S²E

A Platform for In-Vivo Multi-Path Analysis of Software Systems

Vitaly Chipounov, Volodymyr Kuznetsov,
George Candea

School of Computer & Communication Sciences

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Bug Finding

Bug Finding

```
int main(argc, argv)
{
  if (argc == 2) {
    printf("%c", *argv[2]);
    return -1;
  }

  return 0;
}
```

Bug Finding

```
int main(argc, argv)
{
  if (argc == 2) {
    printf("%c", *argv[2]);
    return -1;
  }

  return 0;
}
```

```
$ ./prog
```

Bug Finding

```
int main(argc, argv)
{
  if (argc == 2) {
    printf("%c", *argv[2]);
    return -1;
  }

  return 0;
}
```

```
$ ./prog

$ ./prog p1
Segmentation fault
```

Bug Finding

```
int main(argc, argv)
{
  if (argc == 2) {
    printf("%c", *argv[2]);
    return -1;
  }

  return 0;
}
```

```
$ ./prog

$ ./prog p1
Segmentation fault


$ valgrind ./prog p1
Invalid read of size 1
main (prog.c:10)
```

Performance Profiling

Performance Profiling

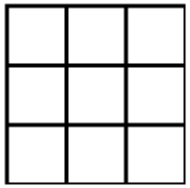
```
int matrixSum(matrix_t m)
{
  int sum=0;

  for(i = 0; i < m.w; i++)
   for(j = 0; j < m.h; j++)
     sum +=  m[i][j];

  return sum;
}
```

Performance Profiling

```
int matrixSum(matrix_t m)
{
  int sum=0;

  for(i = 0; i < m.w; i++)
   for(j = 0; j < m.h; j++)
     sum +=  m[i][j];

  return sum;
}
```

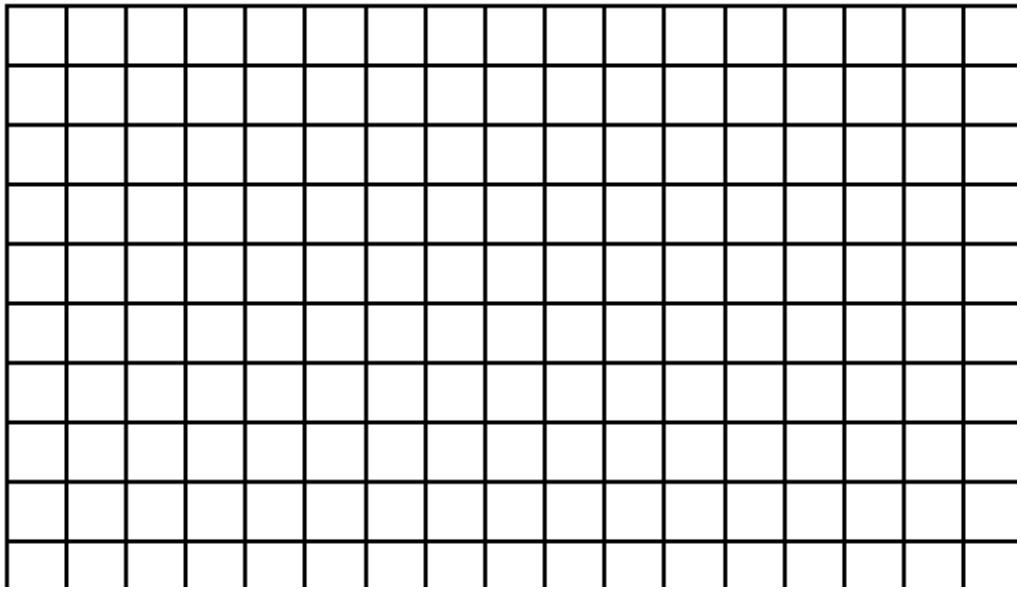
Performance Profiling

```
int matrixSum(matrix_t m)
{
  int sum=0;

  for(i = 0; i < m.w; i++)
   for(j = 0; j < m.h; j++)
     sum +=  m[i][j];

  return sum;
}
```



Performance Profiling

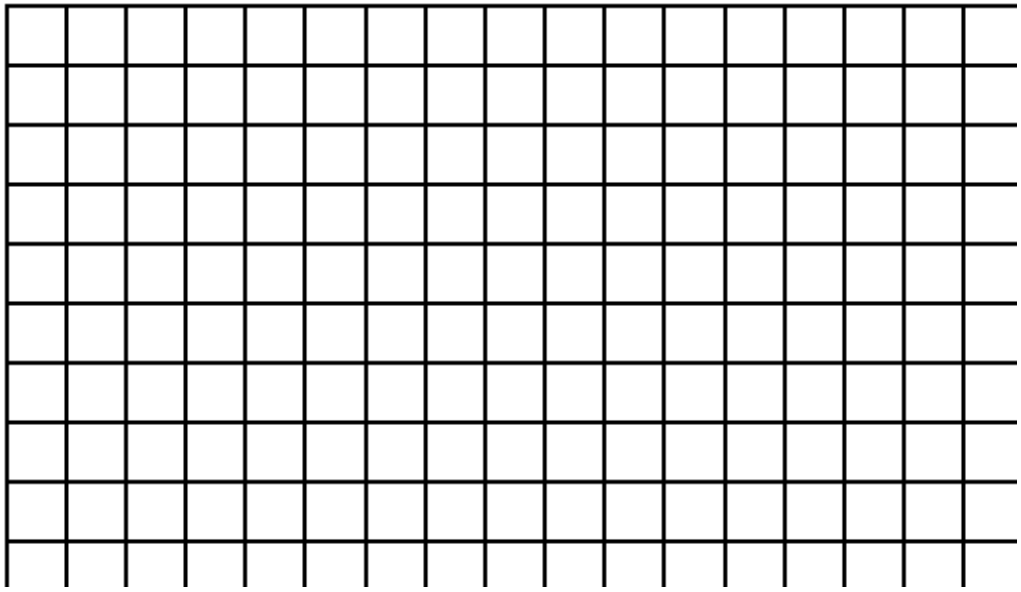
```
int matrixSum(matrix_t m)
{
  int sum=0;

  for(i = 0; i < m.w; i++)
   for(j = 0; j < m.h; j++)
     sum +=  m[i][j];

  return sum;
}
```



Performance Profiling

```
int matrixSum(matrix_t m)
{
  int sum=0;

  for(i = 0; i < m.w; i++)
   for(j = 0; j < m.h; j++)
     sum +=  m[i][j];

  return sum;
}
```

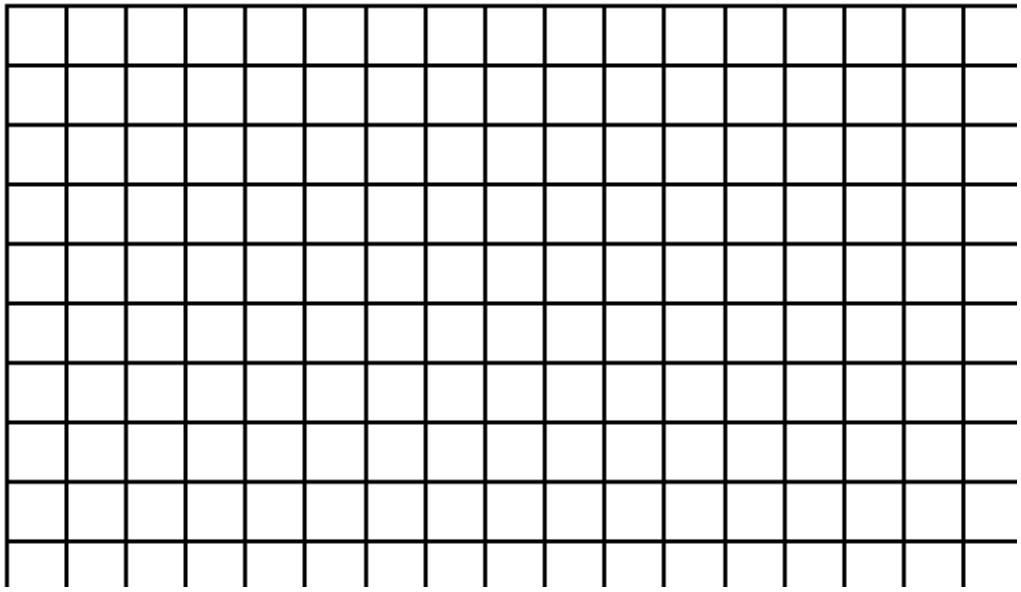
OProfile



Performance Profiling

```
int matrixSum(matrix_t m)
{
  int sum=0;

  for(i = 0; i < m.w; i++)
   for(j = 0; j < m.h; j++)
     sum += m[i][j];

  return sum;
}
```

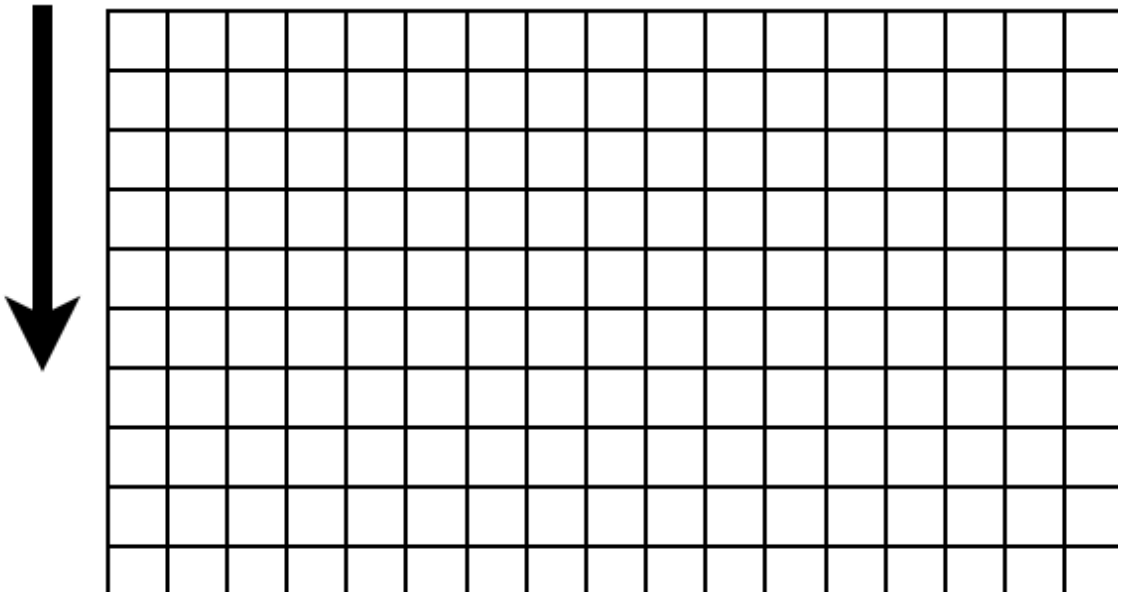
OProfile



Performance Profiling

```
int matrixSum(matrix_t m)
{
  int sum=0;

  for(i = 0; i < m.w; i++)
   for(j = 0; j < m.h; j++)
     sum += m[i][j];

  return sum;
}
```

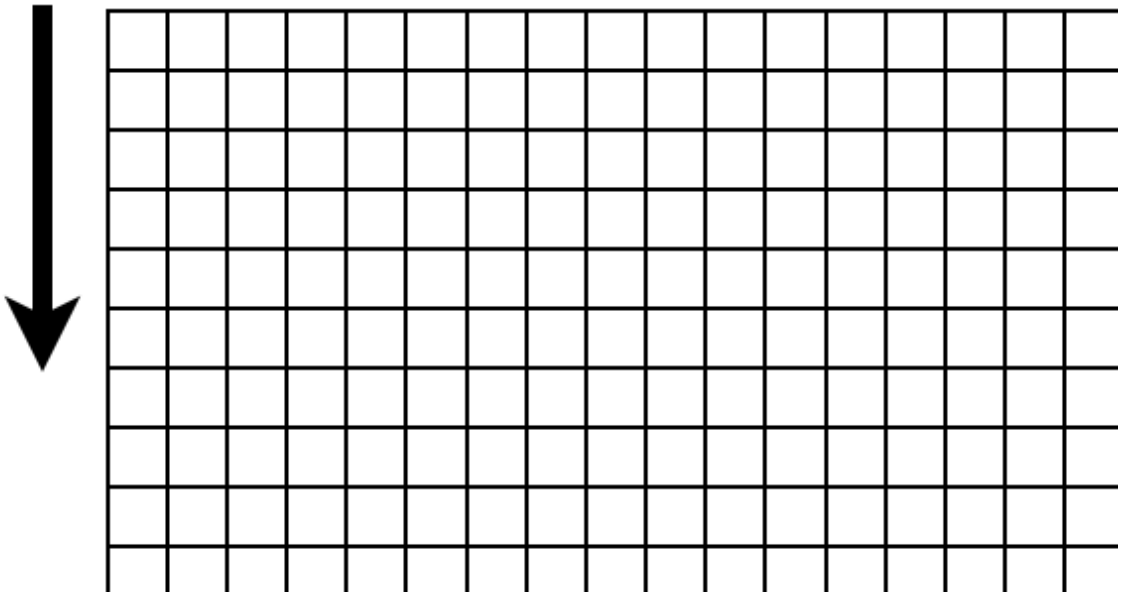
OProfile



Performance Profiling

```
int matrixSum(matrix_t m)
{
  int sum=0;

  for(i = 0; i < m.w; i++)
   for(j = 0; j < m.h; j++)
     sum += m[i][j];

  return sum;
}
```

