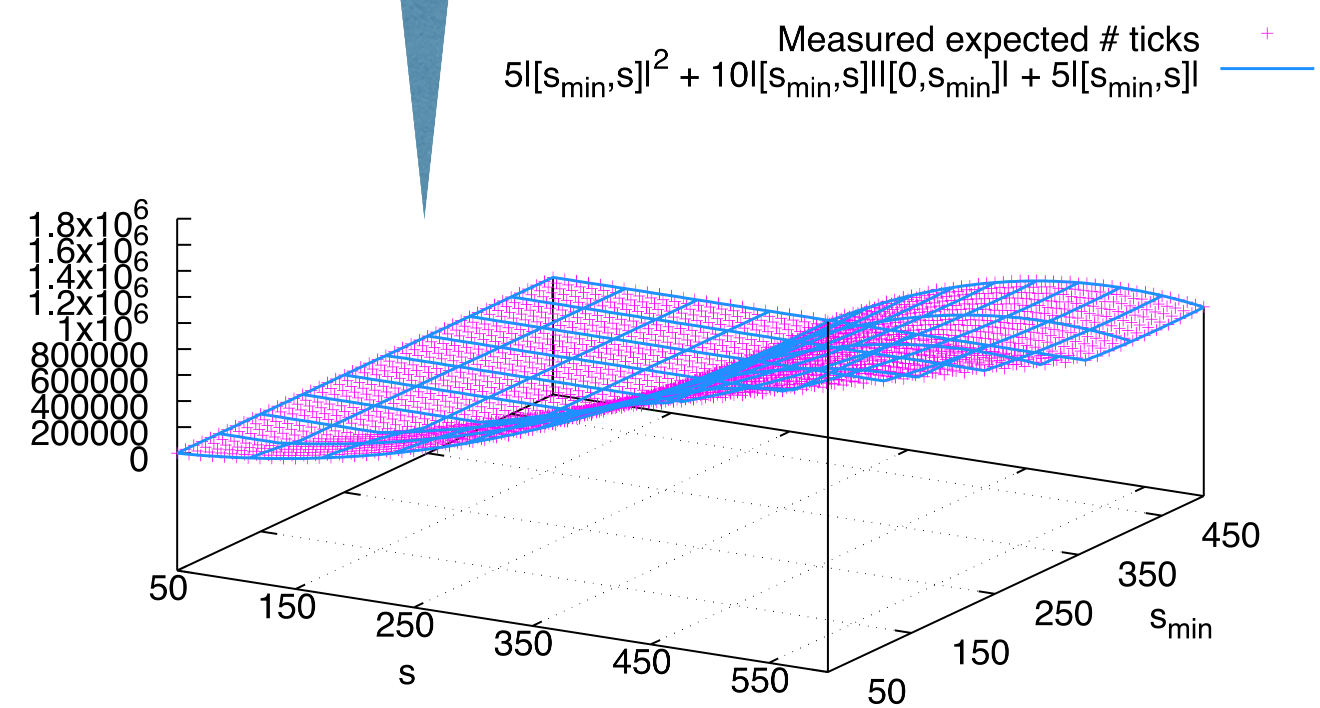
Precise constant factors

- Percentage errors between statically derived bounds and simulation-based values are less than 1% for almost programs
- For example, figures show the constant factors in derived bounds for polynomial programs are very precise

Blue lines are plotting of derived bounds

Red points are simulation-based values





Expected potential method

- Associate potential functions to program points
  - Function from states to non-negative values
- Potential pays the expected resource consumption and the expected potential at the following point
- The initial potential is an upper bound on the expected resource usage

Expected potential method

Valuation of program variables

$$\Phi(state) \geq 0$$

- Associate potential functions to program points
  - Function from states to non-negative values
- Potential pays the expected resource consumption and the expected potential at the following point
- The initial potential is an upper bound on the expected resource usage

Expected potential method

- Associate potential functions to program points
  - Function from states to non-negative values
- Potential pays the expected resource consumption and the expected potential at the following point
- The initial potential is an upper bound on the expected resource usage

Valuation of program variables

$$\Phi(state) \geq 0$$

$$\Phi(state) \geq \mathbb{E}(\text{cost}) + \mathbb{E}(\Phi'(next\_state))$$

Expected potential method

- Associate potential functions to program points
  - Function from states to non-negative values
- Potential pays the expected resource consumption and the expected potential at the following point
- The initial potential is an upper bound on the expected resource usage

Valuation of program variables

$$\Phi(state) \geq 0$$

$$\Phi(state) \geq \mathbb{E}(\text{cost}) + \mathbb{E}(\Phi'(next\_state))$$

Total expectation and linearity

$$\Phi(init\_state) \geq \mathbb{E}(\Sigma\text{cost})$$

Automation

- Fix potential functions as linear combinations of monomials with unknown coefficients

$$\Phi(\sigma) = \Sigma_i k_i \cdot m_i$$

- Encode the potential relations as linear constraints

- Solve the constraints with an off-the-shelf LP solver

$$\Phi(state) \geq 0$$

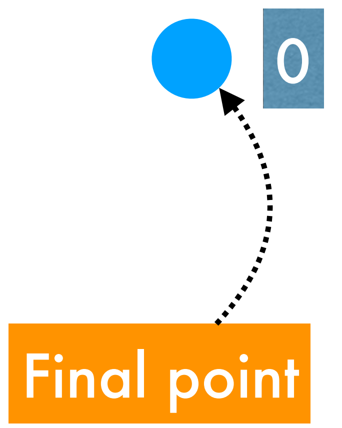
$$\Phi(state) \geq \mathbb{E}(\text{cost}) + \mathbb{E}(\Phi'(next\_state))$$

Total expectation and linearity

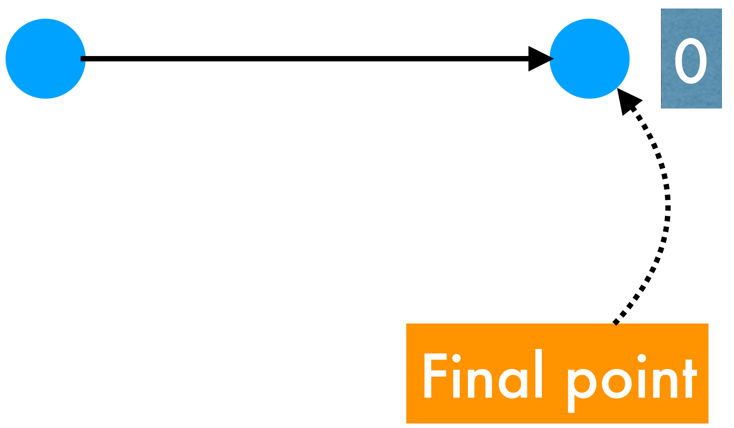
$$\Phi(init\_state) \geq \mathbb{E}(\Sigma\text{cost})$$

Example analysis on random walk

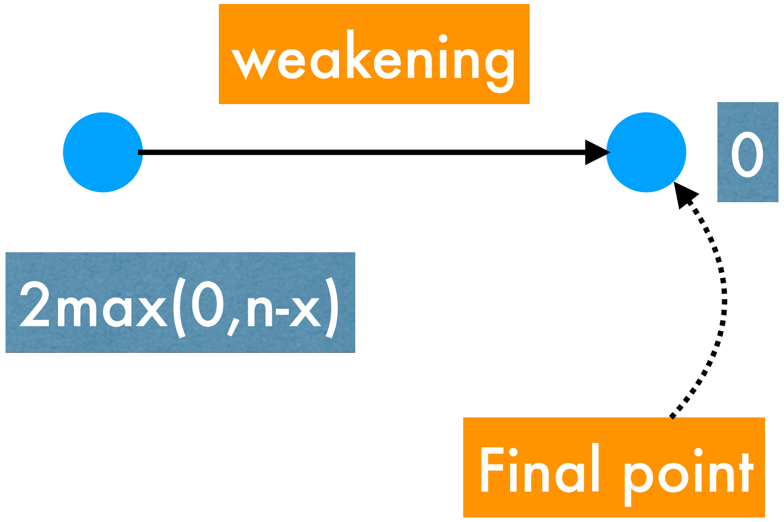
Example analysis on random walk



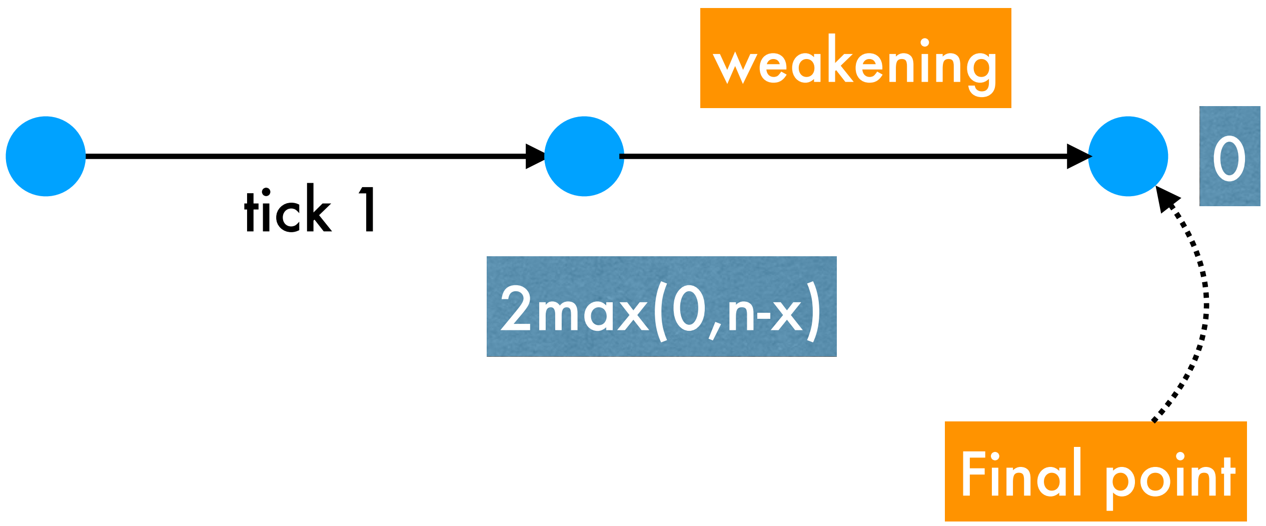
Example analysis on random walk



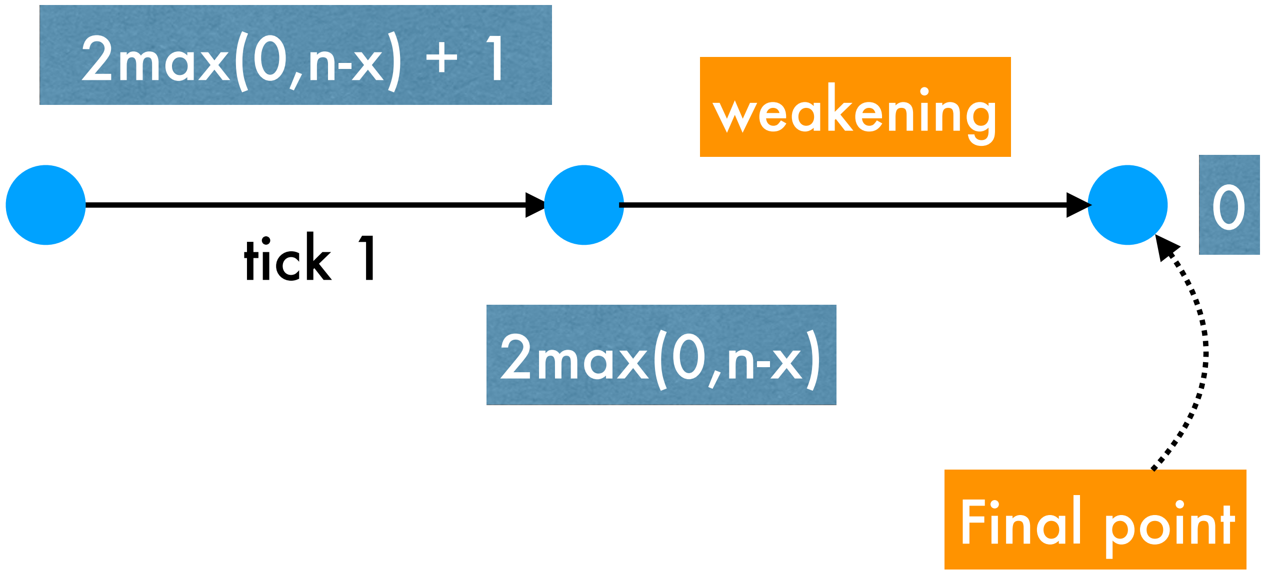
Example analysis on random walk



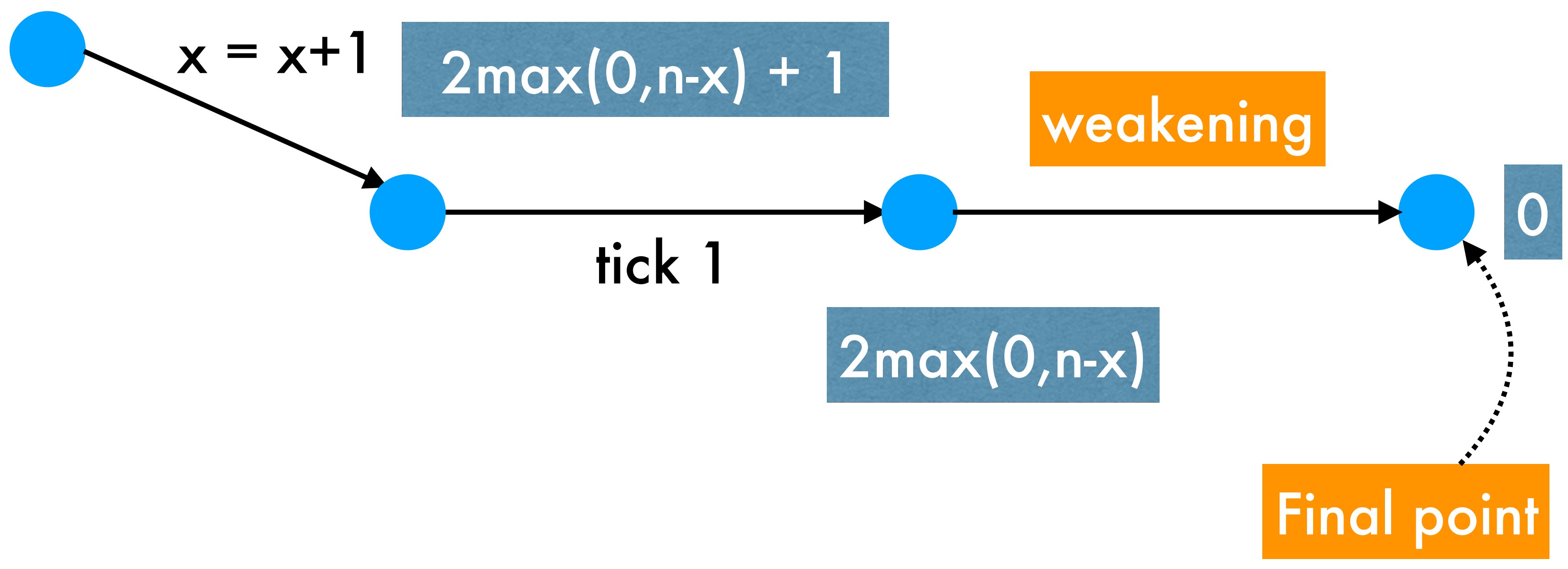
Example analysis on random walk



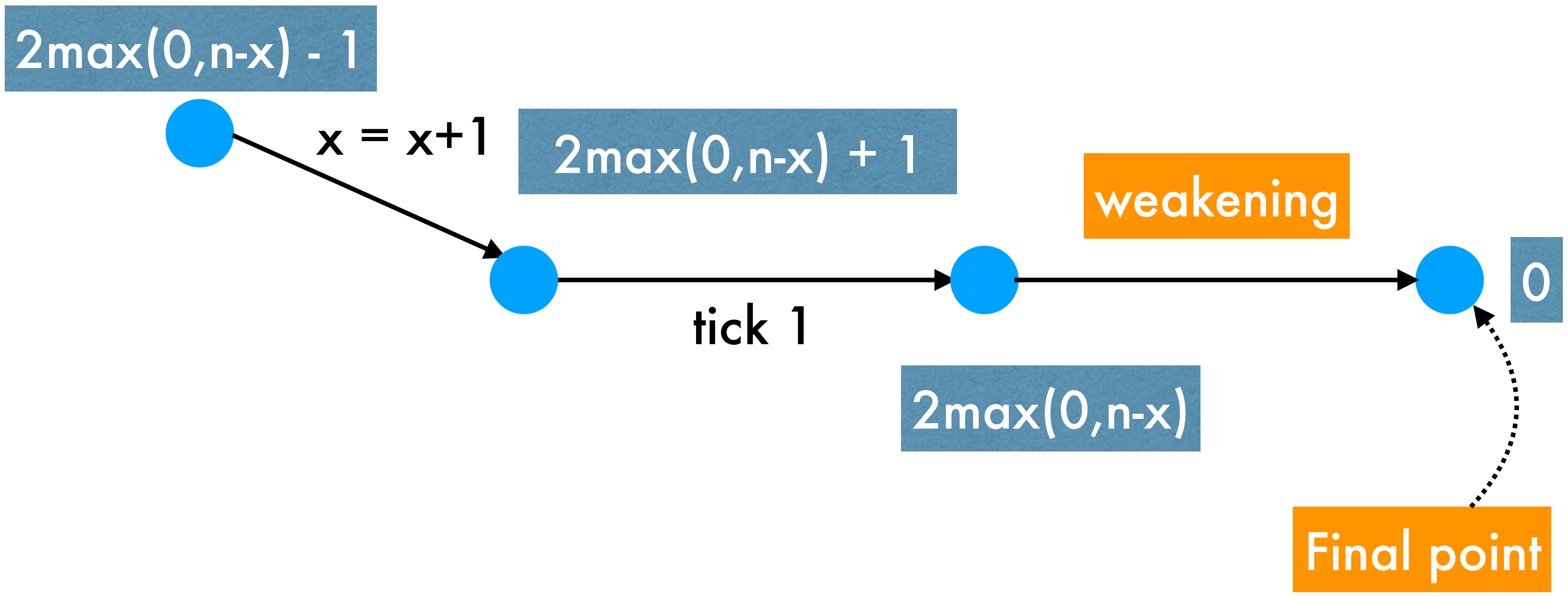
Example analysis on random walk



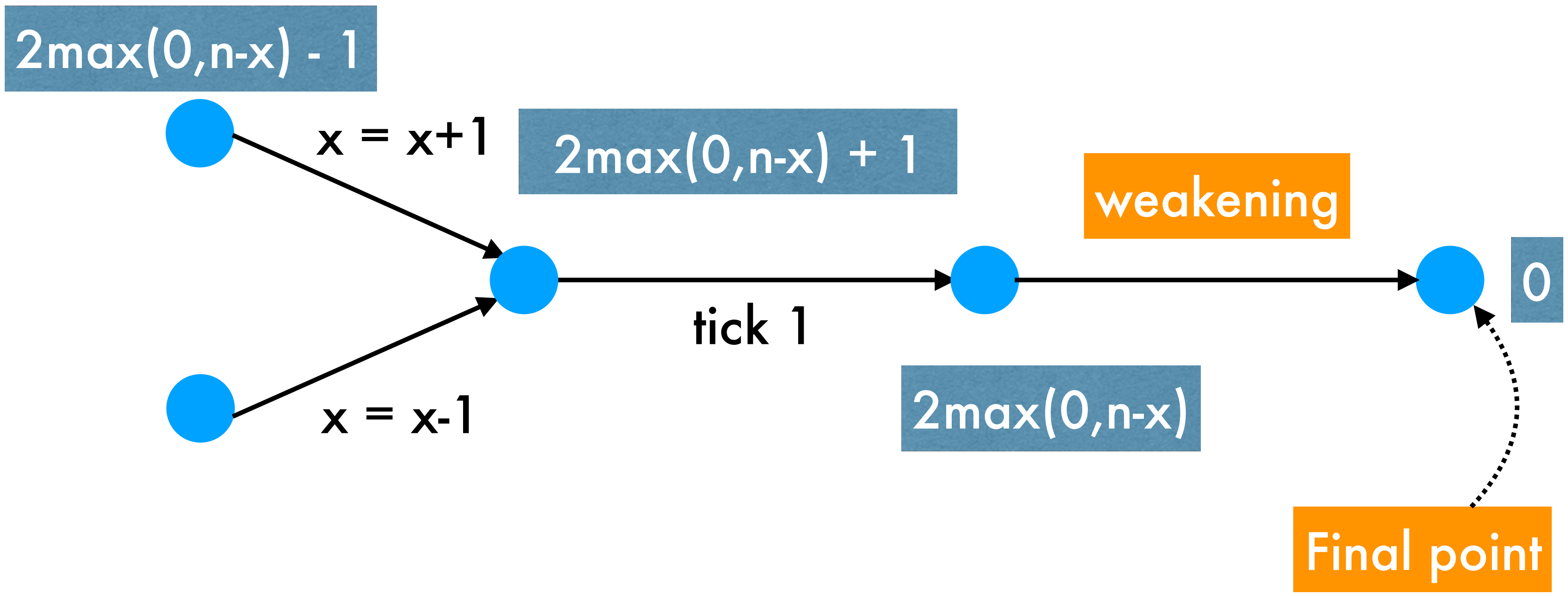
Example analysis on random walk



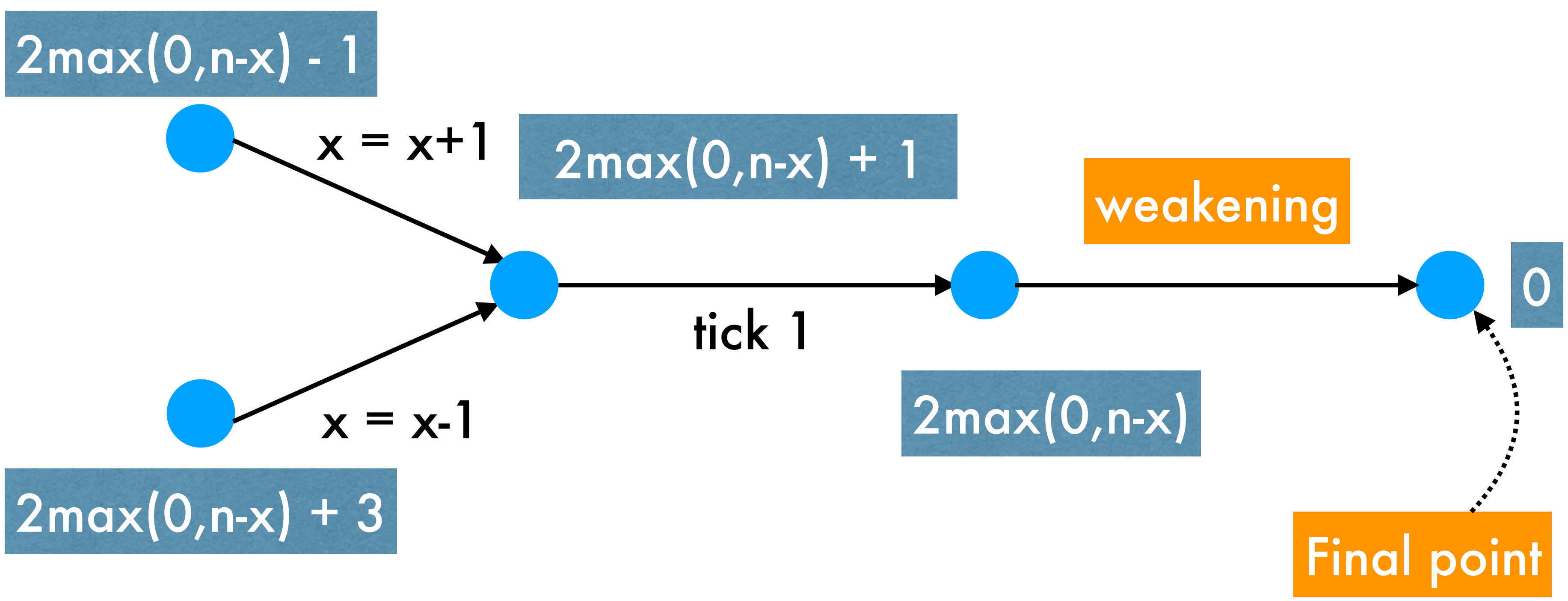
Example analysis on random walk



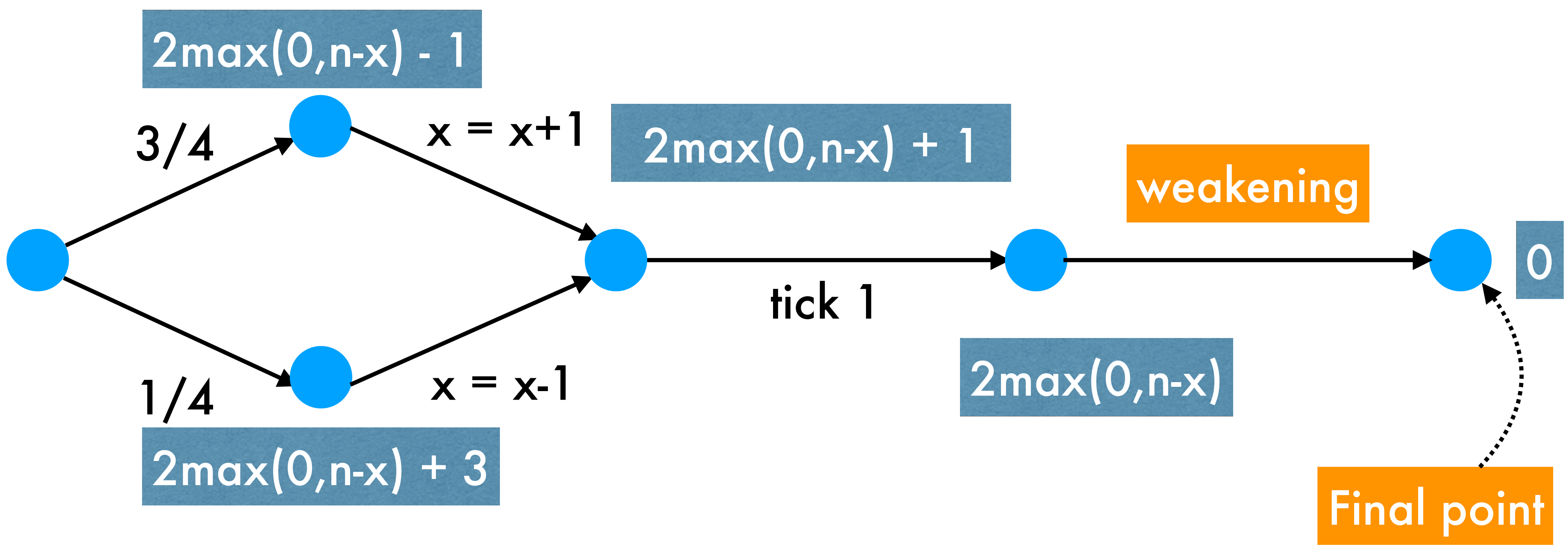
Example analysis on random walk



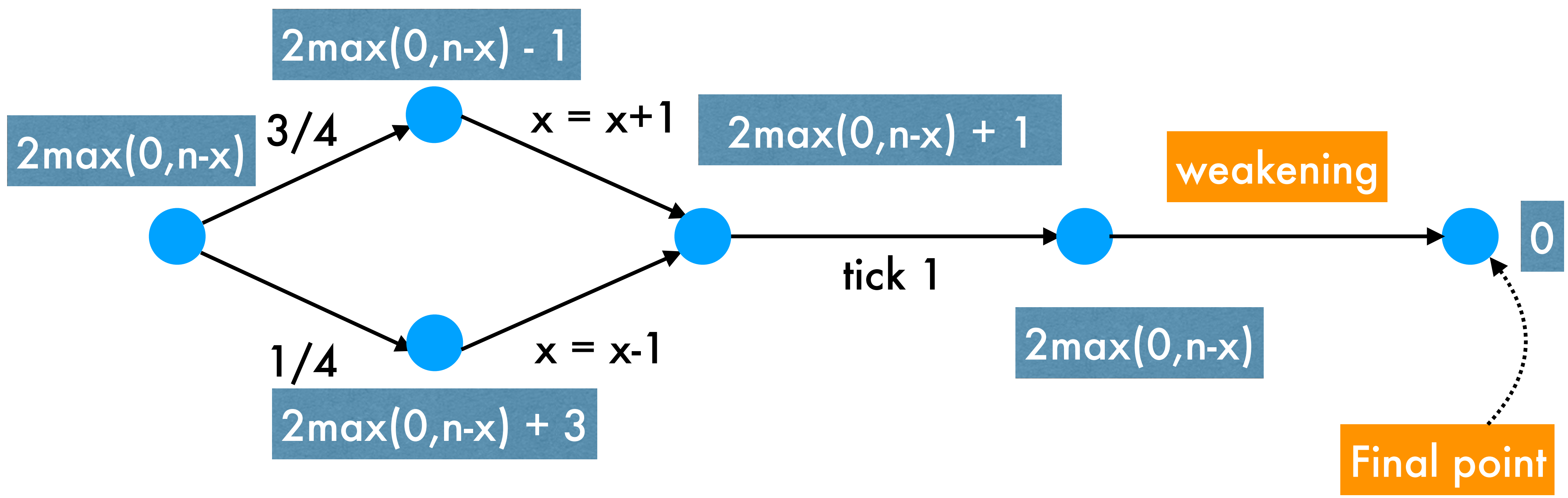
Example analysis on random walk



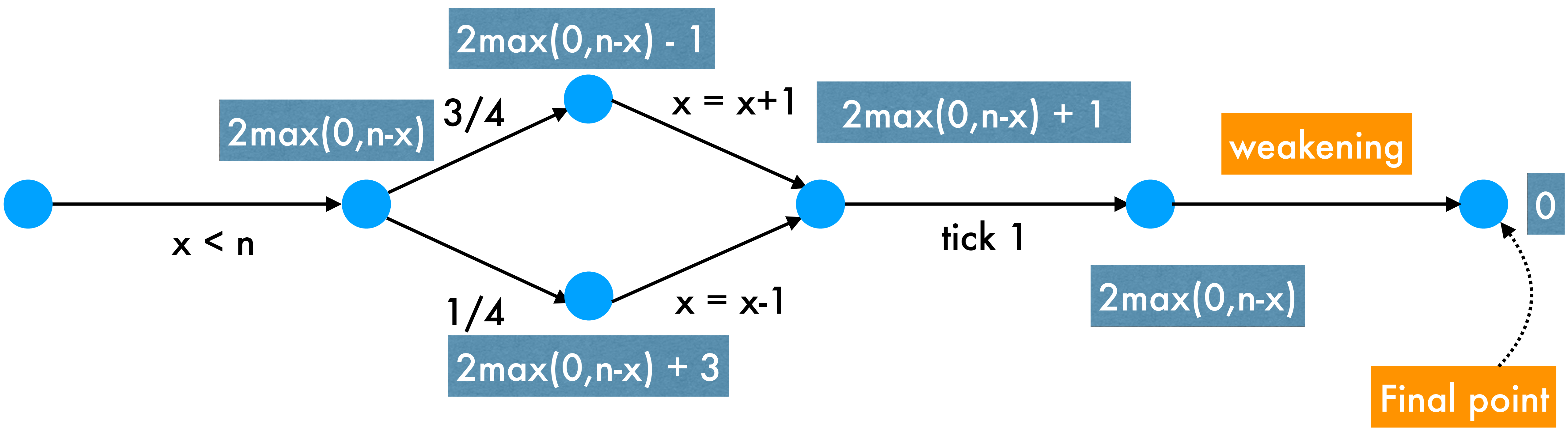
Example analysis on random walk



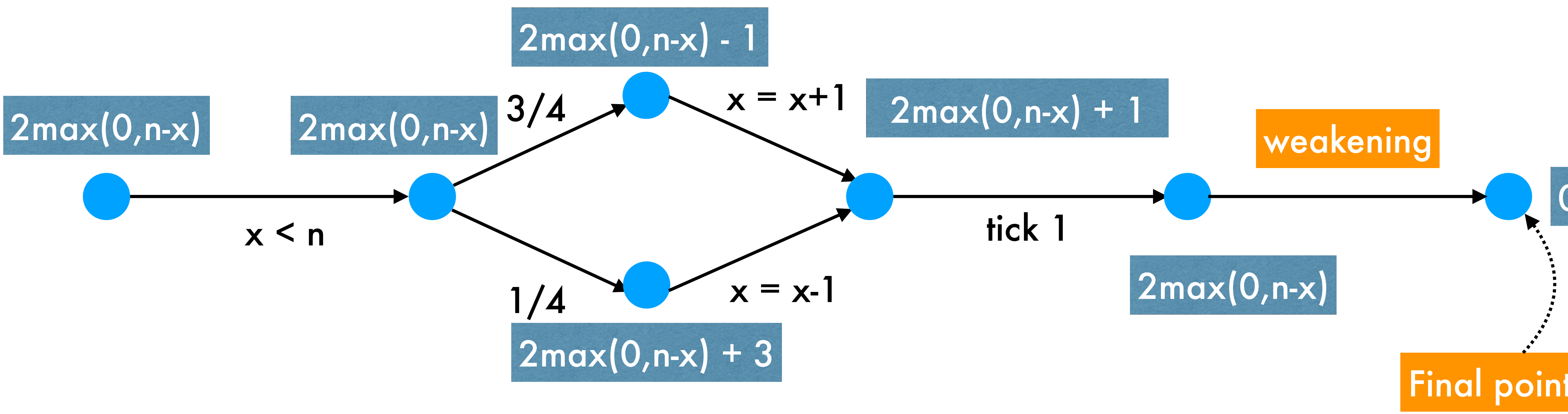
Example analysis on random walk