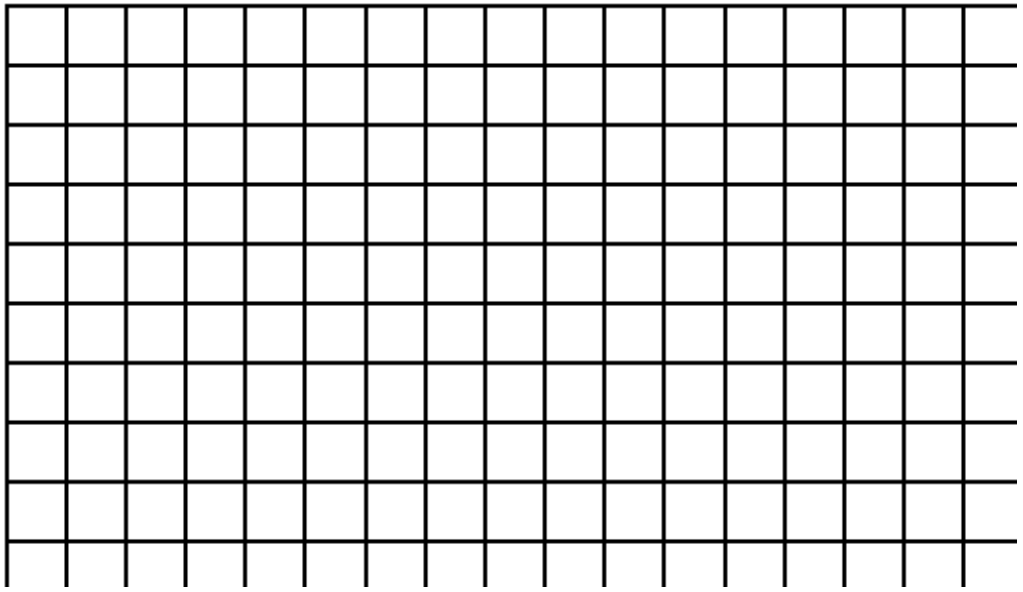
OProfile



Performance Profiling

```
int matrixSum(matrix_t m)
{
  int sum=0;

  for(i = 0; i < m.w; i++)
   for(j = 0; j < m.h; j++)
     sum += m[i][j];

  return sum;
}
```

OProfile



Analyses

- Bug finding
- Performance profiling
- Verification/Certification
- Security analysis
- ...

Analyses

- Bug finding
- Performance profiling
- Verification/Certification
- Security analysis
- ...

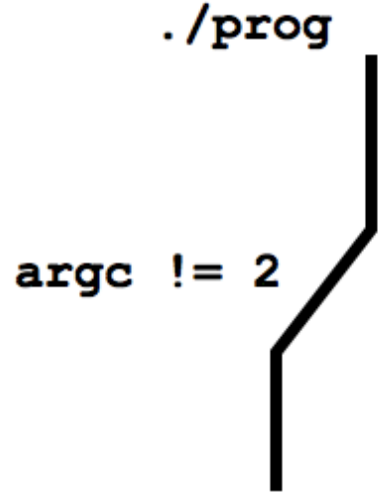
Check properties on execution paths

Bug Finding

```
int main(argc, argv)
{
  if (argc == 2) {
    printf("%c", *argv[2]);
    return -1;
  }

  return 0;
}
```

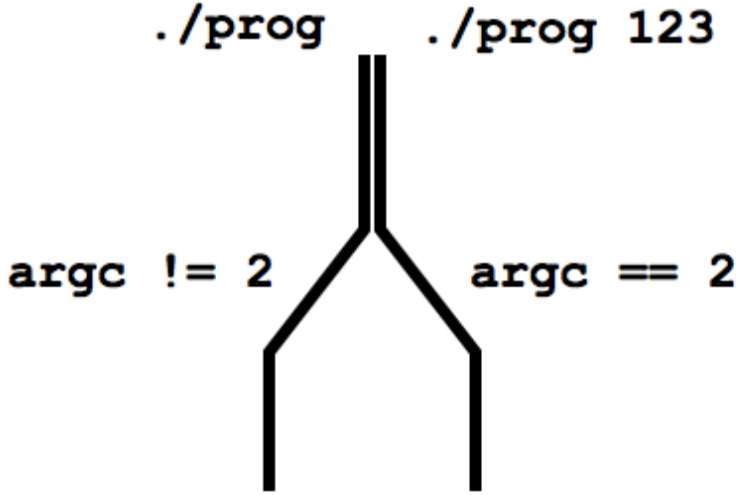
Bug Finding

```
int main(argc, argv)
{
  if (argc == 2) {
    printf("%c", *argv[2]);
    return -1;
  }

  return 0;
}
```



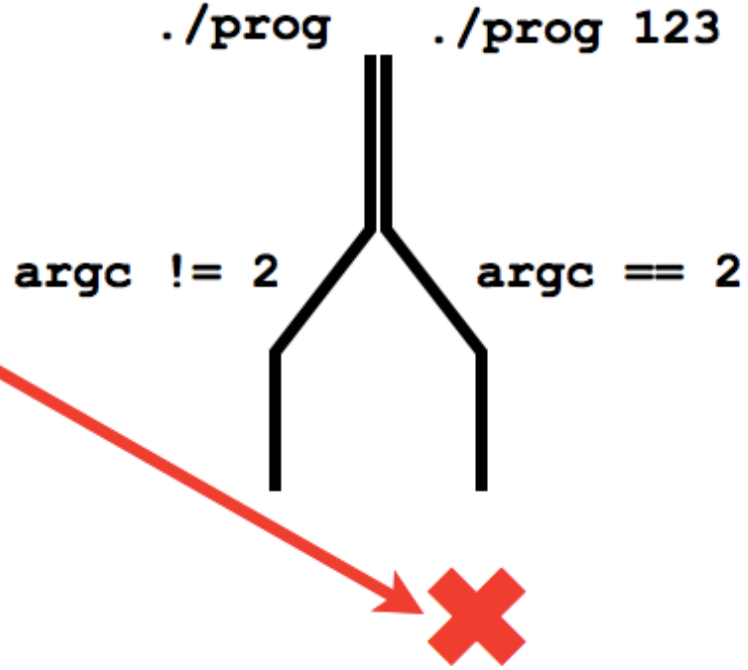
Bug Finding

```
int main(argc, argv)
{
  if (argc == 2) {
    printf("%c", *argv[2]);
    return -1;
  }

  return 0;
}
```



Bug Finding

```
int main(argc, argv)
{
  if (argc == 2) {
    printf("%c", *argv[2]);
    return -1;
  }

  return 0;
}
```

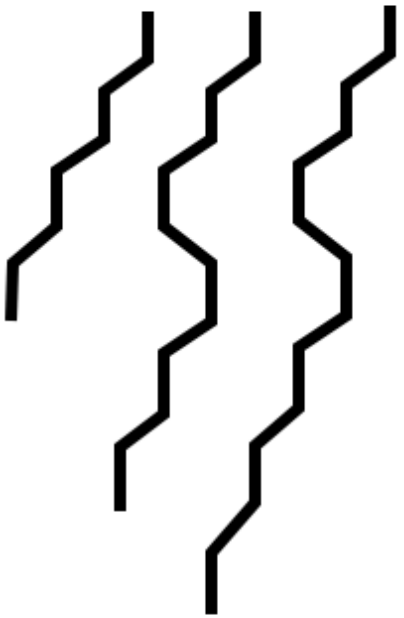


Performance Profiling

```
int matrixSum(matrix_t m)
{
  int sum=0;

  for(i = 0; i < m.w; i++)
   for(j = 0; j < m.h; j++)
     sum +=  m[i][j];

  return sum;
}
```

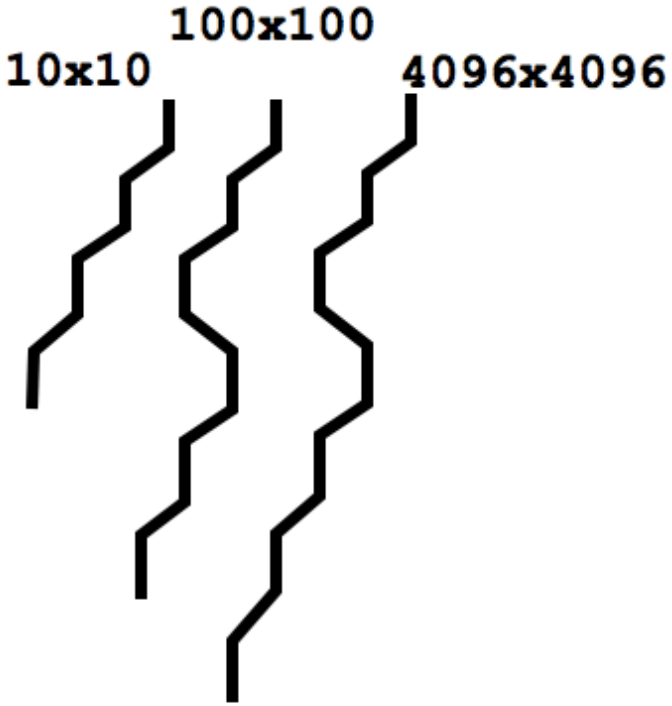
Performance Profiling

```
int matrixSum(matrix_t m)
{
  int sum=0;

  for(i = 0; i < m.w; i++)
   for(j = 0; j < m.h; j++)
     sum +=  m[i][j];

  return sum;
}
```



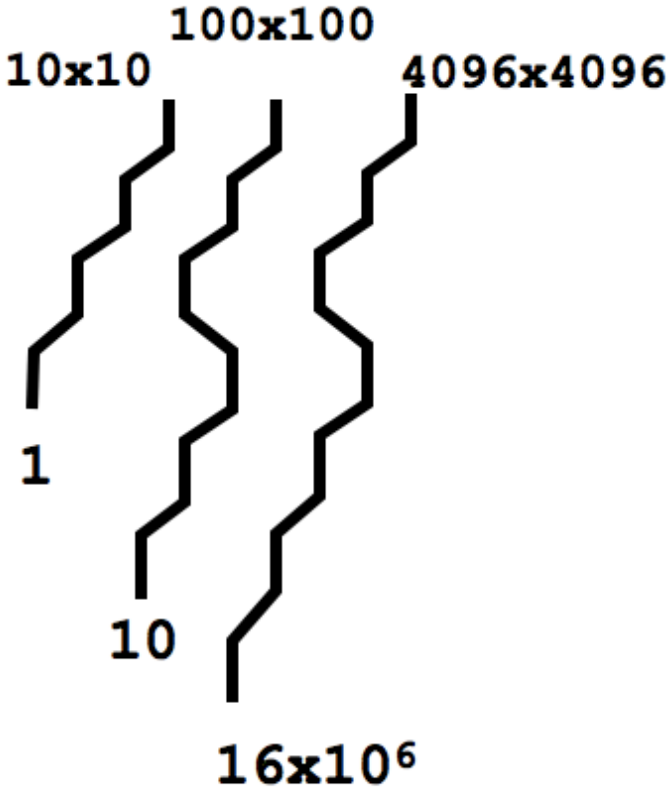
Performance Profiling

```
int matrixSum(matrix_t m)
{
  int sum=0;

  for(i = 0; i < m.w; i++)
   for(j = 0; j < m.h; j++)
     sum +=  m[i][j];

  return sum;
}
```



Performance Profiling

```
int matrixSum(matrix_t m)
{
  int sum=0;

  for(i = 0; i < m.w; i++)
   for(j = 0; j < m.h; j++)
     sum +=  m[i][j];

  return sum;
}
```



Cache misses

Systematic Path Enumeration

- Automatically finding the right paths
  To detect bugs
  To expose performance issues
  To ...

In-Vivo Multi-Path Analysis

Analyze a *living* system, for maximum realism

http://www.wtcc.wur.nl/NR/rdonlyres/E8C658AD-8303-41D9-A141-E5413DB9E3BD/41107/invitro.jpg

In-Vivo Multi-Path Analysis

Analyze a *living* system, for maximum realism

In Vivo

http://www.wtcc.wur.nl/NR/rdonlyres/E8C658AD-8303-41D9-A141-E5413DB9E3BD/41107/invitro.jpg

In-Vivo Multi-Path Analysis

Analyze a *living* system, for maximum realism

In Vitro

In Vivo

http://www.wtcc.wur.nl/NR/rdonlyres/E8C658AD-8303-41D9-A141-E5413DB9E3BD/41107/invitro.jpg

Challenge

Challenge

$2^{\text{system size}}$ paths

Today's Approaches

Today's Approaches

Analyze only some of the paths
Introduces false negatives (FNs)

Today's Approaches

Analyze only some of the paths
Introduces false negatives (FNs)

Abstract away parts of the paths
Introduces false positives (FPs)

Outline

- Theory
  Execution consistency models
- System
  S²E: Platform for in-vivo multi-path analysis
- Results
  Using S²E in practice

http://s2e.epfl.ch

Outline

- Theory
  Execution consistency models
- System
  S²E: Platform for in-vivo multi-path analysis
- Results
  Using S²E in practice

http://s2e.epfl.ch

Execution Consistency Models

Execution Consistency Models

- Specify the set of paths to be analyzed

Execution Consistency Models

- Specify the set of paths to be analyzed
- Principled FPs/FNs trade-offs

Execution Consistency Models

- Specify the set of paths to be analyzed
- Principled FPs/FNs trade-offs
- Remember memory consistency models ?

Consistency Models in S2E

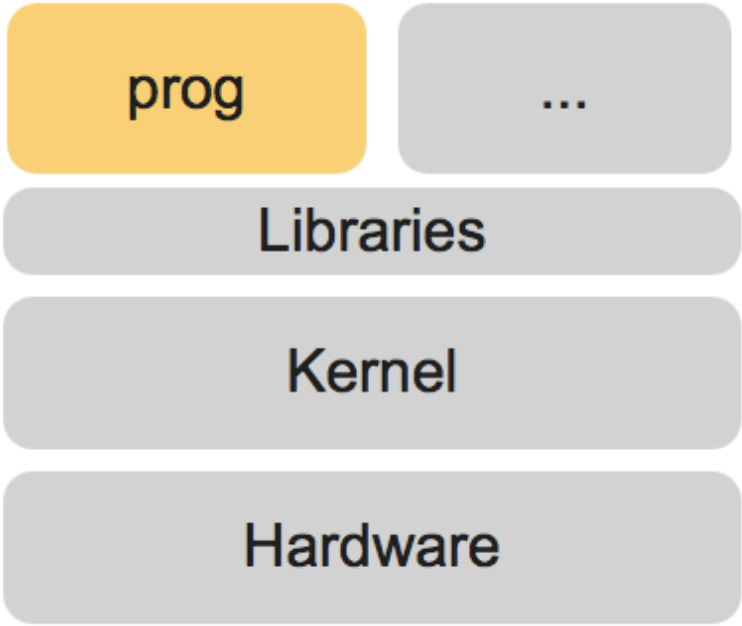
Consistency Models in S2E

```
int main(argc, argv) {

  if (argc == 0) {
     ...
  }

  p = malloc(...);

  if (p == NULL) {
     ...
  }
  ...
}
```

prog

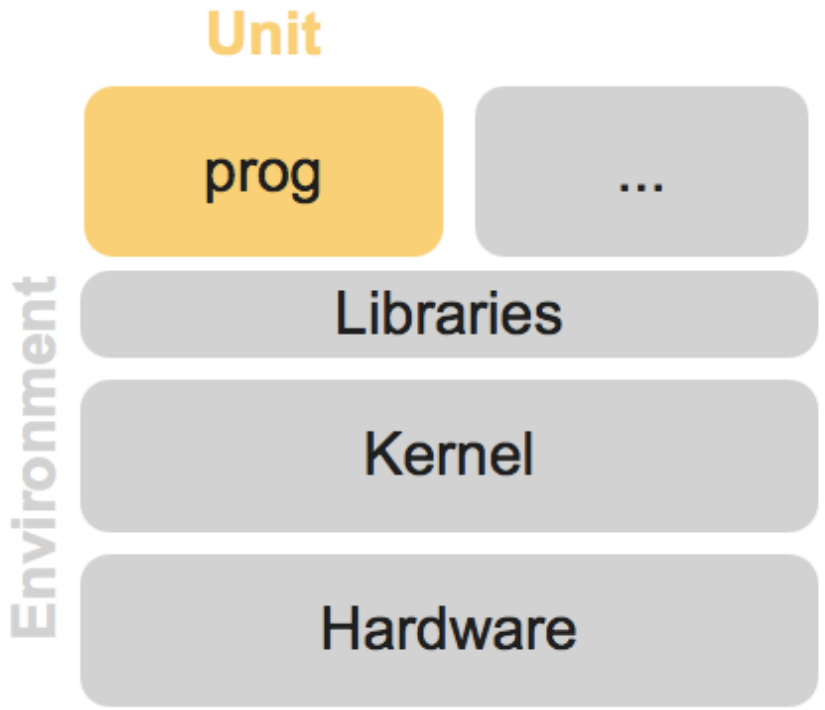
Consistency Models in S2E

```
int main(argc, argv) {

  if (argc == 0) {
     ...
  }

  p = malloc(...);

  if (p == NULL) {
     ...
  }
  ...
}
```



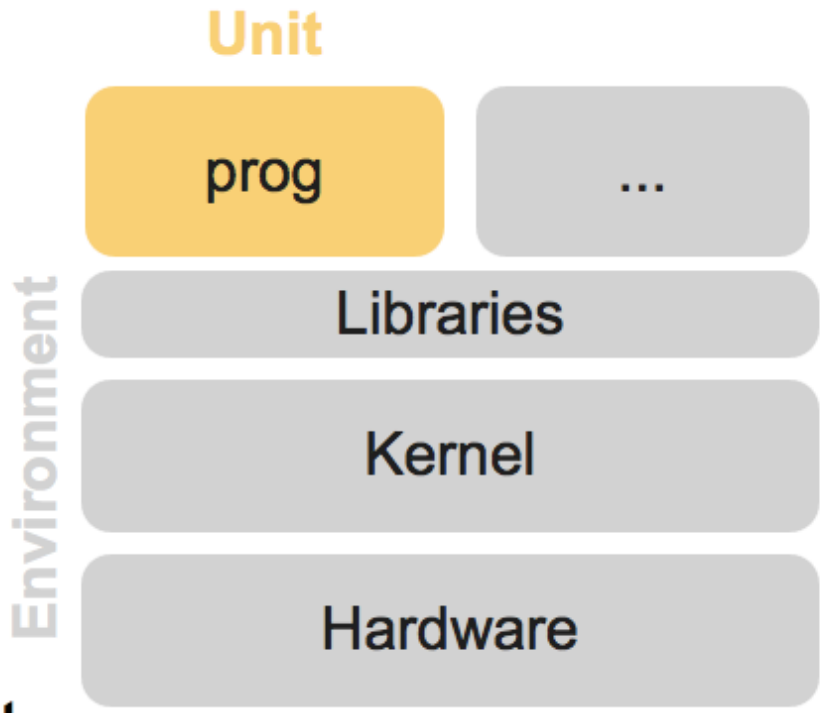
Consistency Models in S2E

```
int main(argc, argv) {

  if (argc == 0) {
      ...
  }

  p = malloc(...);

  if (p == NULL) {
      ...
  }
  ...
}
```



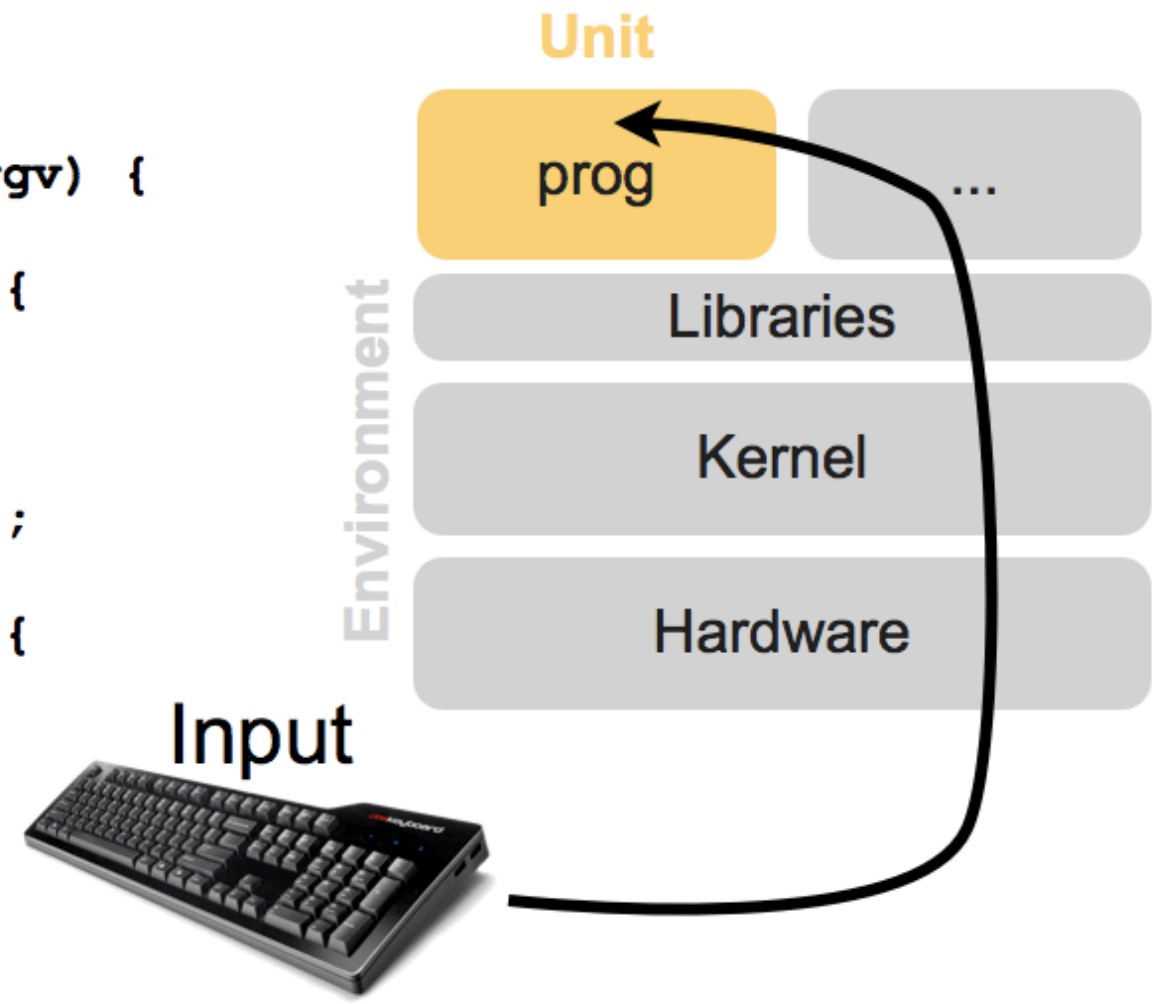
Consistency Models in S2E

```
int main(argc, argv) {

  if (argc == 0) {
     ...
  }

  p = malloc(...);

  if (p == NULL) {
     ...
  }
  ...
}
```



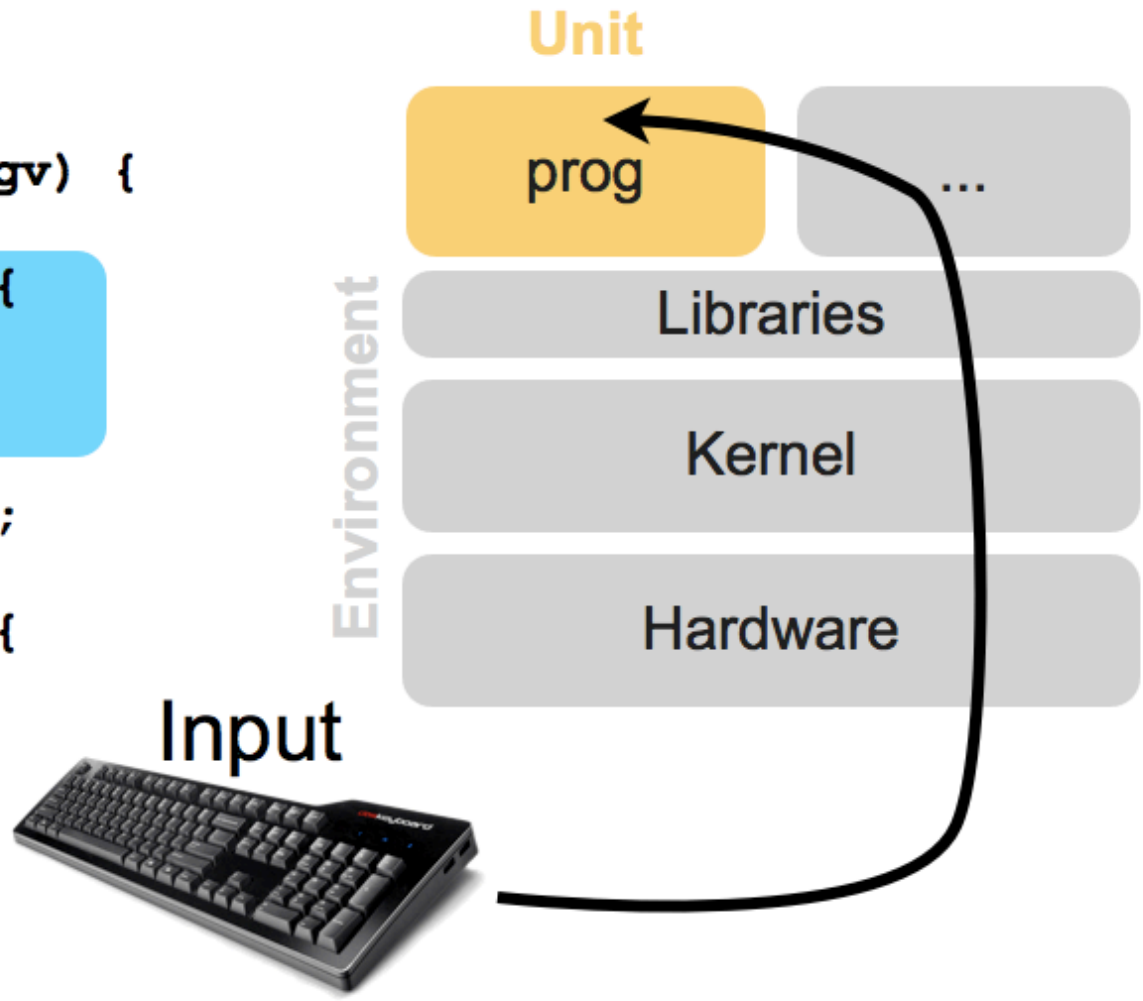
Consistency Models in S2E

```
int main(argc, argv) {

  if (argc == 0) {
      ...
  }

  p = malloc(...);

  if (p == NULL) {
      ...
  }
  ...
}
```