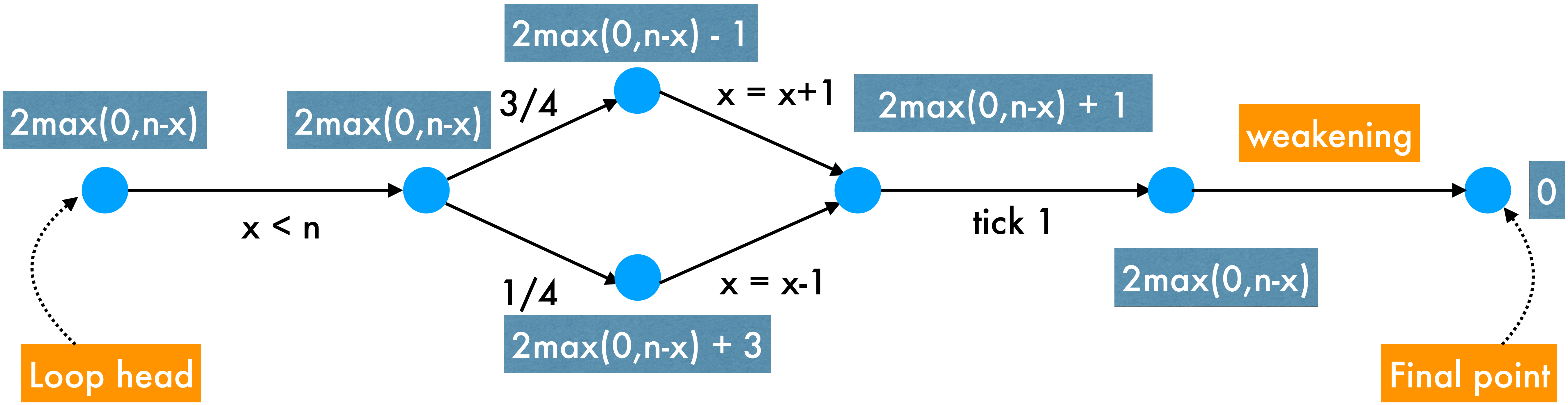
Example analysis on random walk



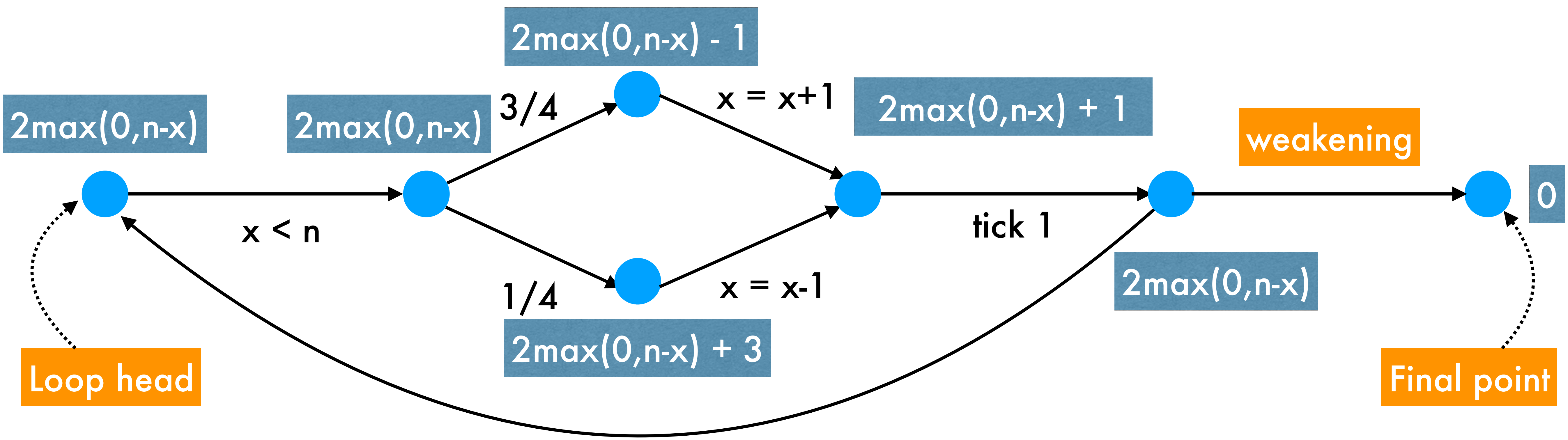
Example analysis on random walk



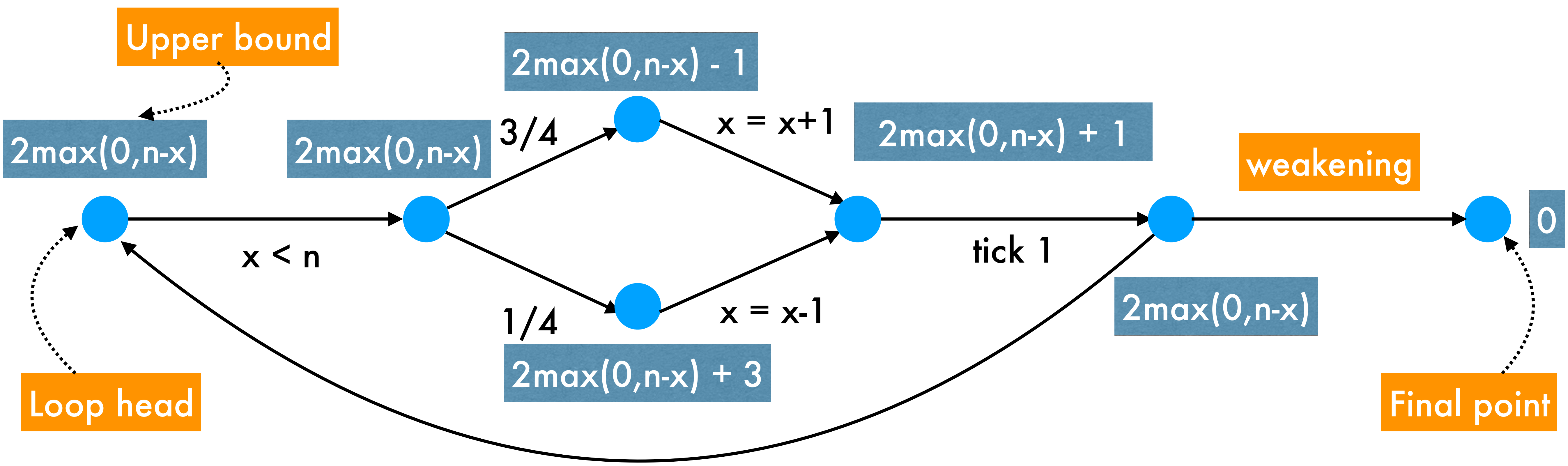
Example analysis on random walk



Example analysis on random walk



Example analysis on random walk



Summary

Contributions

- First automatic expected bound analysis for probabilistic program compilation
- Multivariate polynomial bounds with very precise constant factors
- Practical implementation for imperative probabilistic programs

Limitations

- Non-polynomial bounds
- Discrete probability distributions with finite domains

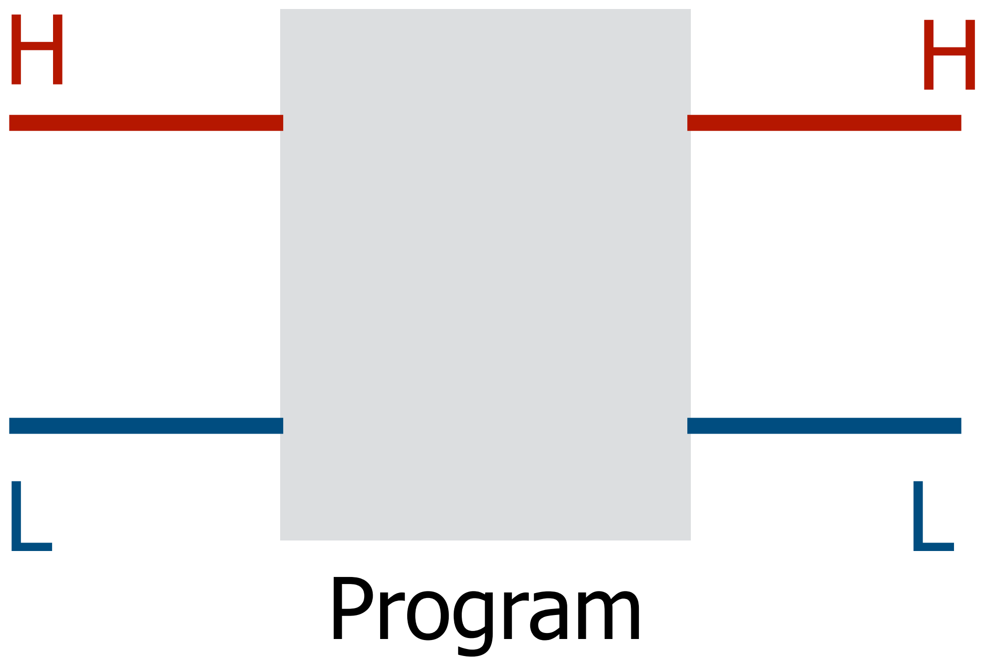
Outline

- Why quantitative analysis?
- Static analysis for probabilistic programming
- Type system for enforcing security [Oakland ’17]
- Future research directions