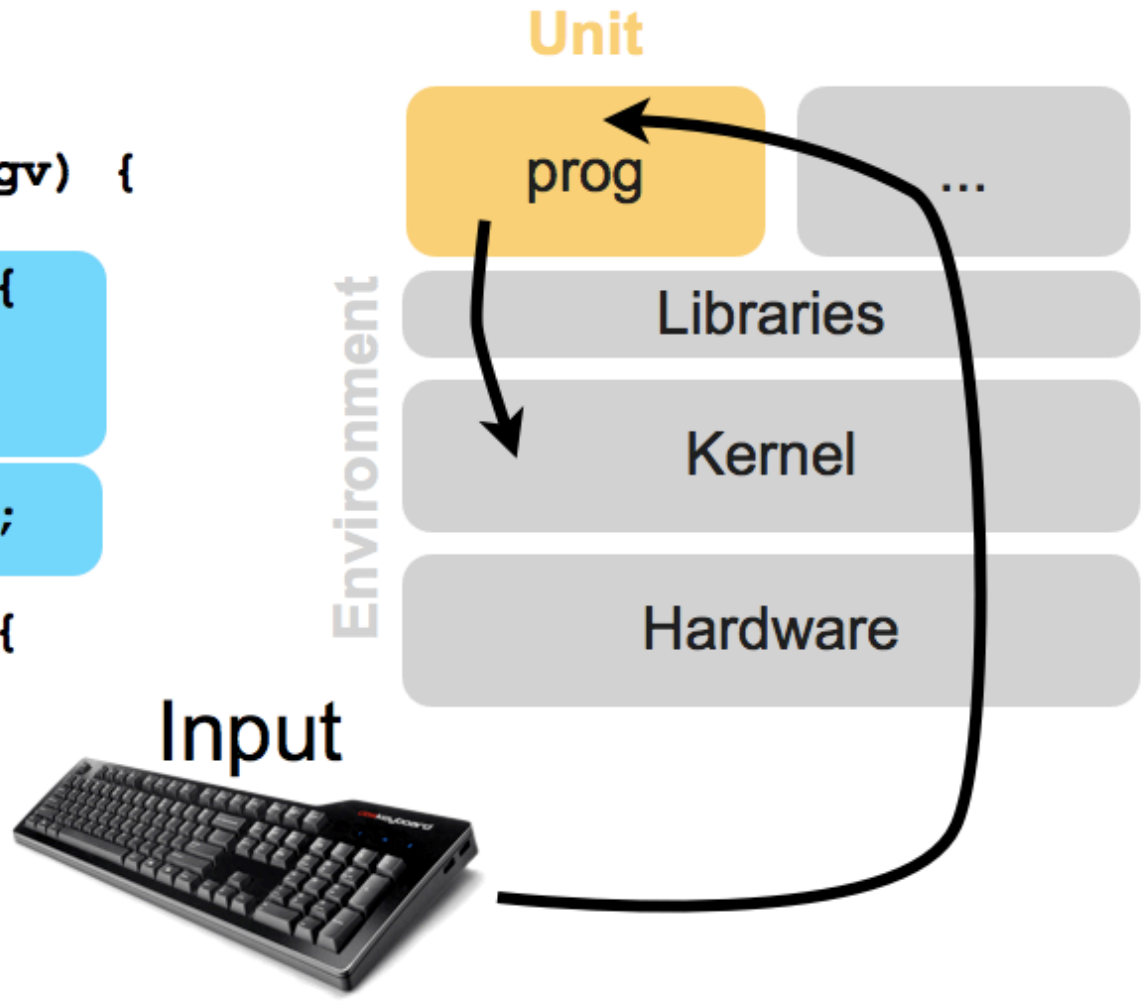
Consistency Models in S2E

```
int main(argc, argv) {
  if (argc == 0) {
      ...
  }
  p = malloc(...);
  if (p == NULL) {
      ...
  }
  ...
}
```



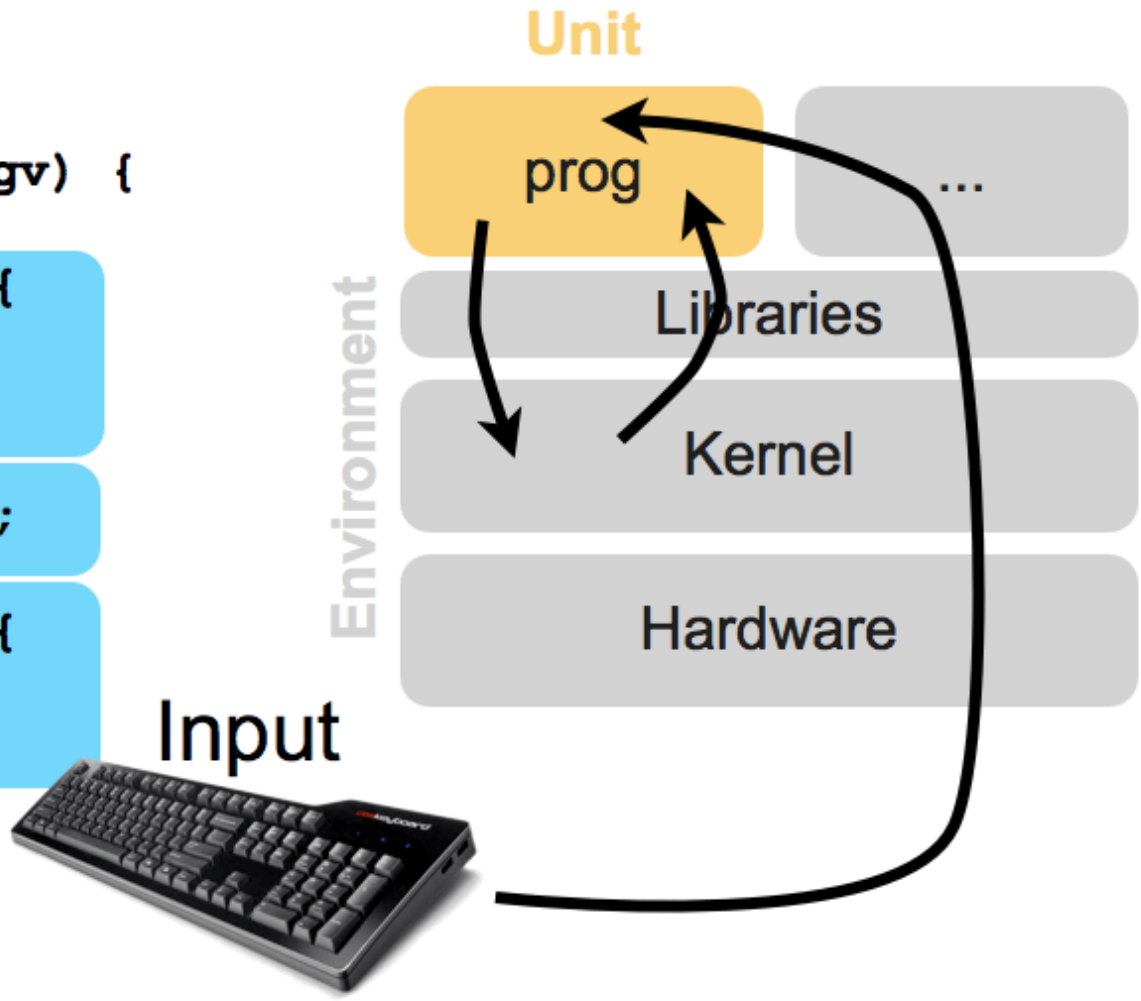
Consistency Models in S2E

```
int main(argc, argv) {
  if (argc == 0) {
      ...
  }
  p = malloc(...);
  if (p == NULL) {
      ...
  }
  ...
}
```



Consistency Models in S2E

```
int main(argc, argv) {
  if (argc == 0) {
      ...
  }
  p = malloc(...);
  if (p == NULL) {
      ...
  }
  ...
}
```



SC-SE

Strictly Consistent *System*-Level Execution

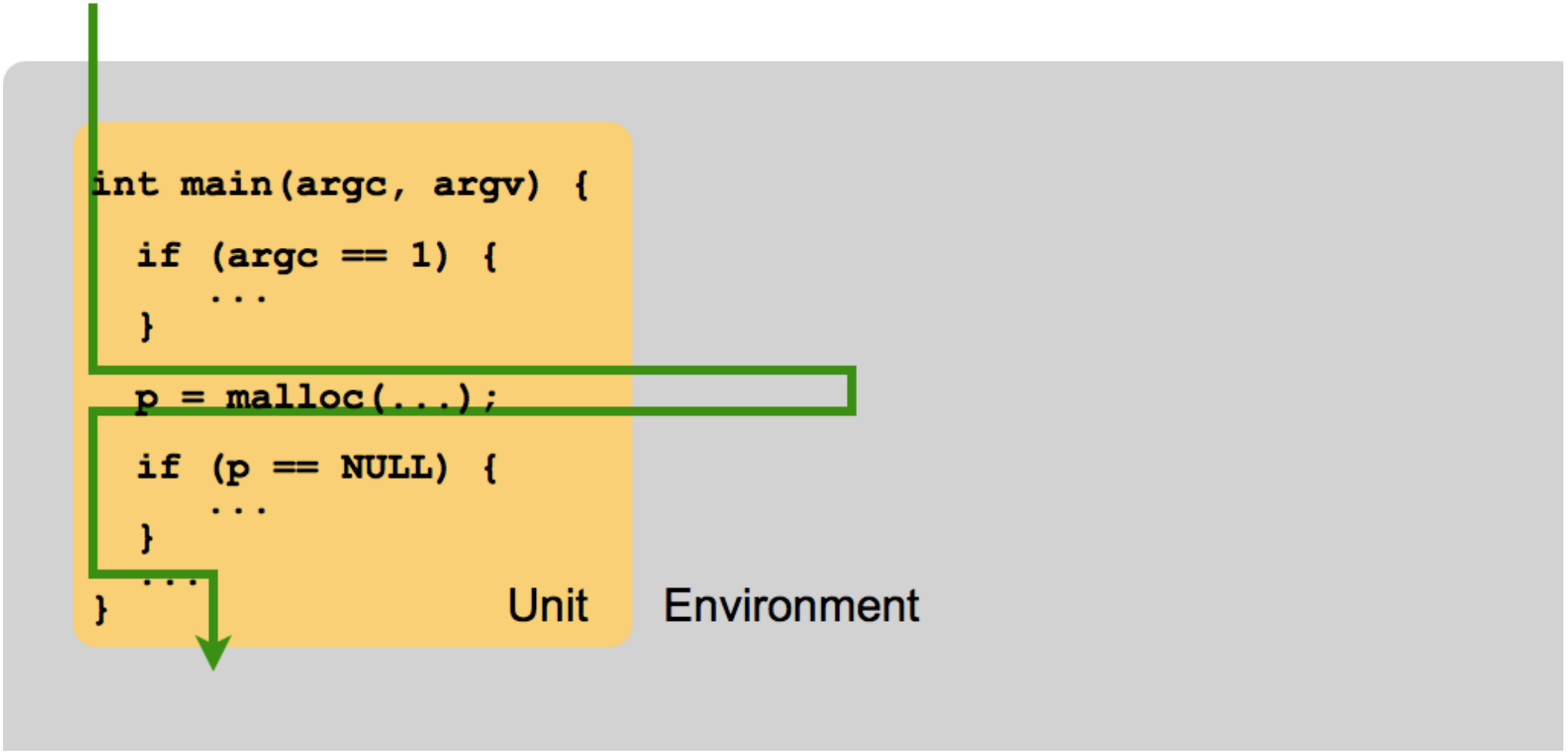
```
int main(argc, argv) {
  if (argc == 1) {
     ...
  }
  p = malloc(...);
  if (p == NULL) {
     ...
  }
  ...
}
```

Unit

Environment

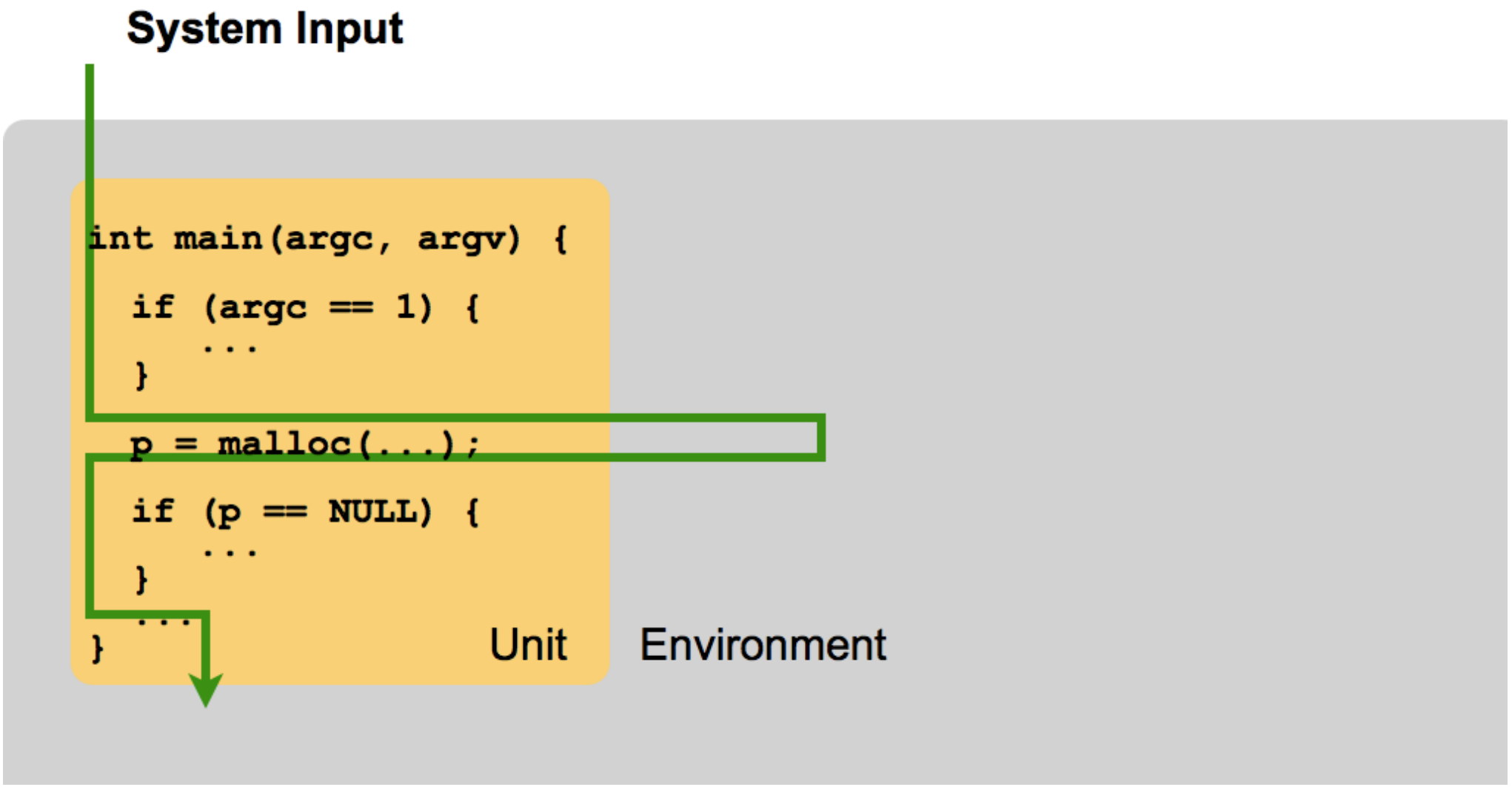
SC-SE

Strictly Consistent *System*-Level Execution



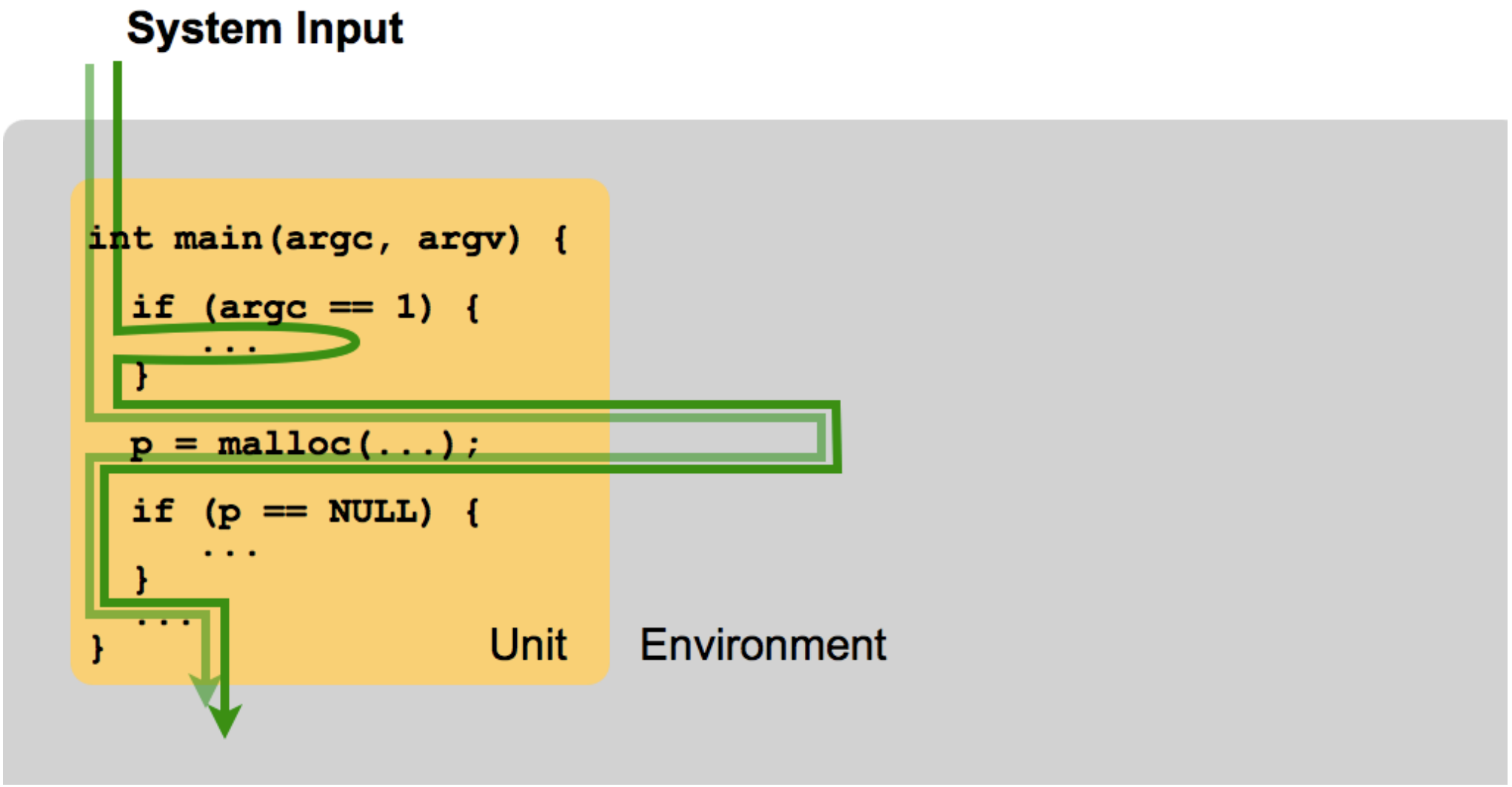
SC-SE

Strictly Consistent *System*-Level Execution



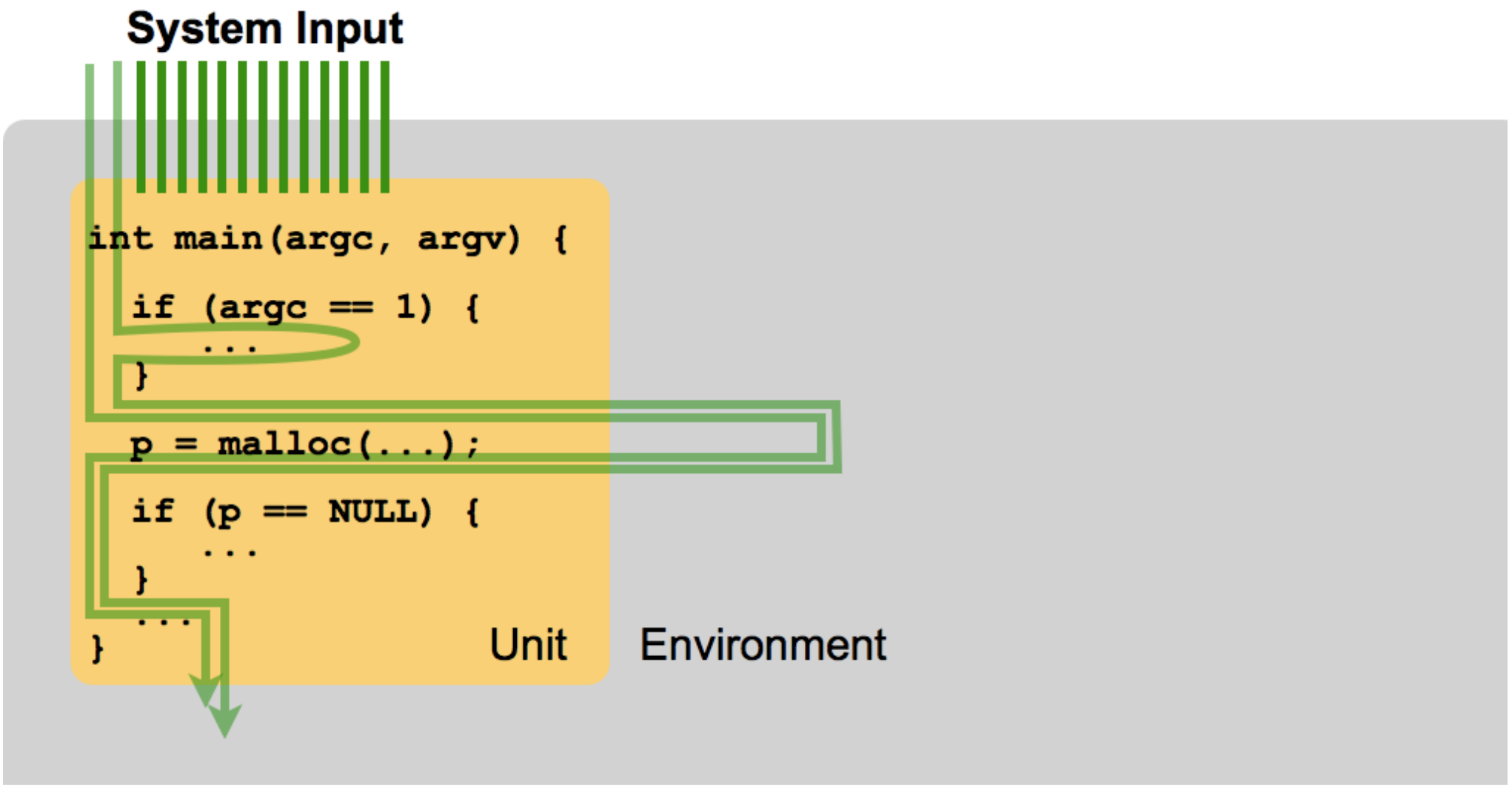
SC-SE

Strictly Consistent *System*-Level Execution



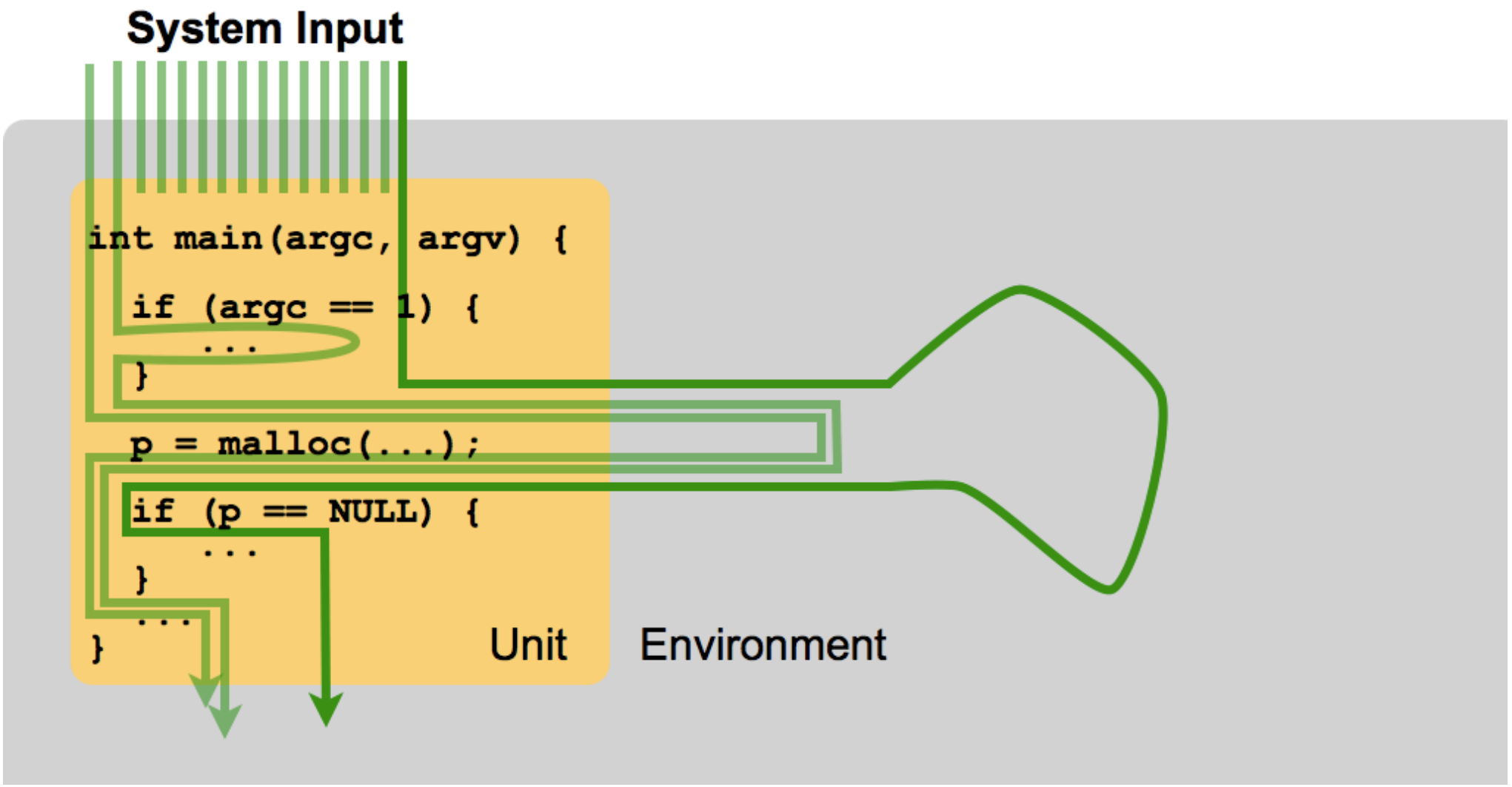
SC-SE