Noninterference

Attacker model

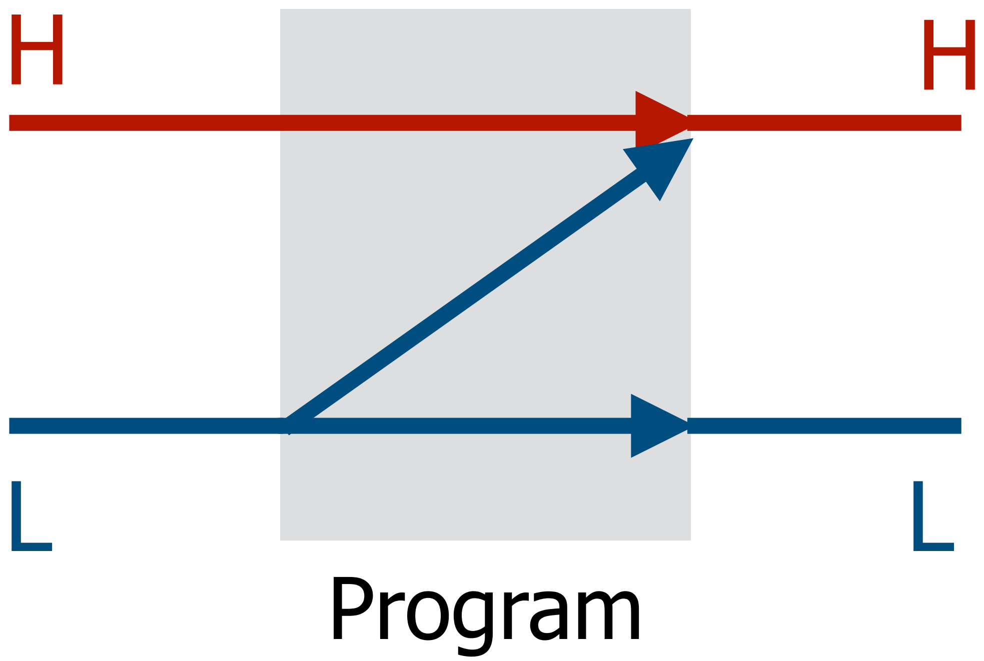
- Observe public outputs
- Control public inputs



Noninterference

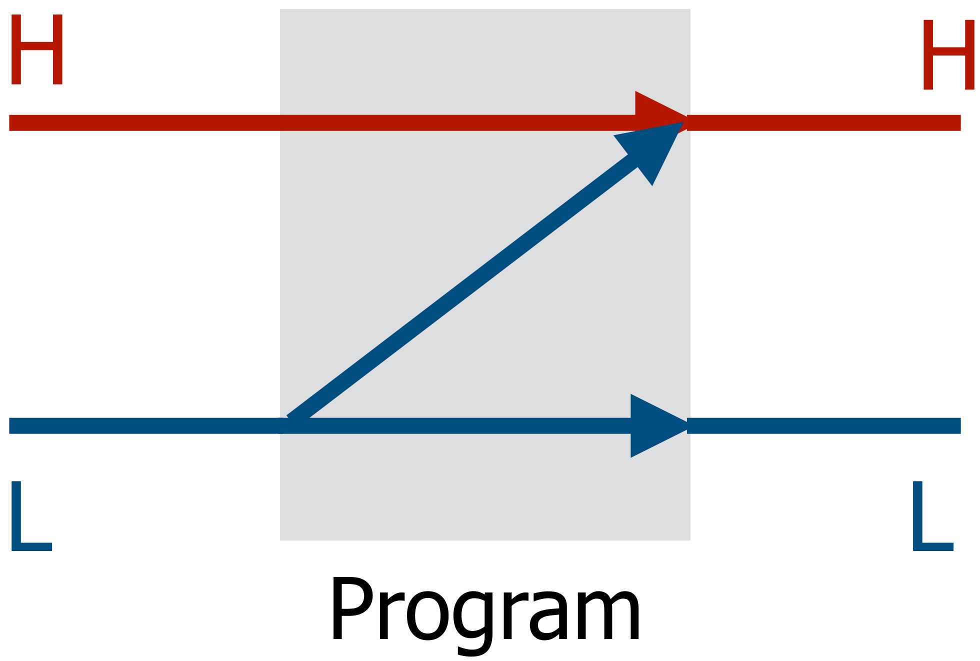
Attacker model

- Observe public outputs
- Control public inputs



- No secret data (H) flows to public data (L). Or secret data does not affect public data
- By observing and controlling public data, the attacker learns nothing about secret data

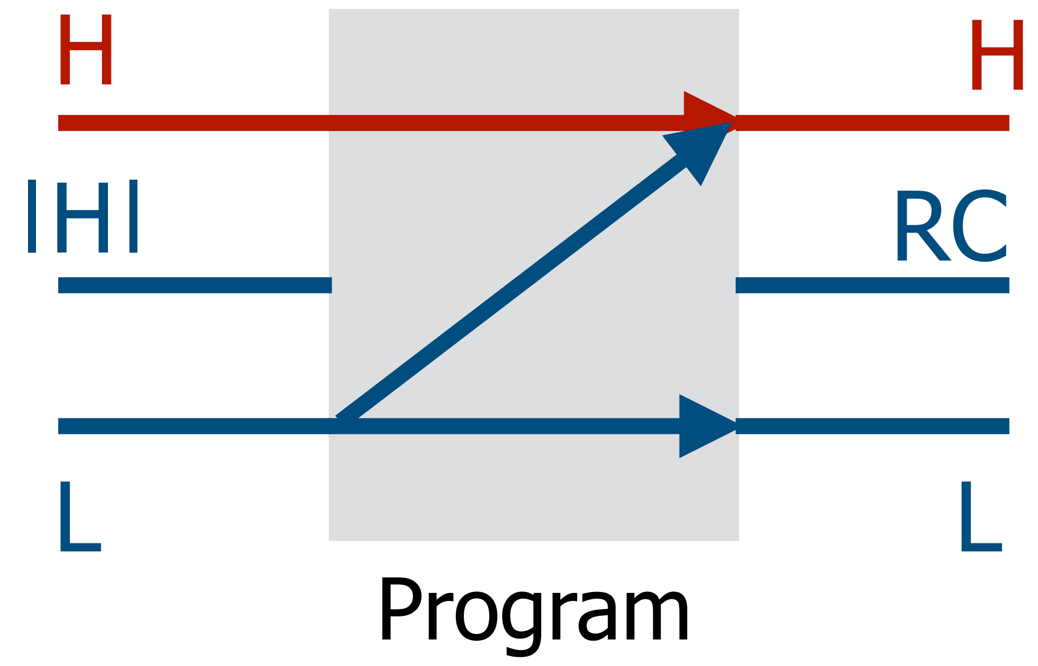
Flow to resource usage



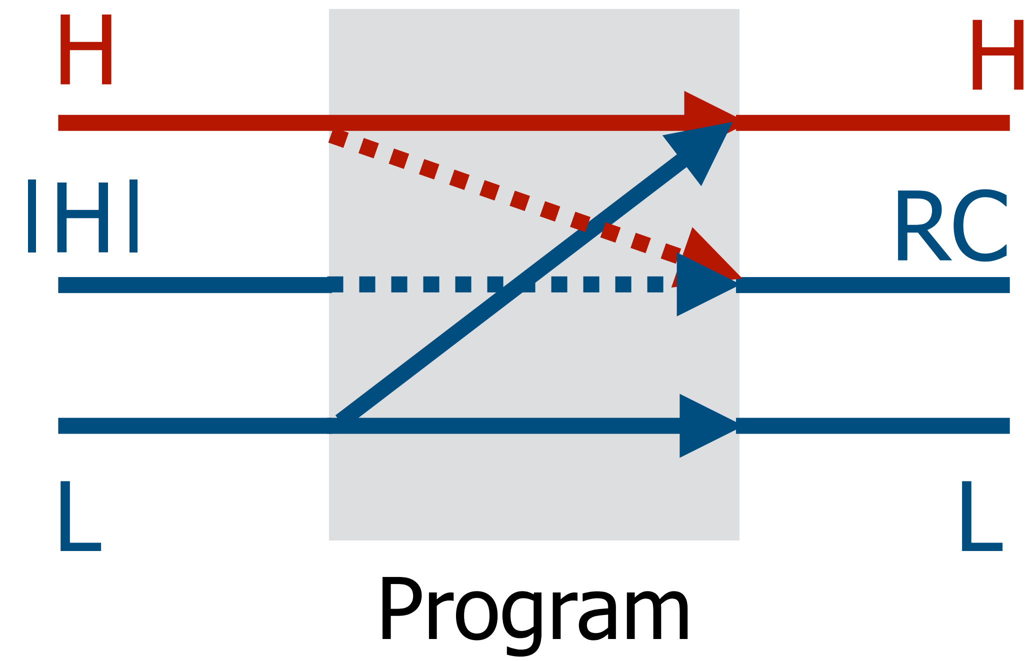
Flow to resource usage

Attacker model

- Observe public outputs
- Control public inputs
- Observe the total resource usage, e.g., execution time, energy consumption, …
- Observe the sizes of secret data (|H|)



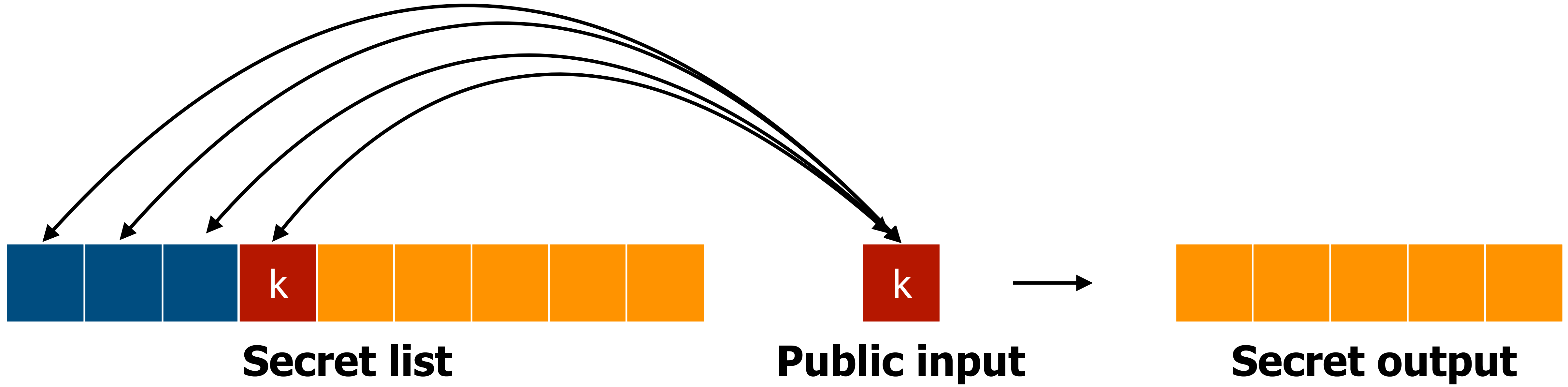
Flow to resource usage



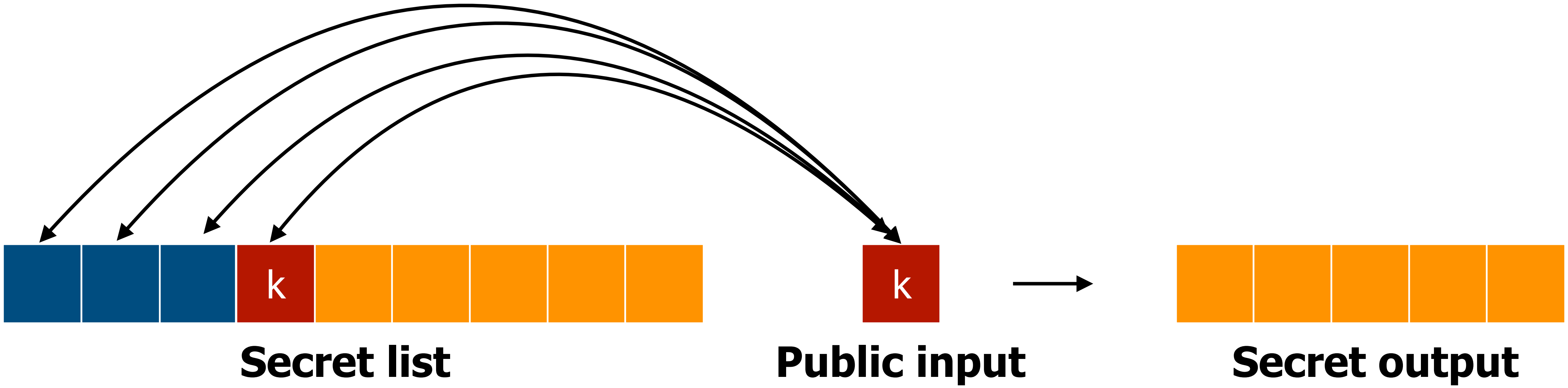
Attacker model

- Observe public outputs
- Control public inputs
- Observe the total resource usage, e.g., execution time, energy consumption, …
- Observe the sizes of secret data (|H|)
- Nothing about the information flow from H or |H| to resource consumption (RC). Can H flows to RC?
- Noninterference cannot reason about program security