Strictly Consistent *System*-Level Execution



SC-SE

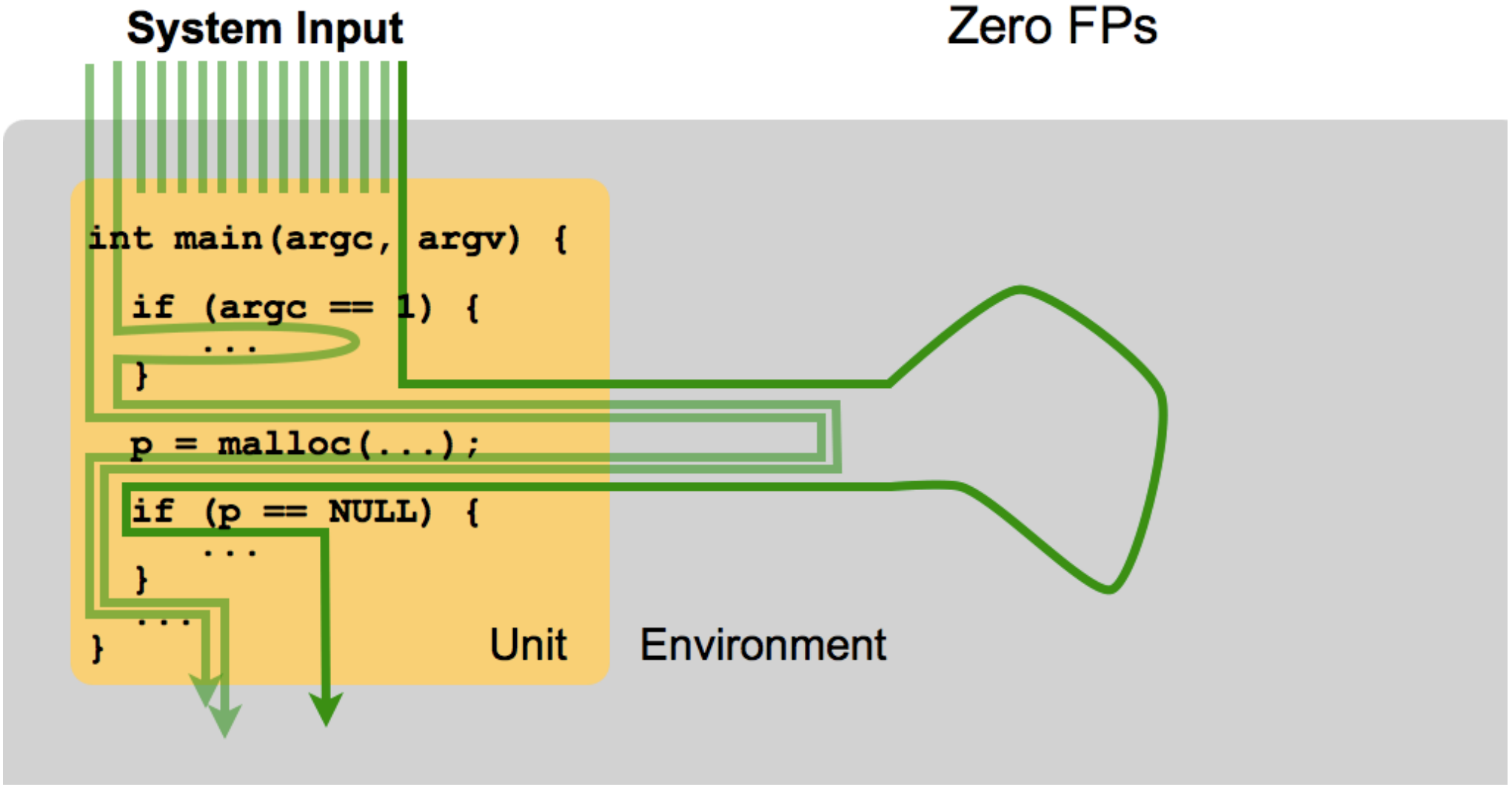
Strictly Consistent *System*-Level Execution



SC-SE

Strictly Consistent *System*-Level Execution

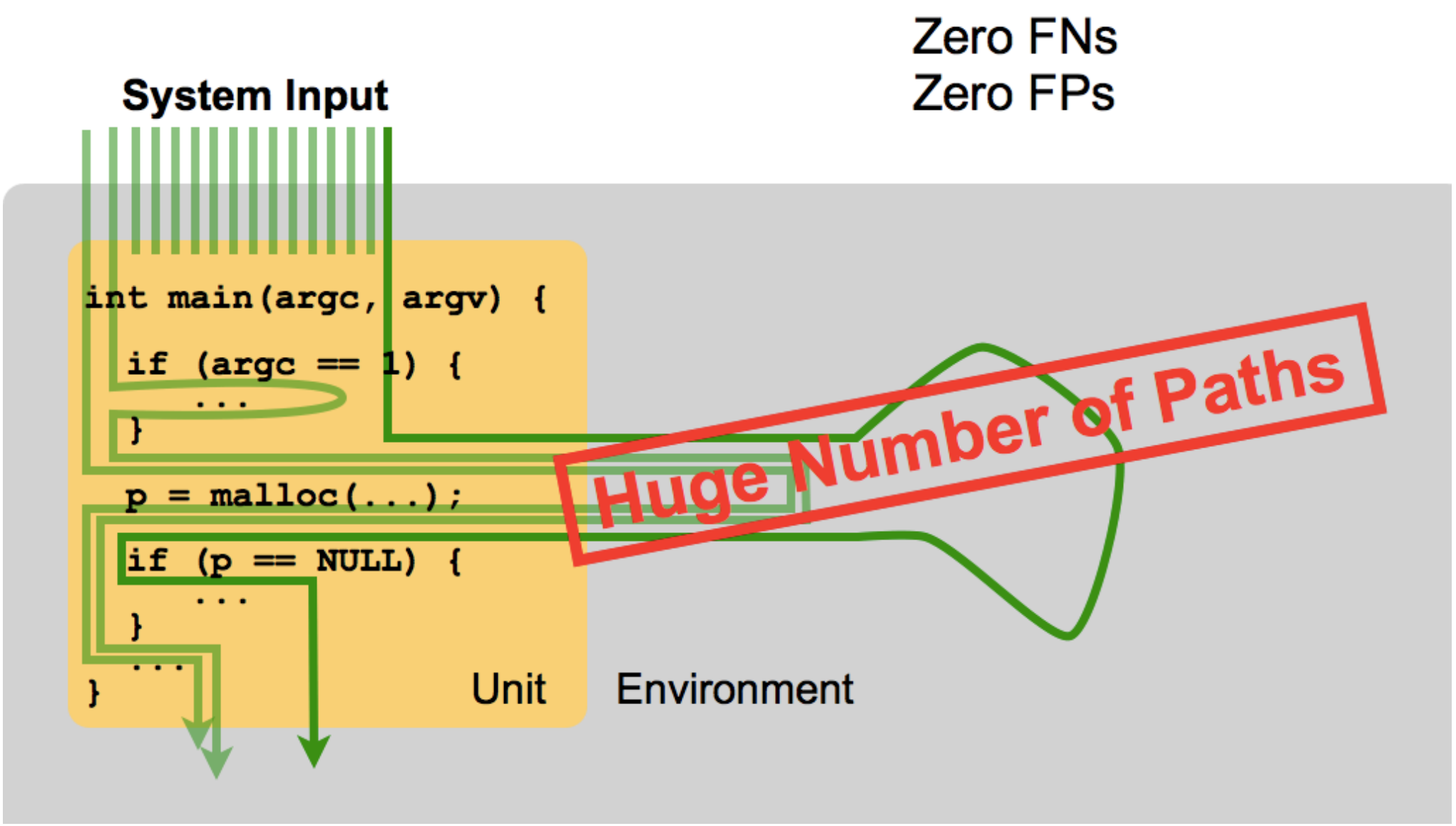
Zero FNs
Zero FPs



SC-SE

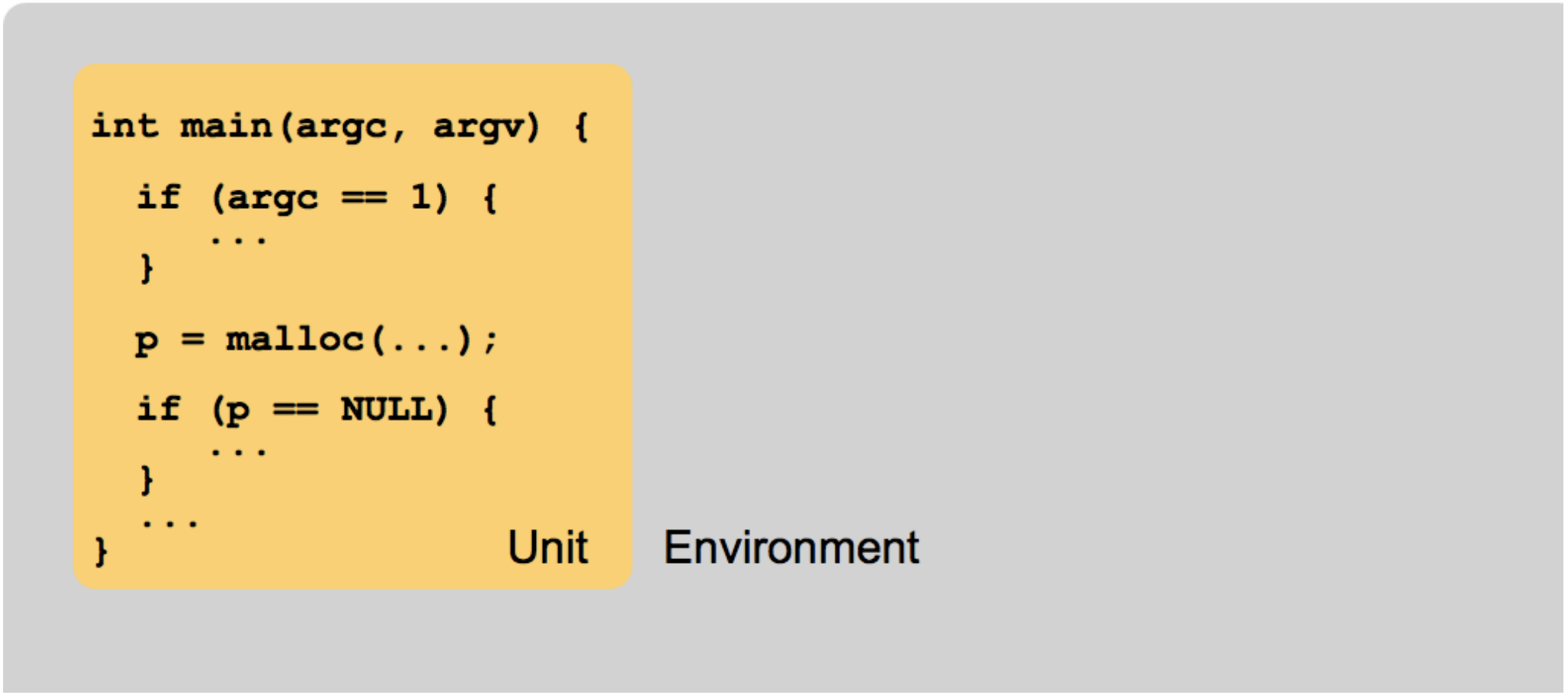
Strictly Consistent *System*-Level Execution

Zero FNs
Zero FPs



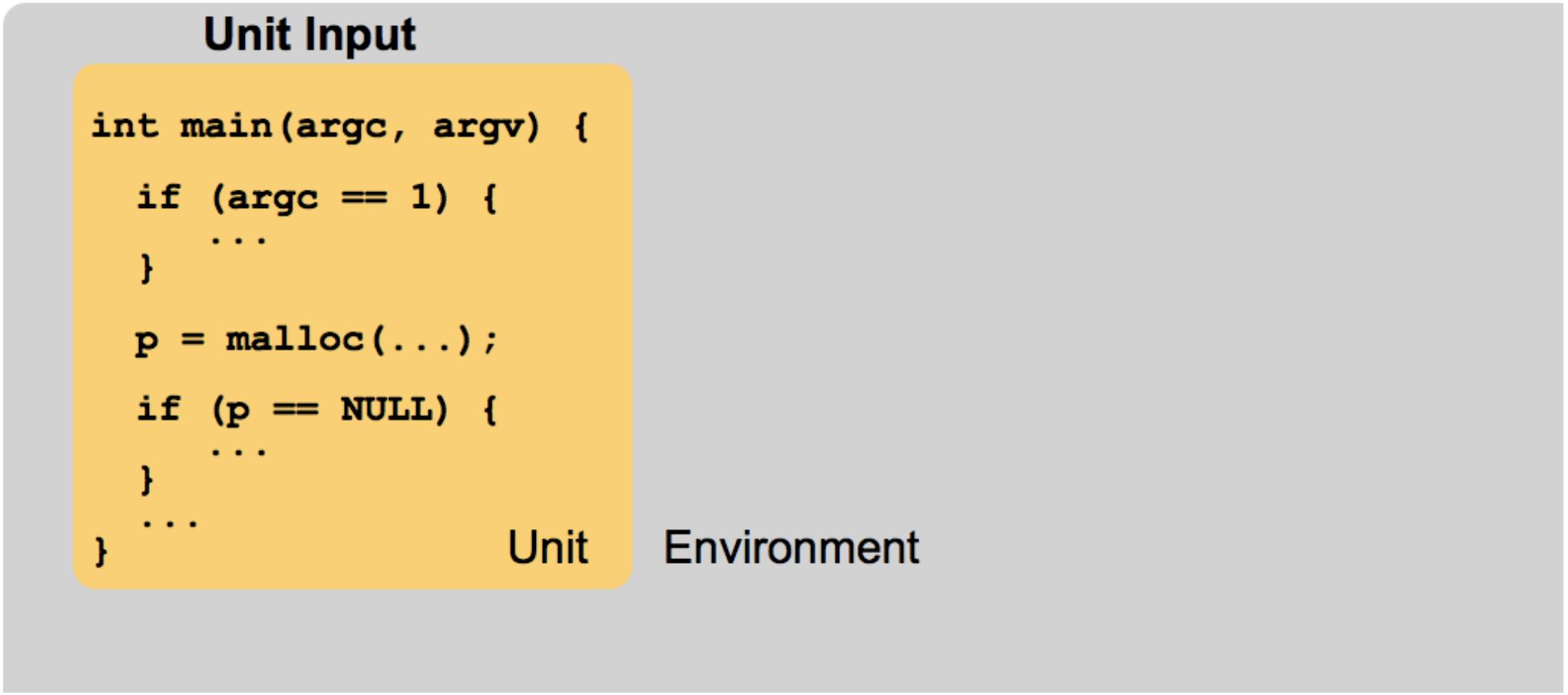
SC-UE

Strictly Consistent *Unit-Level* Execution



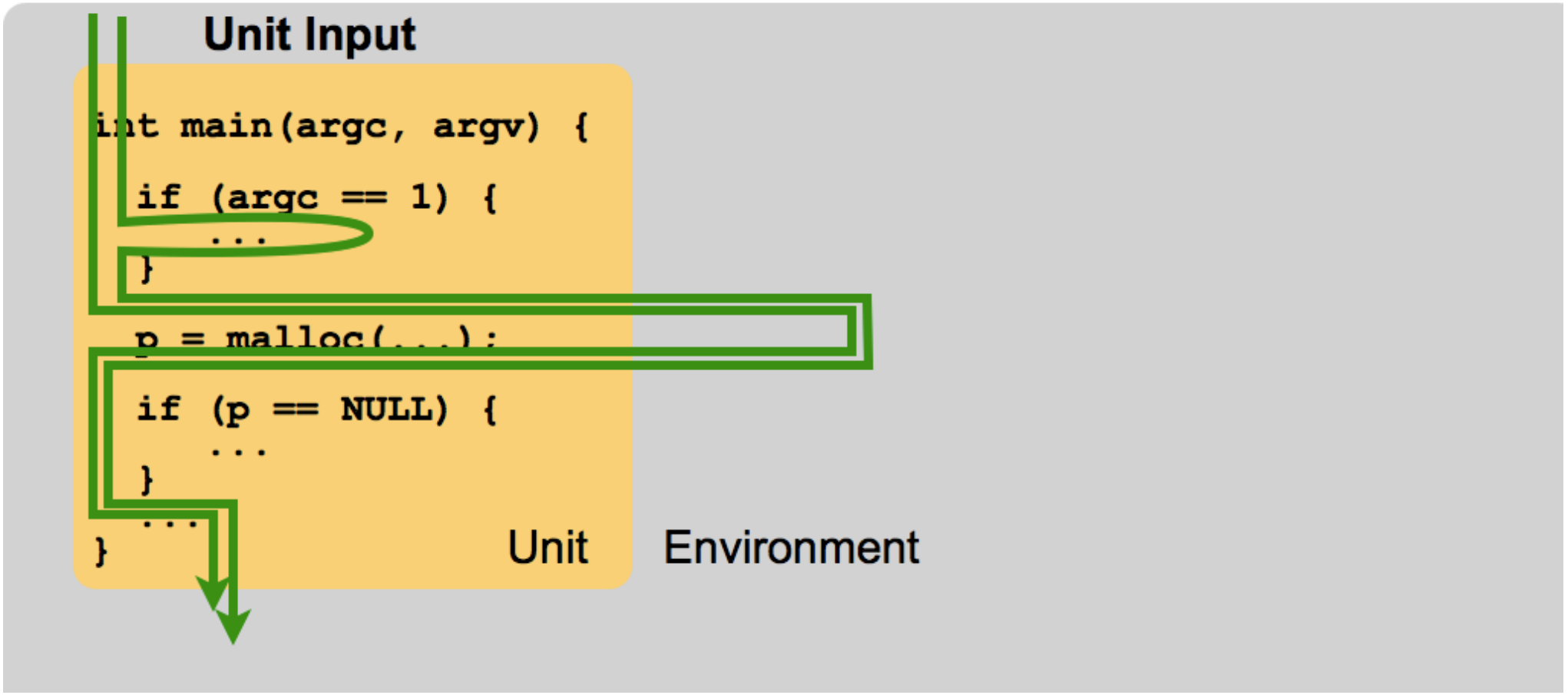
SC-UE

Strictly Consistent *Unit-Level* Execution



SC-UE

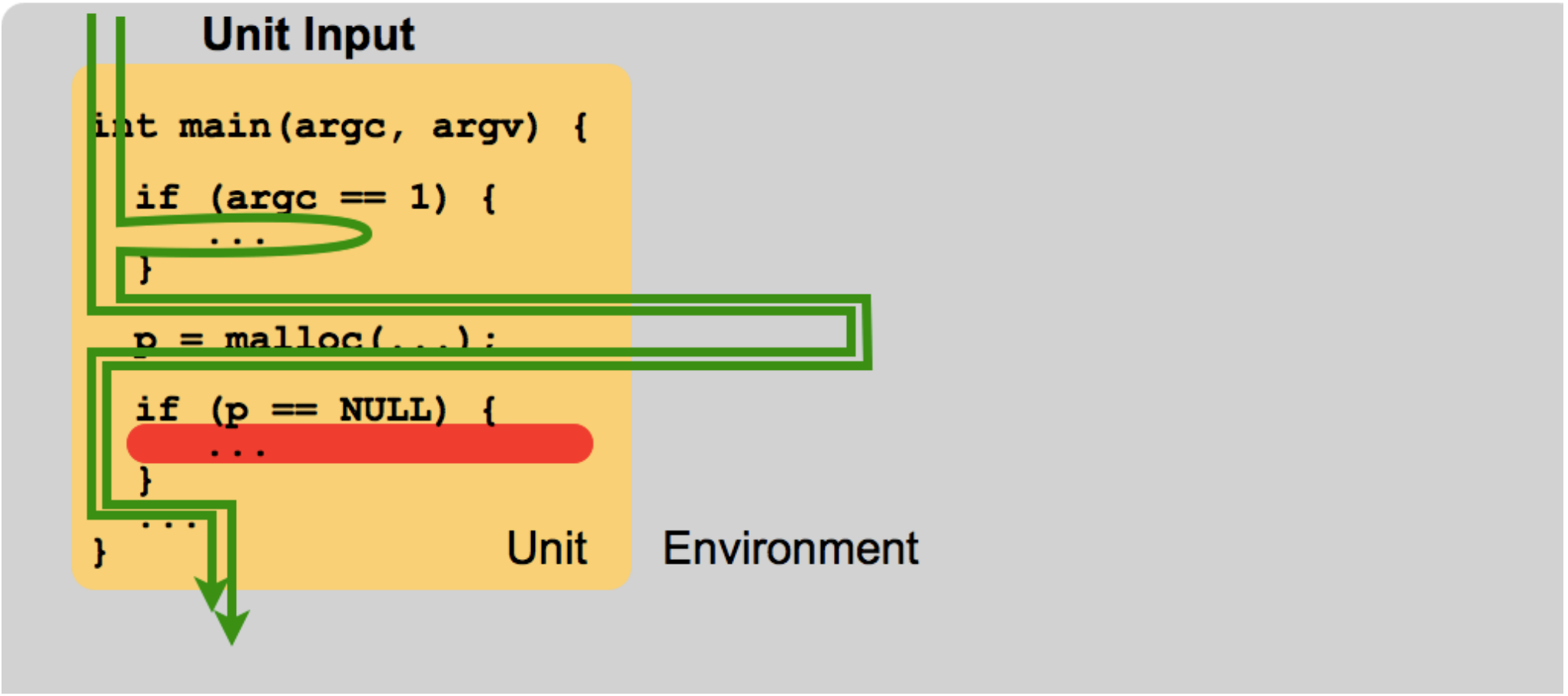
Strictly Consistent *Unit-Level* Execution



SC-UE

Strictly Consistent *Unit-Level* Execution

Presence of FNs

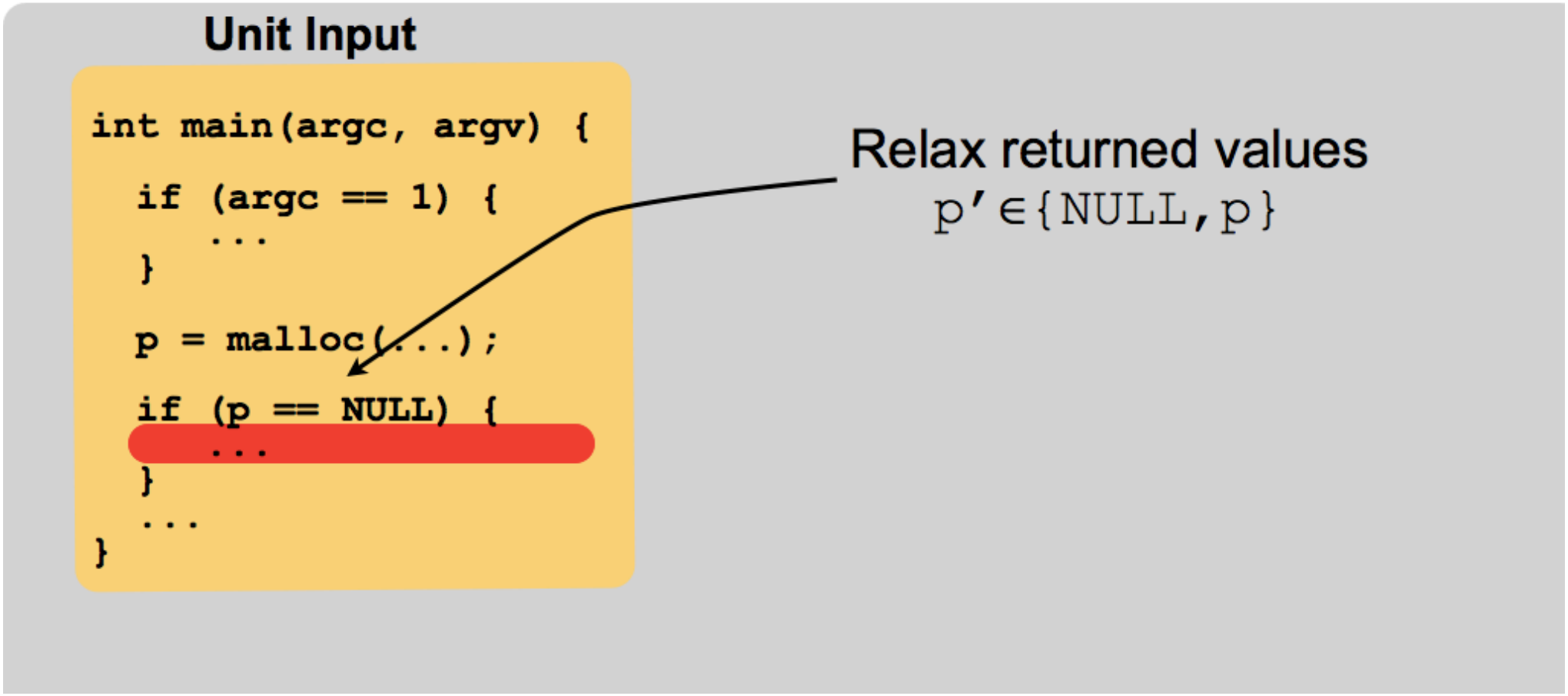


RC
Relaxed Consistency

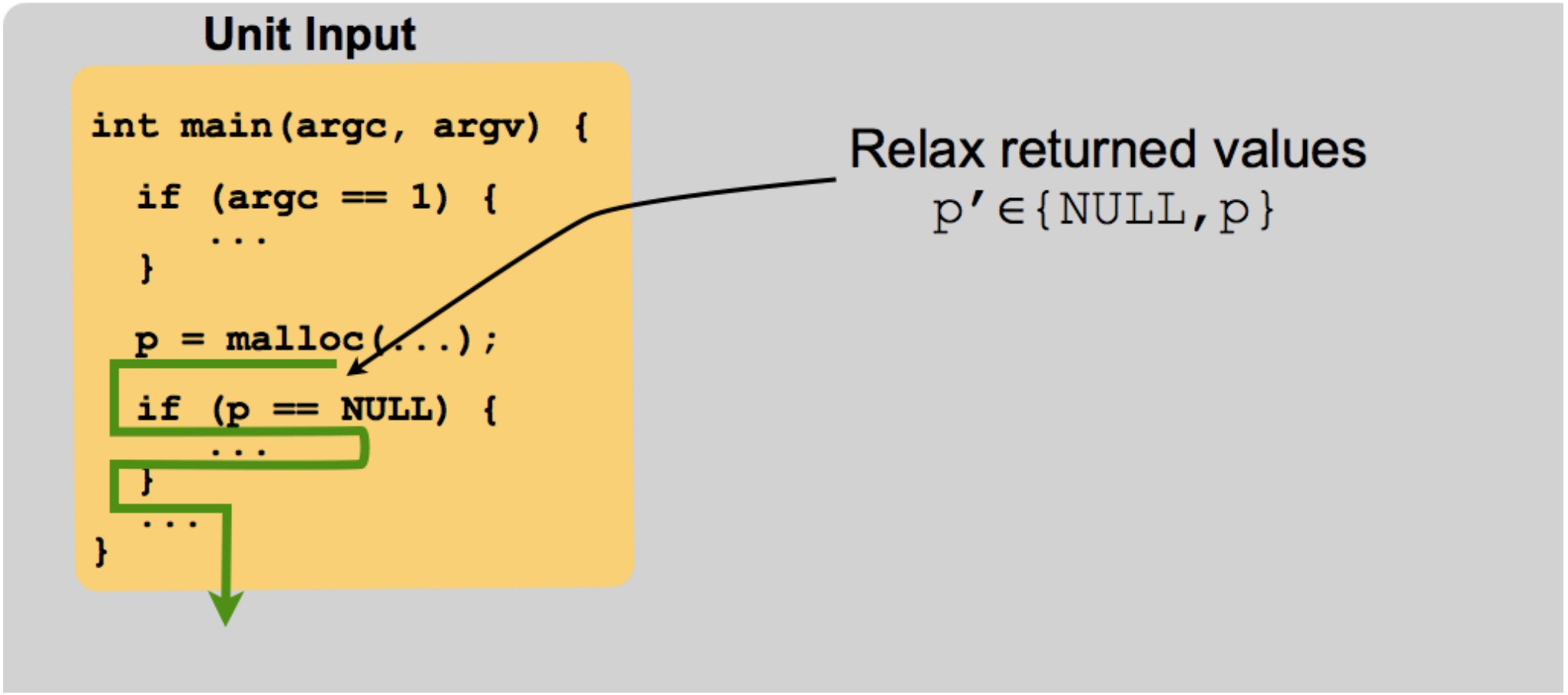
Unit Input

```
int main(argc, argv) {

  if (argc == 1) {
     ...
  }

  p = malloc(...);

  if (p == NULL) {
     ...
  }
  ...
}
```