Example

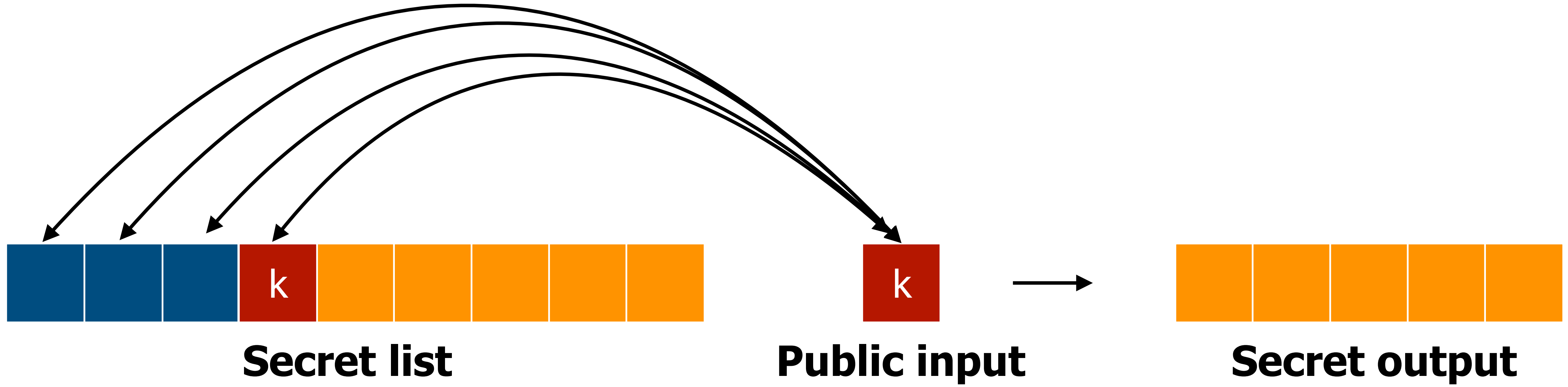


Example



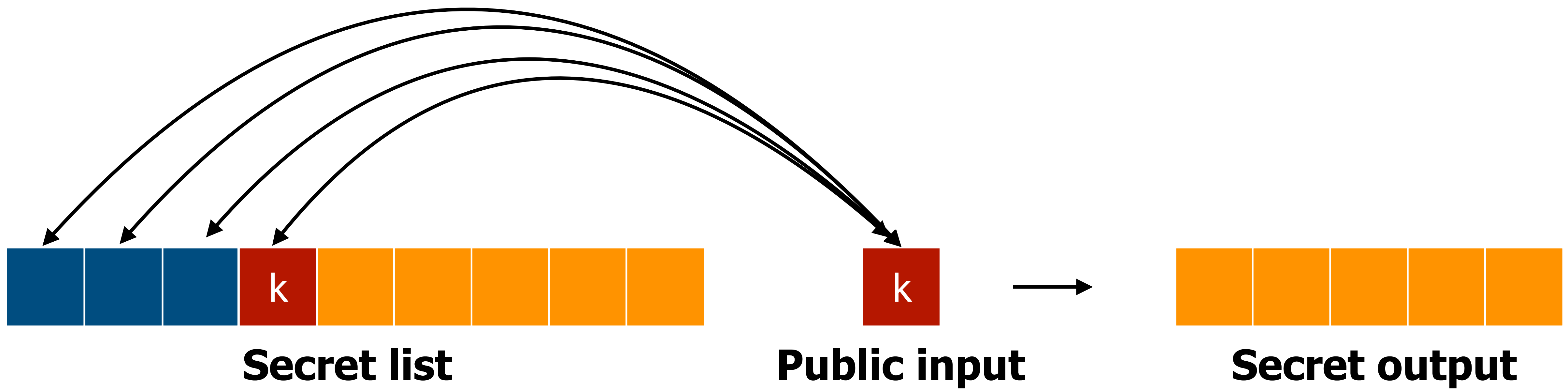
- Checks sequentially elements until a match for k is found

Example



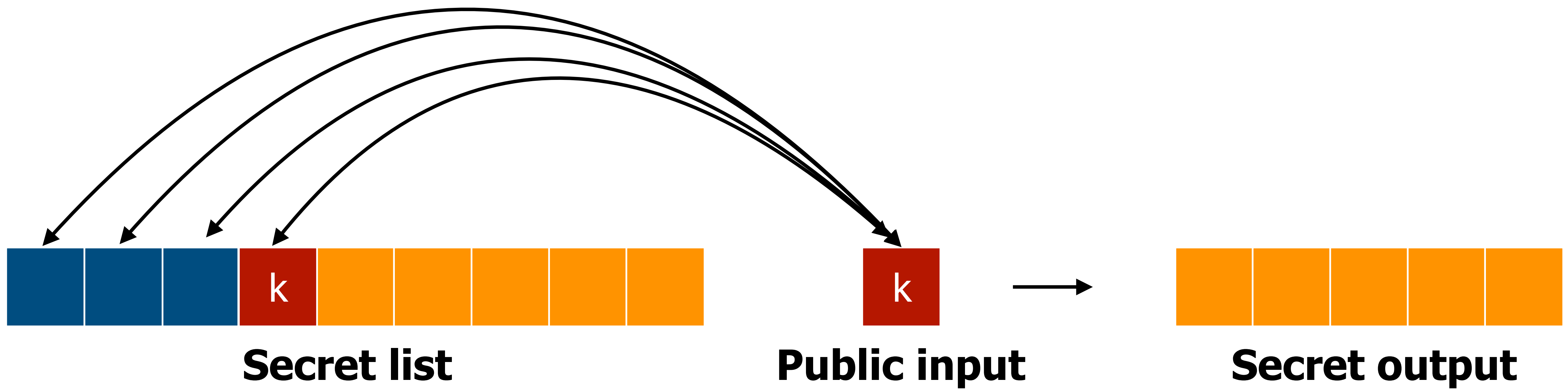
- Checks sequentially elements until a match for k is found
- Returns a list from k to the end of the list

Example



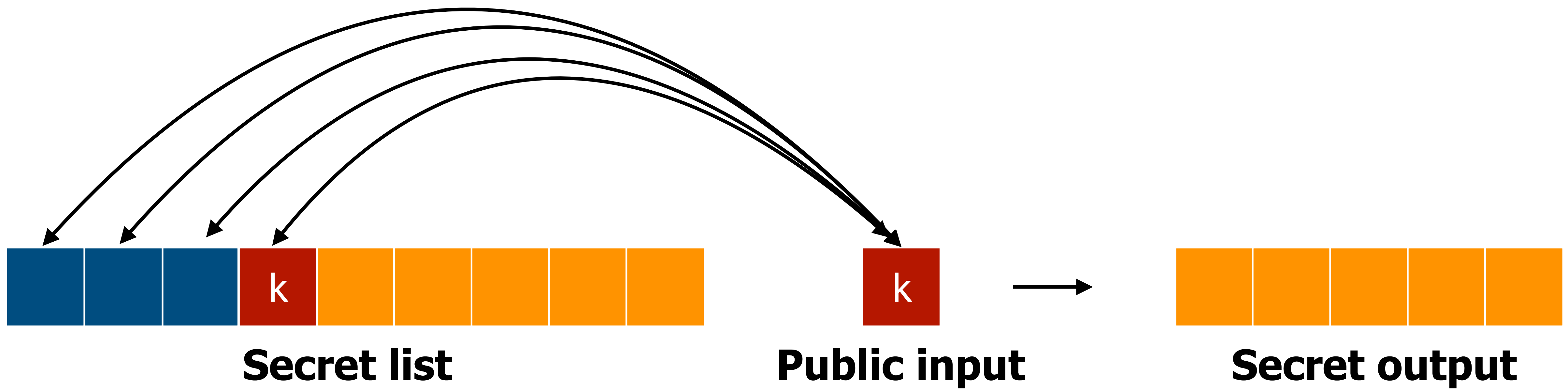
- Checks sequentially elements until a match for k is found
- Returns a list from k to the end of the list
- Each comparison of k with a list element takes 1 time unit

Example



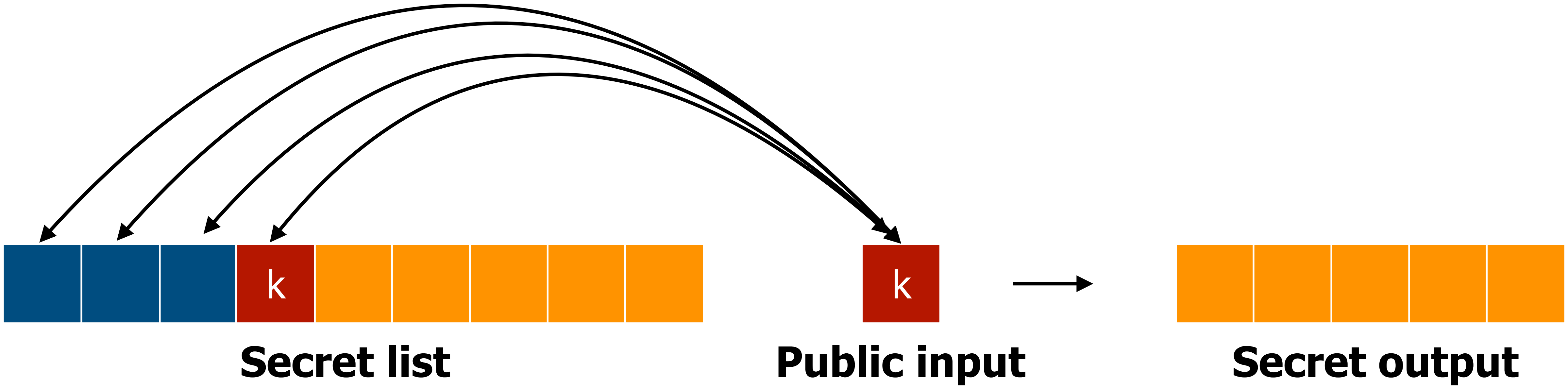
- Checks sequentially elements until a match for k is found
- Returns a list from k to the end of the list
- Each comparison of k with a list element takes 1 time unit
- By observing the total execution time and controlling k, the attacker can reveal the secret list

Example



- Checks sequentially elements until a match for k is found
- Returns a list from k to the end of the list
- Each comparison of k with a list element takes 1 time unit
- By observing the total execution time and controlling k, the attacker can reveal the secret list
- With k = 1, if execution time is 4 then the fourth element is 1

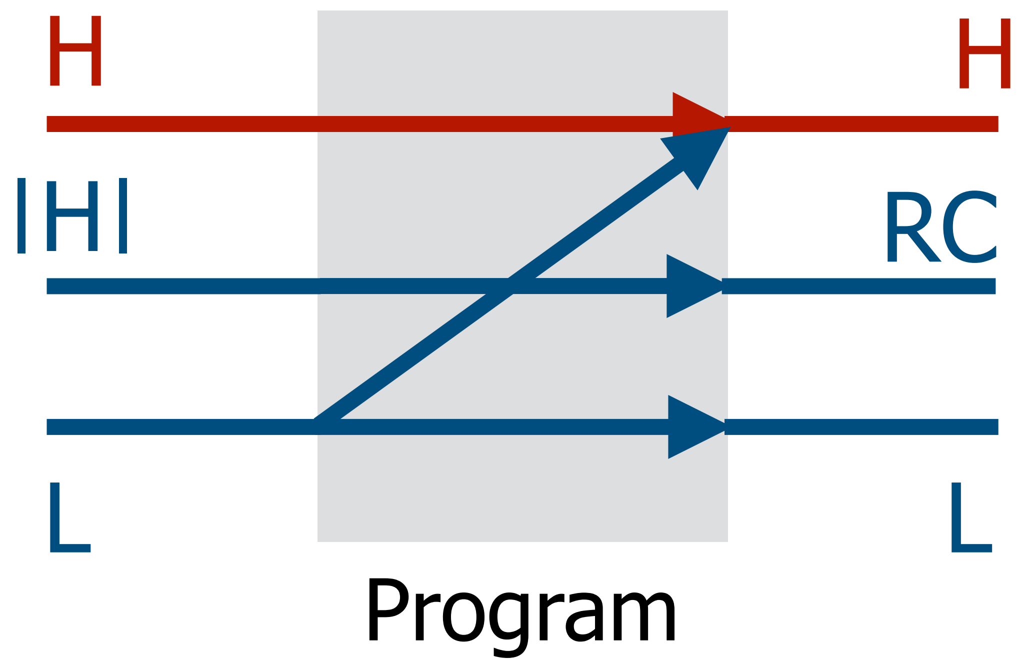
Example



- Checks sequentially elements until a match for k is found
- Returns a list from k to the end of the list
- Each comparison of k with a list element ta
- By observing the total execution time and
attacker can reveal the secret list
- With k = 1, if execution time is 4 then the fourth element is 1

Timing side-channel attack

Resource-aware noninterference



- No secret data (H) flows to public data (L)
- All executions where sizes of secret data are fixed, produce total constant resource consumption
- Observing resource consumption tells nothing about secret data

Type-based technique

- Resource type system proves that resource consumption is constant if input sizes are fixed
- Security type system co-operating with resource type system enforces resource-aware non-interference
- Quantification of information leakage of non-constant-resource programs
- Interactive and automatic program repair

Type-based technique

Focus of this part

- … proves that resource consumption is … are fixed
- Security type system co-operating with resource type system enforces resource-aware non-interference
- Quantification of information leakage of non-constant-resource programs
- Interactive and automatic program repair

Verification

Cost model

Annotated program

```
let rec trunc(l,k) =
  match l with
  | [] -> []
  | x::xs ->
    if x = k then
      tick(1); xs
    else
      tick(1); trunc(xs,k)
```

Security type system ⟷ **Resource type system**

Secure

Automatic repair

Verification

Cost model

Annotated program

```
let rec trunc(l,k) =
  match l with
  | [] -> []
  | x::xs ->
    if x = k then
      tick(1); xs
    else
      tick(1); trunc(xs,k)
```

Specifies confidential level of inputs & outputs

Security type system ↔ **Resource type system**

Secure

Automatic repair

Verification

Cost model

Specifies resource usage of primitive constructs

Annotated program

```
let rec trunc(l,k) =
  match l with
   | [] -> []
   | x::xs ->
     if x = k then
       tick(1); xs
     else
       tick(1); trunc(xs,k)
```

Specifies confidential level of inputs & outputs

Security type system ↔ **Resource type system**

→ **Secure**

→ **Automatic repair**

RAML - Resource Aware ML

- Accepts functional programs written in a subset of OCaml
- Infers linear and polynomial resource consumption
- Evaluates and proves resource-aware noninterference of common primitive functions, functions related to cryptography, and database query

Constant Function	Metric	Resource Usage	Time
cond_rev : $(L(\text{int}), L(\text{int}), \text{bool}) \to \text{unit}$	steps	$13n+13x+35$	0.03s
trunc_rev : $(L(\text{int}), \text{int}) \to L(\text{int})$	function calls	$1n$	0.06s
ipquery : $L(\text{logline}) \to (L(\text{int}), L(\text{int}))$	steps	$86n+99$	0.86s
kmeans : $L(\text{float}, \text{float}) \to L(\text{float}, \text{float})$	steps	$1246n+3784$	8.18s
tea_enc : $(L(\text{int}), L(\text{int}), \text{nat}) \to L(\text{int})$	ticks	$128n^2z+32nxz+1184nz+96n+128z+96$	13.73s
tea_dec : $(L(\text{int}), L(\text{int}), \text{nat}) \to L(\text{int})$	ticks	$128n^2z+32nxz+1184nz+96n+96z+96$	14.34s

Enforcing resource-aware noninterference

- Check noninterference property first
- Two extreme ways: global and local reasoning
- Global reasoning: using the resource type system to check the whole program is constant-resource w.r.t secret data sizes
  - Not efficient (e.g., requires to reason about parts not affected by secret data)
- Local reasoning: ensuring every part affected by secret data is resource-aware noninterference
  - Not sufficient (e.g., rejects valid programs)

Example

Program is resource-aware interference but rejected by the local reasoning

```
let f(b,l1,l2) =
  let l3 = if b then
             rev(l2); l1
           else
             rev(l1); l2
  in rev(l3)
```

Example

Program is resource-aware interference but rejected by the local reasoning

Secret data

```
let f(b,l1,l2) =
  let l3 = if b then
             rev(l2); l1
           else
             rev(l1); l2
  in rev(l3)
```

Example

Program is resource-aware interference but rejected by the local reasoning

Secret data

```
let f(b,l1,l2) =
  let l3 = if b then
             rev(l2); l1
           else
             rev(l1); l2
  in rev(l3)
```

List reversal is resource-aware noninterference

Example

Program is resource-aware interference but rejected by the local reasoning

Secret data

```
let f(b,l1,l2) =
  let l3 = if b then
             rev(l2); l1
           else
             rev(l1); l2
  in rev(l3)
```

List reversal is resource-aware noninterference

Non resource-aware noninterference

Example

Program is resource-aware interference but rejected by the local reasoning

Secret data

Resource-aware noninterference

```
let f(b,l1,l2) =
  let l3 = if b then
             rev(l2); l1
           else
             rev(l1); l2
  in rev(l3)
```

List reversal is resource-aware noninterference

Non resource-aware noninterference

Global and local reasoning

- Security type system uses a mix of both global and local reasoning
- Ensures that every expression affected by secret data is
  - resource-aware noninterference expression, or
  - a part of resource-aware noninterference expression
- Thus, total resource consumption is independent from the secret data

Local reasoning

Resource-aware noninterference Non resource-aware interference

Local reasoning

Resource-aware noninterference

Non resource-aware interference

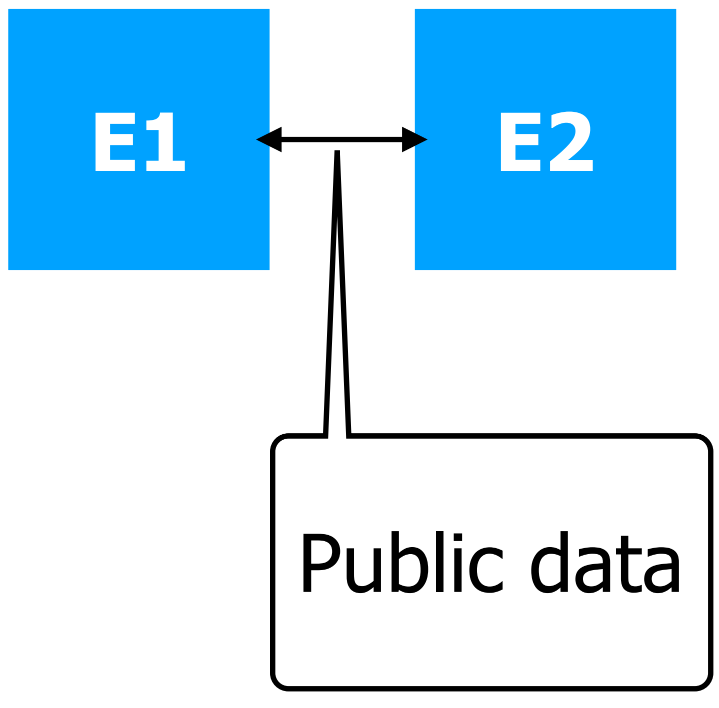
E1

Local reasoning

Resource-aware noninterference   Non resource-aware interference

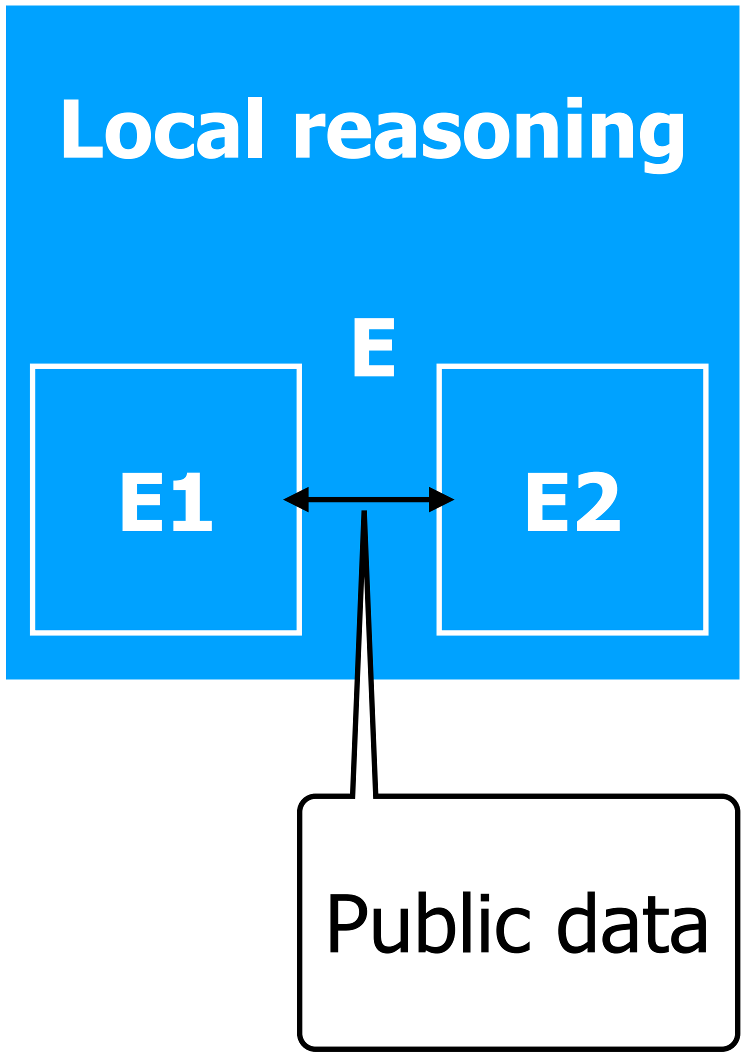
Local reasoning

Resource-aware noninterference   Non resource-aware interference

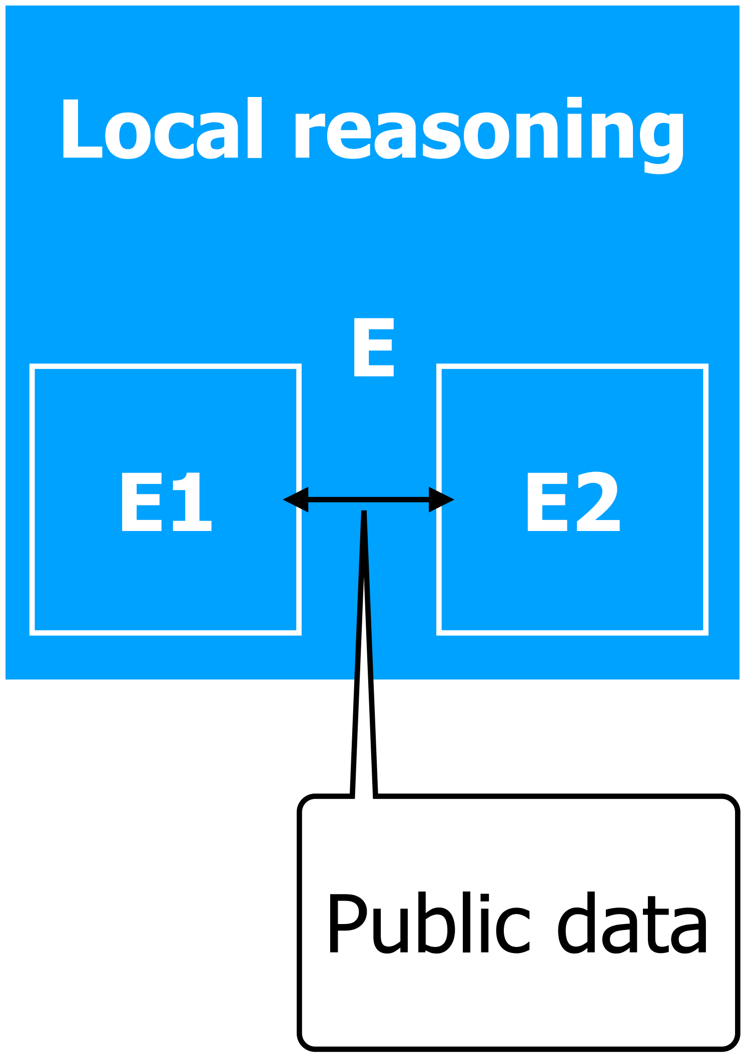


Local reasoning

Resource-aware noninterference | Non resource-aware interference



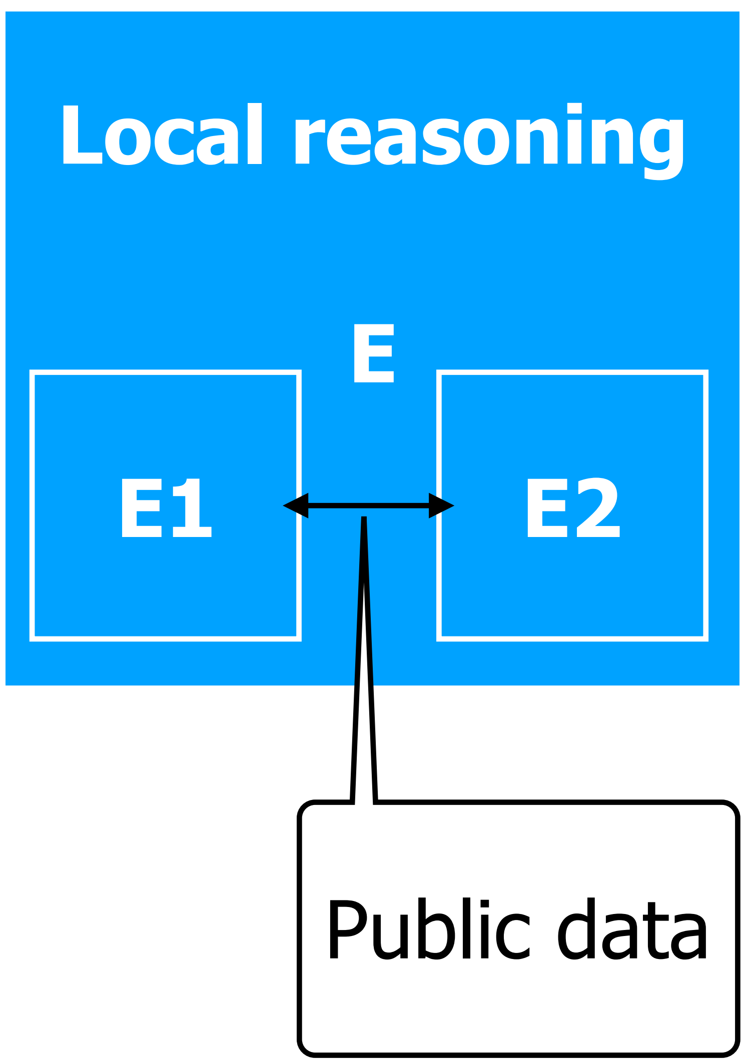
Local reasoning



The following typing rule reflects the local reasoning

Local reasoning

Resource-aware noninterference | Non resource-aware interference



The following typing rule reflects the local reasoning

Resource-aware noninterference

Public data

$$\frac{\cdots \vdash^{\text{const}} e_1 : S \qquad \cdots \vdash^{\text{const}} e_2 : S \qquad L(x) \sqsubseteq \text{low}}{\cdots \vdash^{\text{const}} \text{if}(x, e_1, e_2) : S}$$

Global reasoning

Resource-aware noninterference

Non resource-aware interference

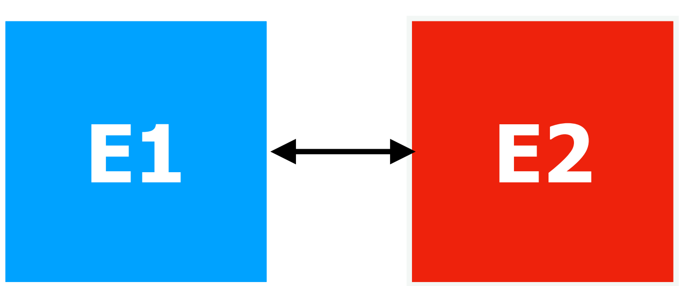
Unknown

Global reasoning

Resource-aware noninterference

Non resource-aware interference

Unknown

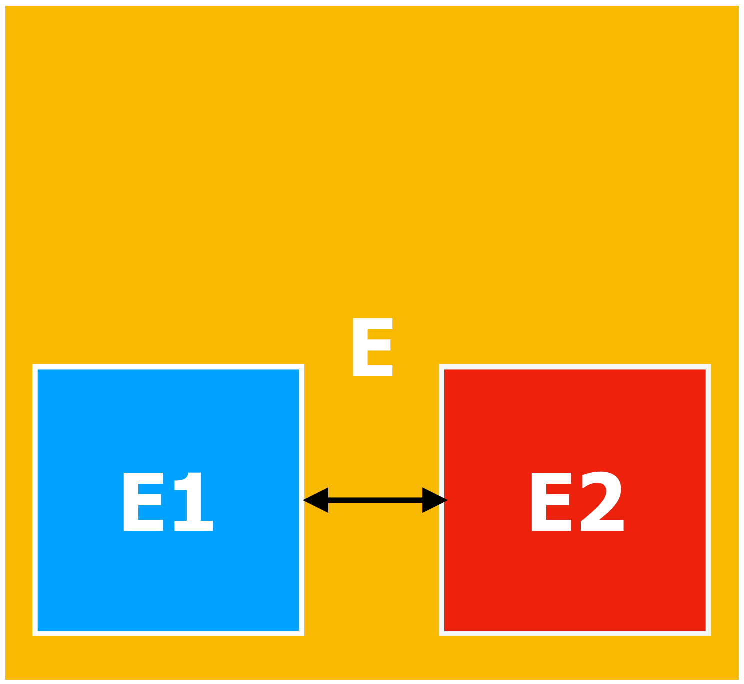


Global reasoning

Resource-aware noninterference

Non resource-aware interference

Unknown



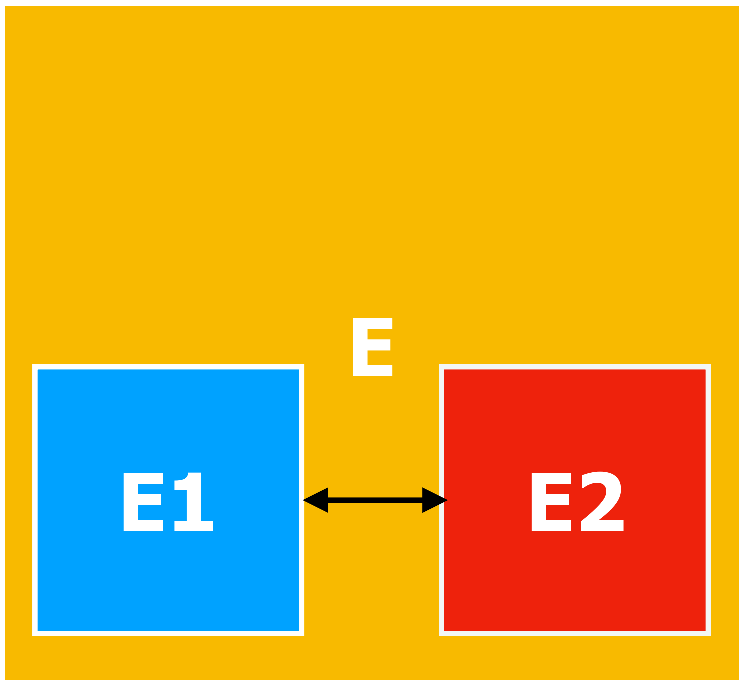
Global reasoning

Resource-aware noninterference

Non resource-aware interference

Unknown

E cannot be reasoned locally



Global reasoning

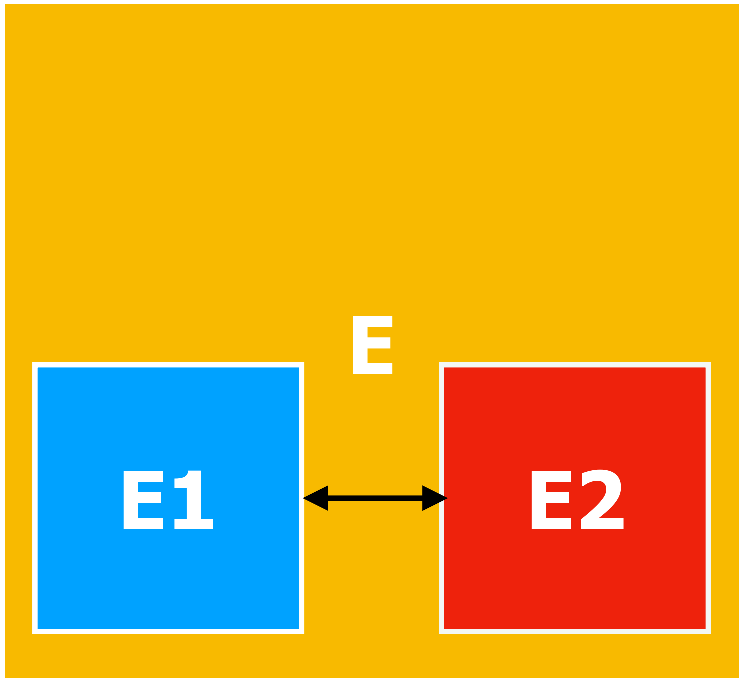
Resource-aware noninterference

Non resource-aware interference

Unknown

E cannot be reasoned locally

E needs to be checked for constant-resource globally w.r.t secrete data



Global reasoning

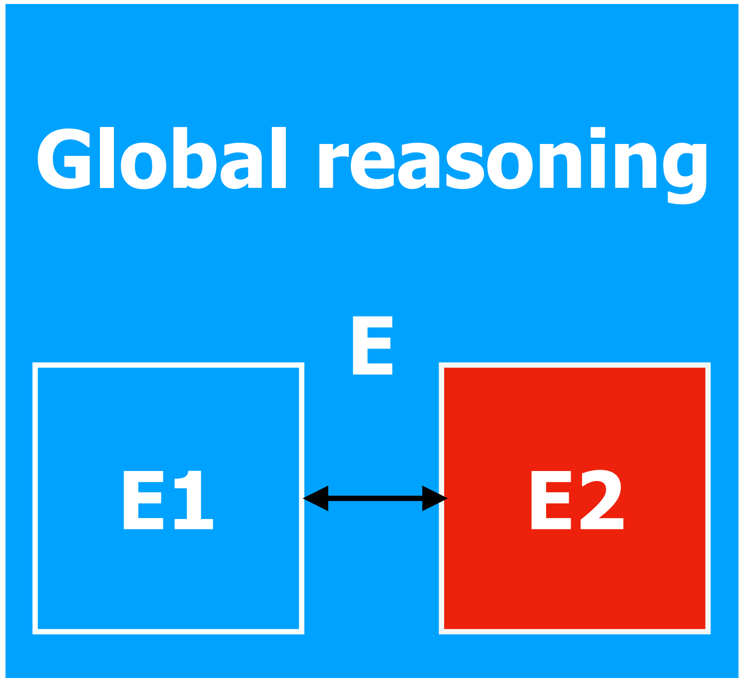
Resource-aware noninterference

Non resource-aware interference

Unknown

E cannot be reasoned locally

E needs to be checked for constant-resource globally w.r.t secrete data

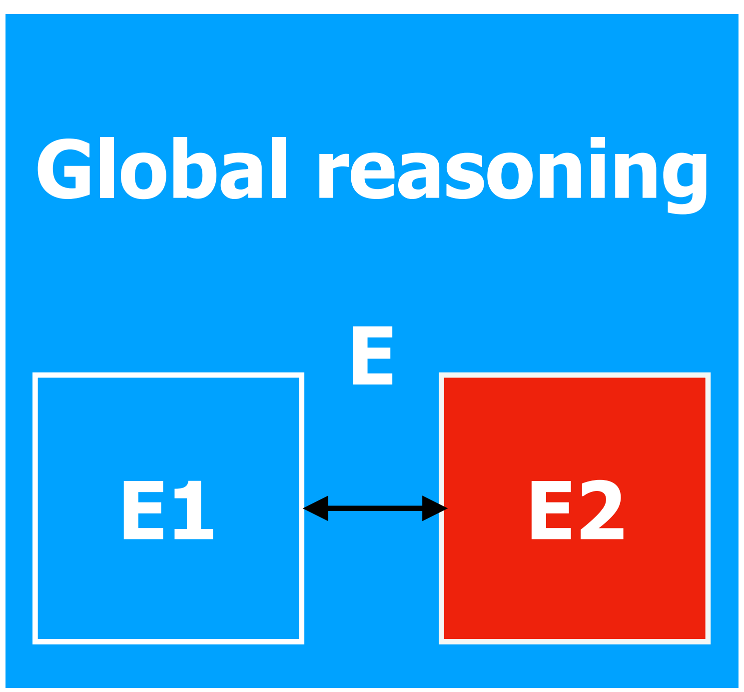


Global reasoning

Resource-aware noninterference

Non resource-aware interference

Unknown



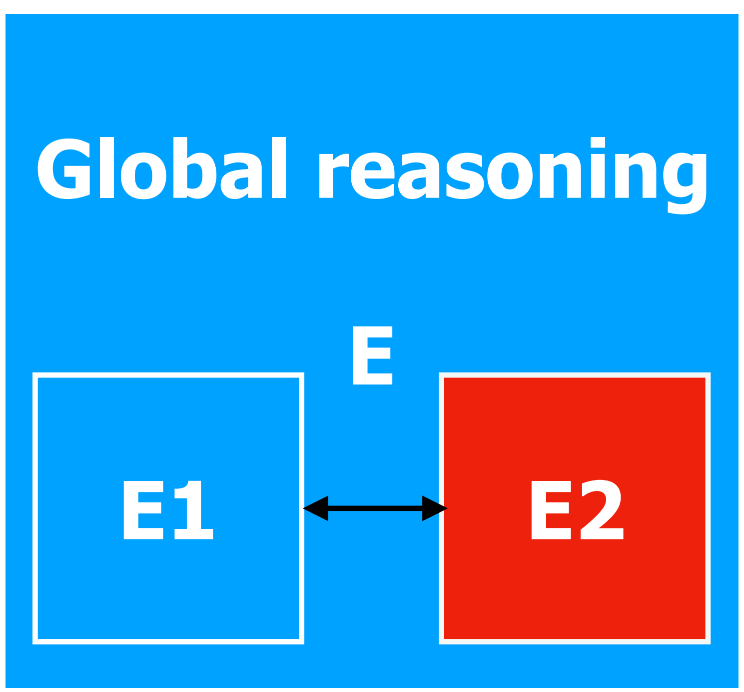
E cannot be reasoned locally

E needs to be checked for constant-resource globally w.r.t secrete data

The following typing rule reflects the global reasoning

Global reasoning

- Resource-aware noninterference
- Non resource-aware interference
- Unknown



E cannot be reasoned locally

E needs to be checked for constant-resource globally w.r.t secrete data

The following typing rule reflects the global reasoning

$$\frac{\cdots \vdash e : S \qquad \text{const}(e)}{\cdots \vdash^{\text{const}} e : S}$$

Checked by resource type system

Summary

Contributions

- Novel type system that tracks both information flow and resource usage
- First automatic repair that transforms programs to be constant-resource
- Practical implementation for OCaml programs

Limitations

- Only guarantee at programming language-level
- No hardware and compilation tools affects

Outline

- **Why quantitative analysis?**
- **Static analysis for probabilistic programming**
- **Type system for enforcing security**
- **Future research directions**

Research plan

Directions

- Developing static analysis for compilation of probabilistic programs
  - Application in formalizing and verifying machine-learning and robotic software
  - Application in reasoning about probabilistic security properties

Techniques for automation

- Programming language-based and compiler techniques such as type checking, program logic, and data-flow analysis
- Formal methods like model checking and theorem proving

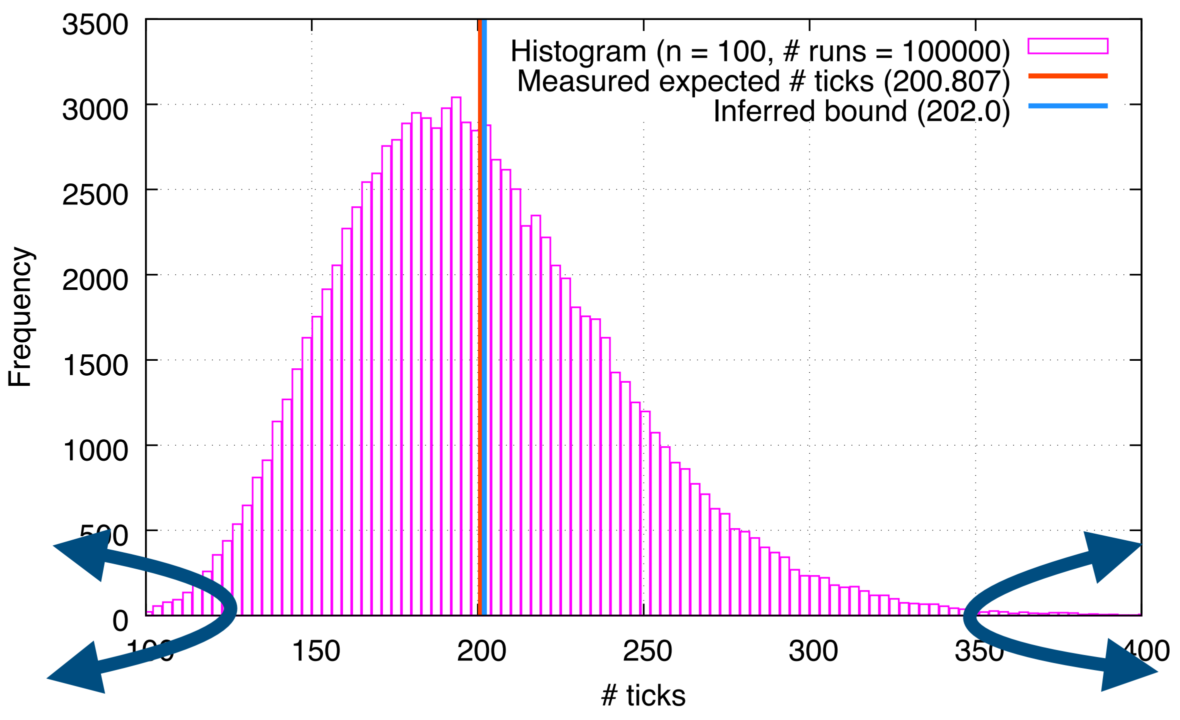
Tail-bounds analysis

Tail-bounds analysis

- Tails are parts of distribution "far" from the expected behavior
- Specify the probability of "something going wrong"
- Good for analyzing safety properties of programs

Tail-bounds analysis

- Tails are parts of distribution "far" from the expected behavior
- Specify the probability of "something going wrong"
- Good for analyzing safety properties of programs

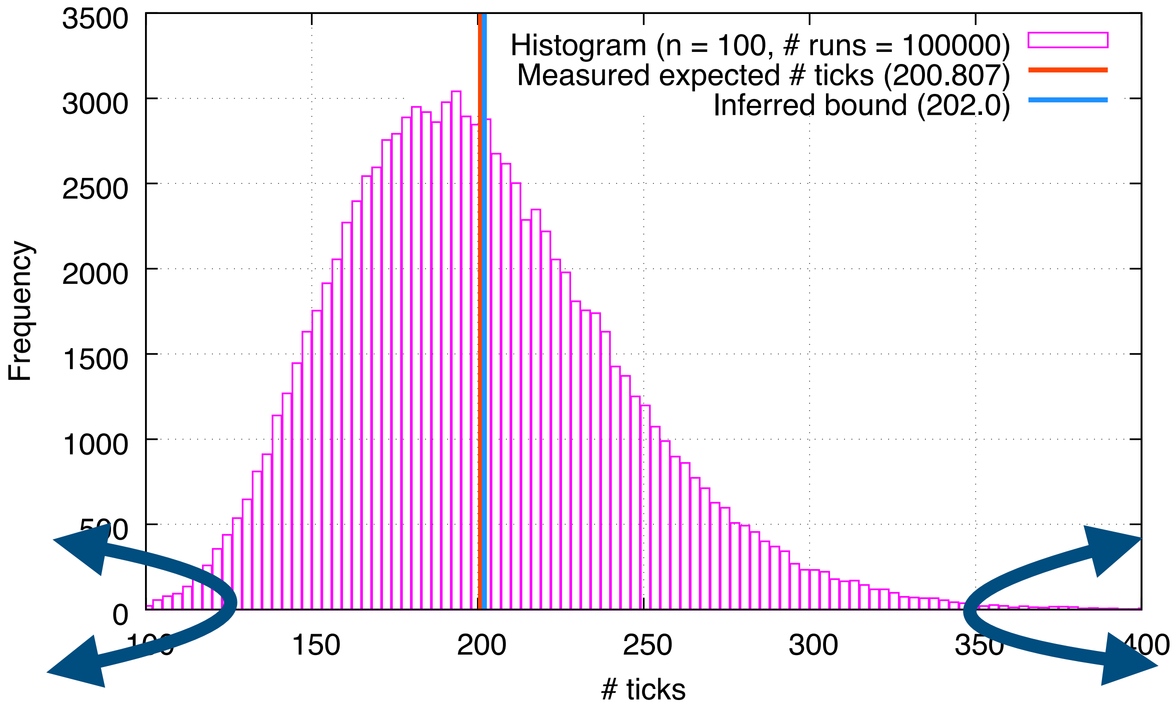


Tail-bounds analysis

- Tails are parts of distribution "far" from the expected behavior

- Specify the probability of "something going wrong"

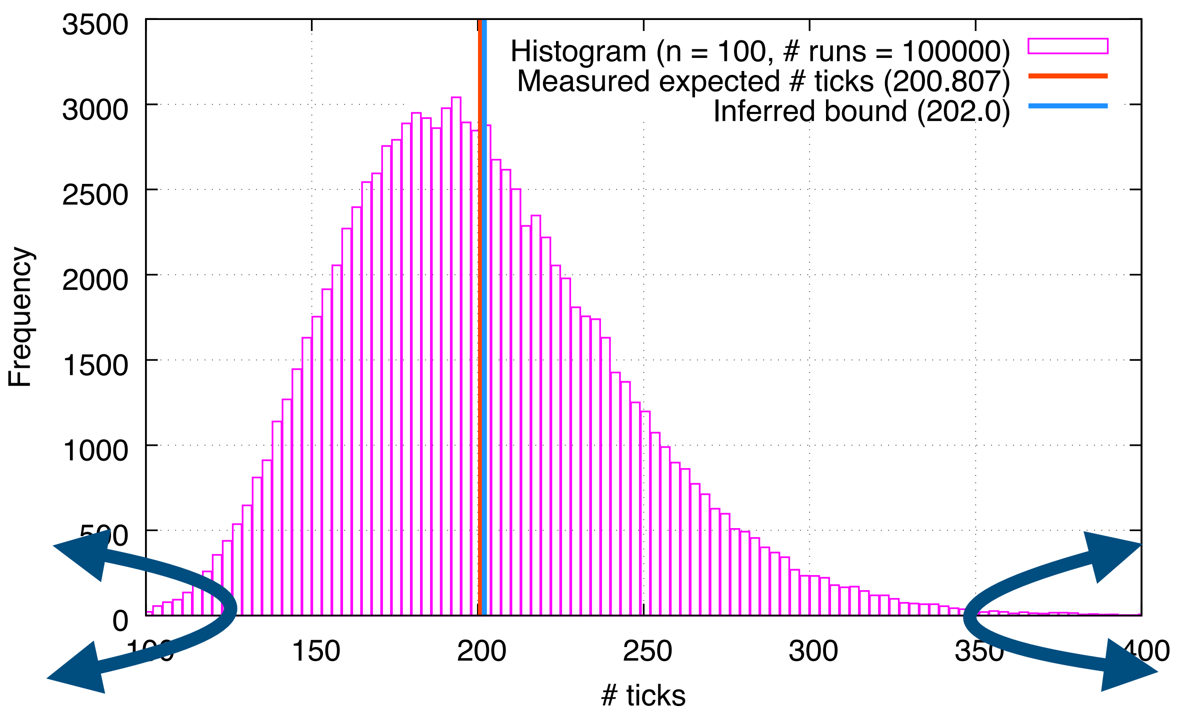
- Good for analyzing safety properties of programs



- The expected time taken is 200, what is the probability that the walker takes more than 800 time units?

Tail-bounds analysis

- Tails are parts of distribution "far" from the expected behavior
- Specify the probability of "something going wrong"
- Good for analyzing safety properties of programs



- The expected time taken is 200, what is the probability that the walker takes more than 800 time units?
- A perception system, e.g., deep neutral network in a self-driving car, makes decision in 10 milliseconds in average. What's the probability that it misses the hard realtime deadline which is 100 milliseconds?

Concentration inequality

- Given variance of a random variable X, its tail-bound probability can be computed using Cherbyshev inequality

$$\forall a > 0 . \mathbb{P}(|X - \mathbb{E}[X]| \geq a) \leq \frac{\mathrm{Var}[X]}{a^2}$$

- For example, the probability that the walker takes more than 800 time units is

$$\mathbb{P}(\mathrm{tick} - \mathbb{E}[\mathrm{tick}] \geq 3\mathbb{E}[\mathrm{tick}]) \leq \frac{1}{6n} = 0.00167$$

- Automatically infer upper-bounds on variances during the compilation based on the expected potential method

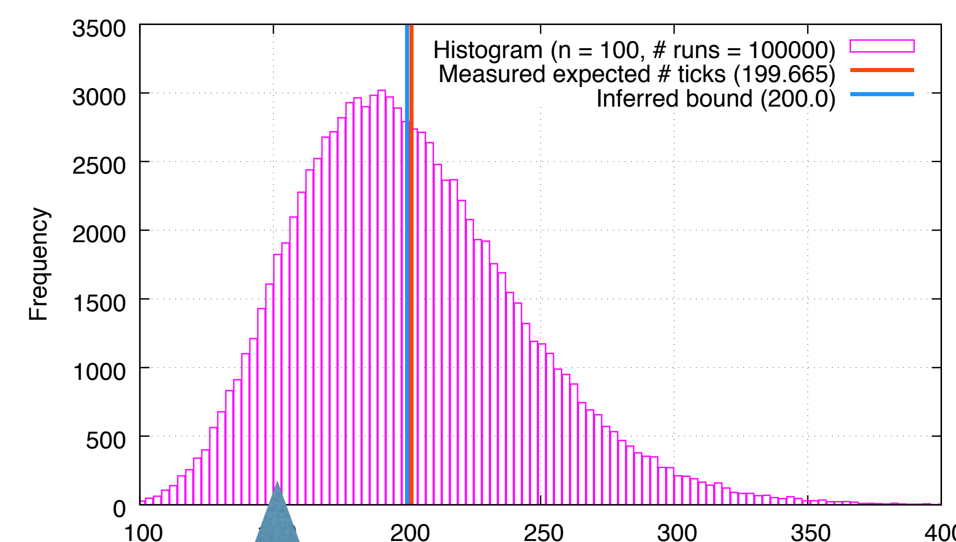
Quantitative analysis with martingales

Quantitative analysis with martingales

- Martingales are fundamental in mathematic and probability theory
- Consider a probabilistic program as a (infinite) sequence of random variables, the martingales can be used to reason about
  - Expected value of random variables
  - Tail-bound probability with Azuma-Hoeffding inequality

Quantitative analysis with martingales

- Martingales are fundamental in mathematatic and probability theory
- Consider a probabilistic program as a (infinite) sequence of random variables, the martingales can be used to reason about
  - Expected value of random variables
  - Tail-bound probability with Azuma-Hoeffding inequality
- With n = 100, martingale (2x - tick) is used to compute

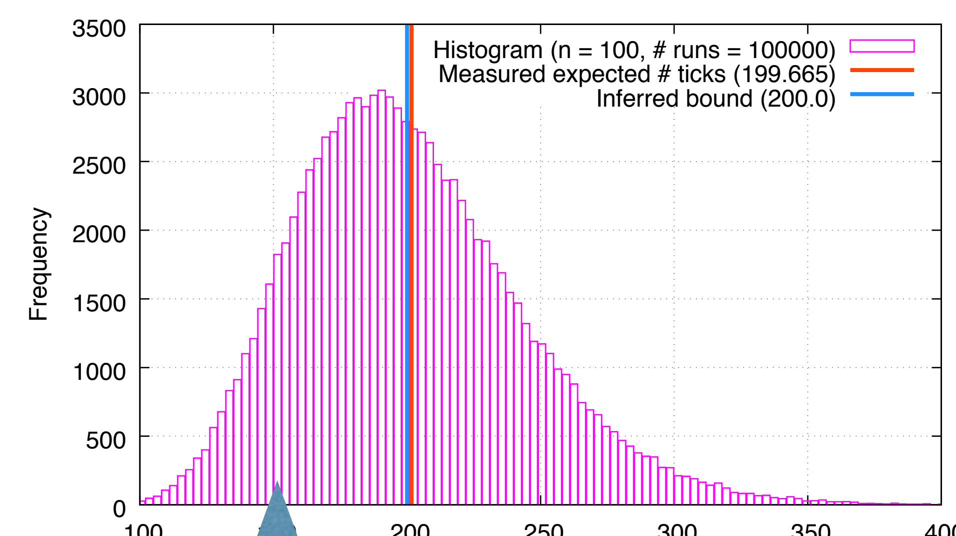


(2x - tick) is a martingale

Quantitative analysis with martingales

- Martingales are fundamental in mathematic and probability theory
- Consider a probabilistic program as a (infinite) sequence of random variables, the martingales can be used to reason about
  - Expected value of random variables
  - Tail-bound probability with Azuma-Hoeffding inequality
- With n = 100, martingale (2x - tick) is used to compute

$$\mathbb{E}[\text{tick}] = 2n = 200$$



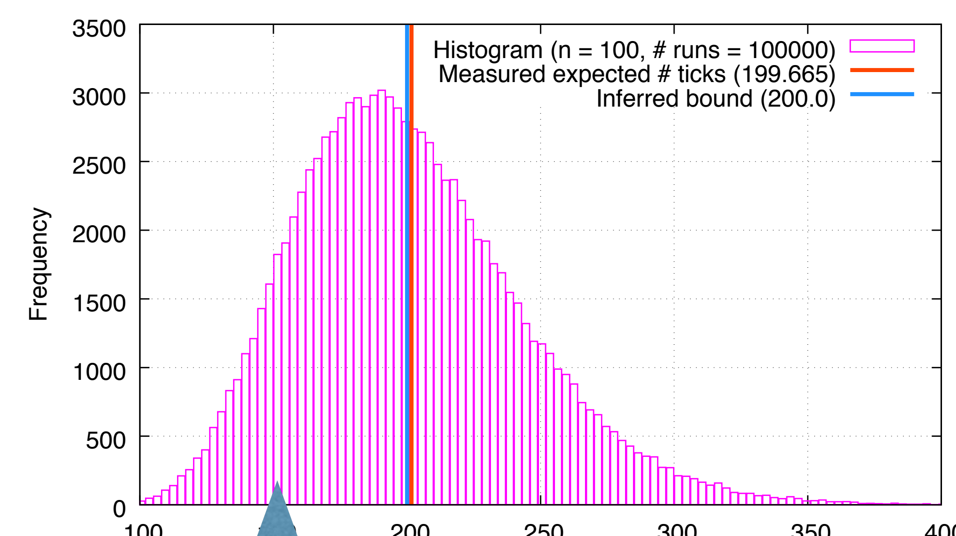
(2x - tick) is a martingale

Quantitative analysis with martingales

- Martingales are fundamental in mathematic and probability theory
- Consider a probabilistic program as a (infinite) sequence of random variables, the martingales can be used to reason about
  - Expected value of random variables
  - Tail-bound probability with Azuma-Hoeffding inequality
- With n = 100, martingale (2x - tick) is used to compute

$$\mathbb{E}[\text{tick}] = 2n = 200$$

$$\mathbb{P}(\text{tick} = 800 \wedge x < 100) \leq e^{-25}$$



(2x - tick) is a martingale

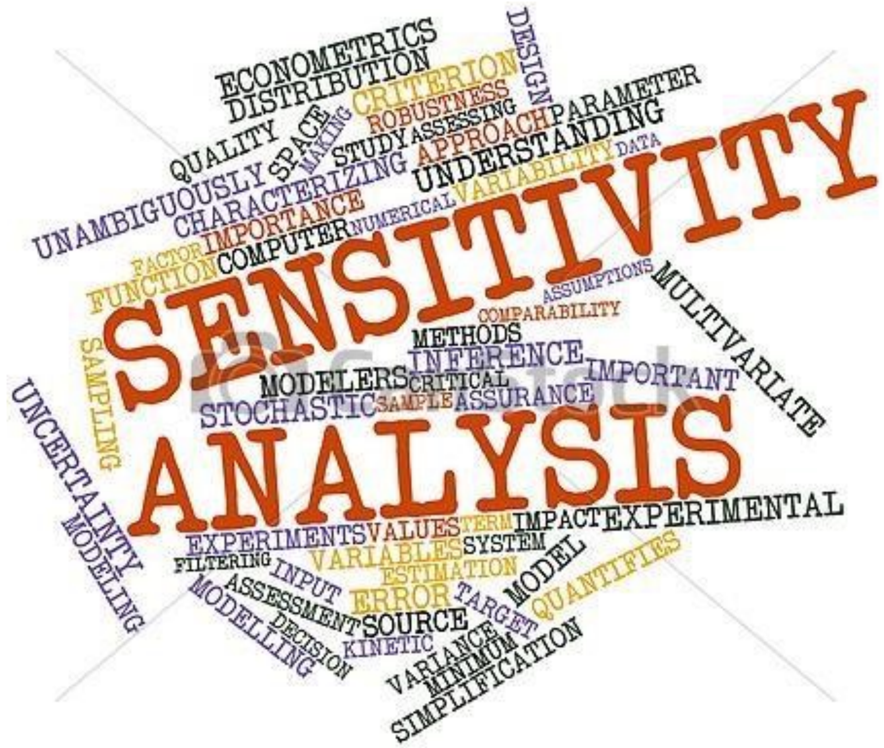
Automatic martingale generation

- Compiling probabilistic programs by automatically generating martingales
- Using the template-based approach
  - Fix martingale expressions as linear combinations of monomials with unknown coefficients at program points
  - Encode the martingale conditions between program points as linear constraints
  - Generate optimal martingales by solving the constraints with off-the-shelf LP solver

Probabilistic sensitivity

- Sensitivity describes how changes in inputs of a program can effect outputs

- Consider a probabilistic program such as a machine learning algorithm or statistical database



- Sensitivity can be used to reason about the stability of the algorithm or the privacy of the database queries

- Defined as difference between the expected outputs or between the output probabilities

Automatic sensitivity analysis

- Design a derivation system that checks automatically the expected and probabilistic sensitive programs
  - Adapt the previous work on automatic expected and tail-bound analyses to reason about the difference between outputs

- For example, expected sensitive w.r.t resource usage can be used to reason probabilistically about security against side-channel attacks
  - Every pair of same size inputs, the expected resource consumption is the same

Conclusions

- Quantitative analysis is critical for reliable and secure software
- Developed solutions for quantitative analysis
  - Formally verified compiler for safety-critical systems
  - Formal verification for timed and probabilistic systems
  - Type system for enforcing security properties
  - Expected bound analysis for probabilistic programming
- Static, automatic, efficient, and very precise
- Future directions for efficient and formally verified probabilistic compilation