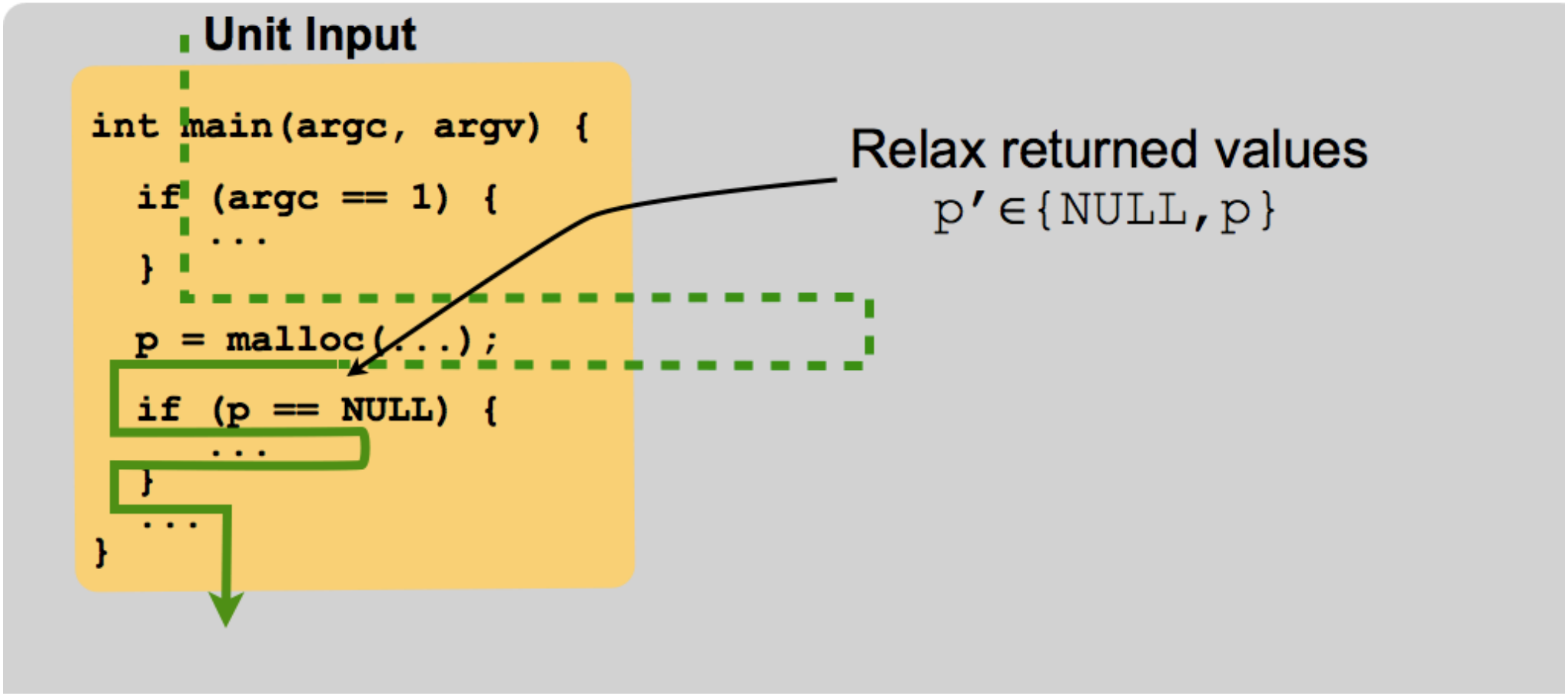
RC
Relaxed Consistency



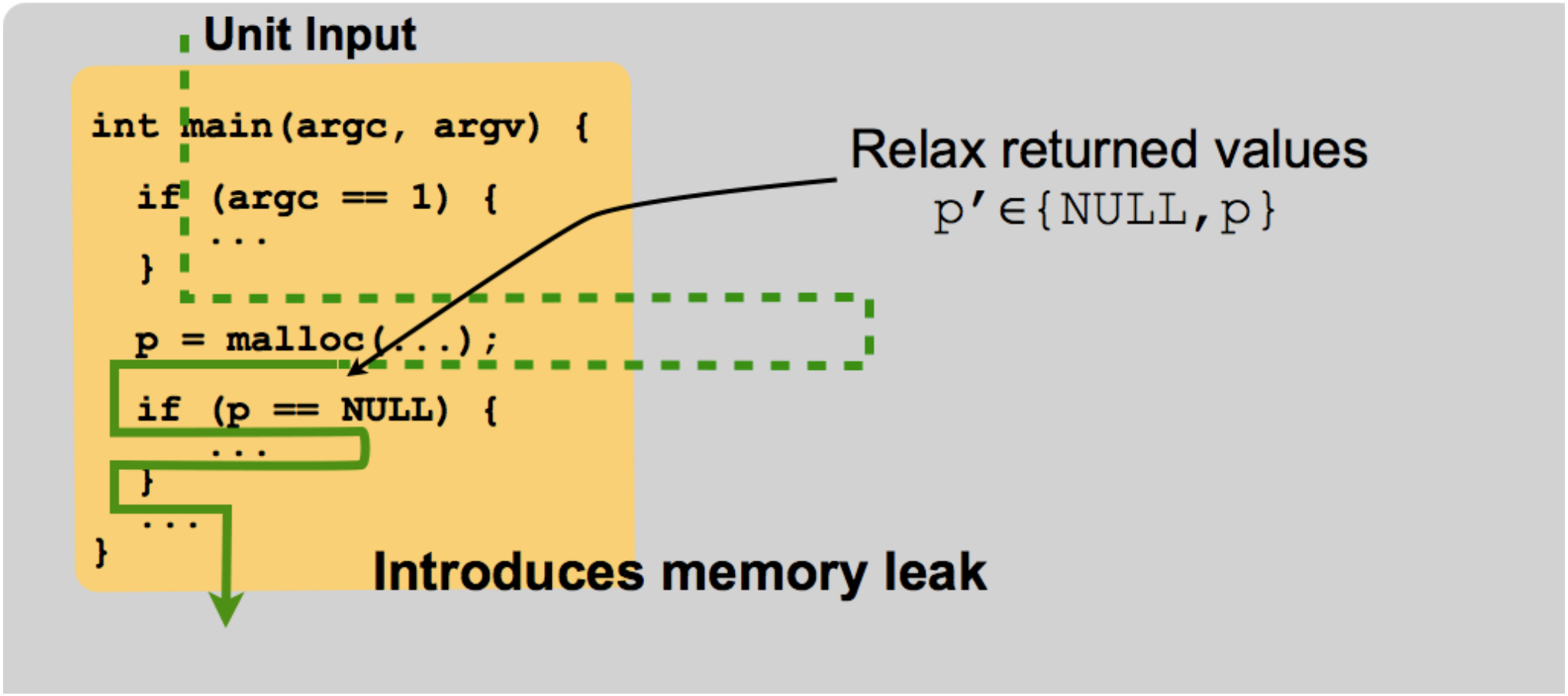
RC
Relaxed Consistency



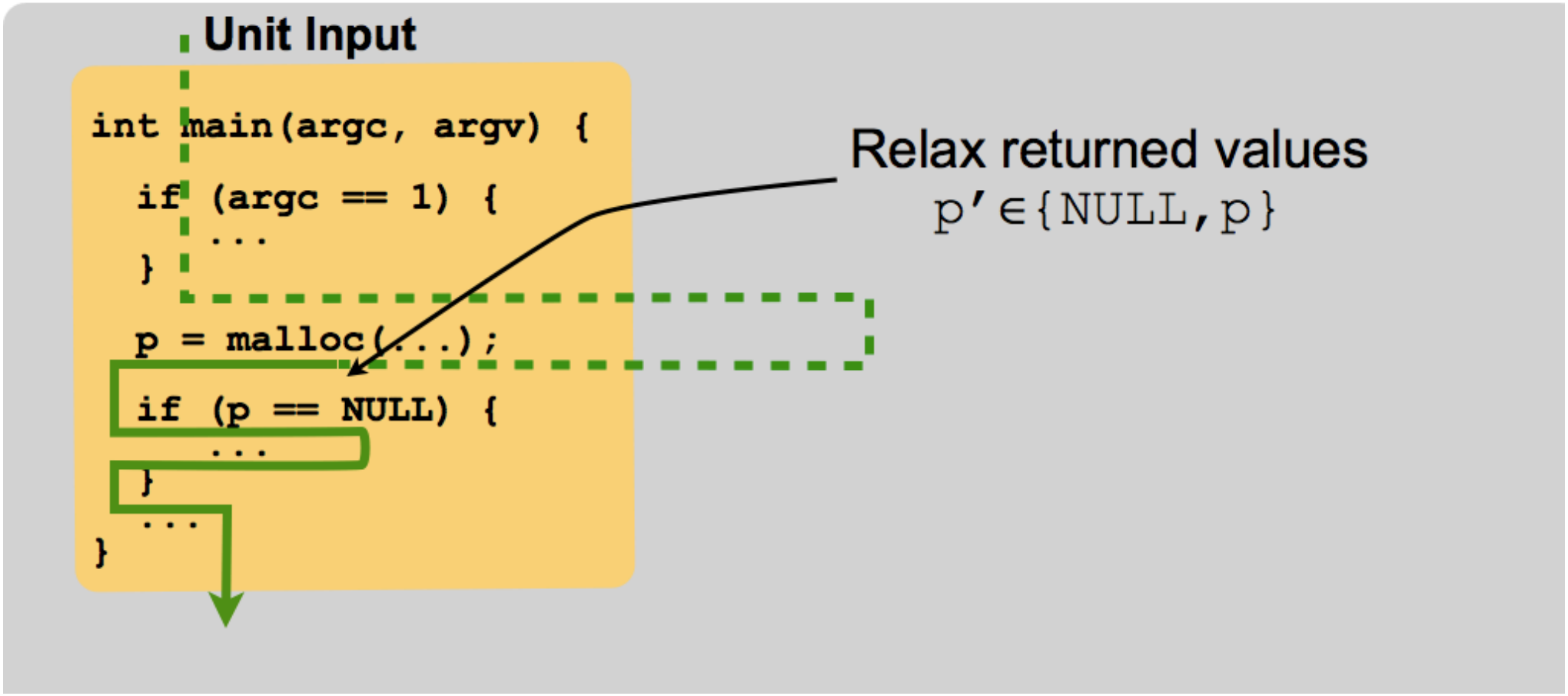
RC
Relaxed Consistency



RC
Relaxed Consistency



RC
Relaxed Consistency



Execution Consistency Models

Model	FNs w.r.t. unit	FPs w.r.t. unit	# system paths

Execution Consistency Models

Model	FNs w.r.t. unit	FPs w.r.t. unit	# system paths
Concrete			

Execution Consistency Models

Model	FNs w.r.t. unit	FPs w.r.t. unit	# system paths
Concrete			
SC-SE			

Execution Consistency Models

Model	FNs w.r.t. unit	FPs w.r.t. unit	# system paths
Concrete			
SC-SE			
SC-UE			

Execution Consistency Models

Model	FNs w.r.t. unit	FPs w.r.t. unit	# system paths
Concrete			
SC-SE			
SC-UE			
RC			

Execution Consistency Models

Model	FNs w.r.t. unit	FPs w.r.t. unit	# system paths
Concrete			
SC-SE			
SC-UE			
RC			

Execution Consistency Models

Model	FNs w.r.t. unit	FPs w.r.t. unit	# system paths
Concrete			
SC-SE			
SC-UE			
RC			
CFG			

Execution Consistency Models

Model	FNs w.r.t. unit	FPs w.r.t. unit	# system paths
Concrete			
SC-SE			
SC-UE			
RC			
CFG			
Local			

Execution Consistency Models

Model	FNs w.r.t. unit	FPs w.r.t. unit	# system paths	Uses
Concrete				Valgrind
SC-SE				KLEE
SC-UE				DART
RC				RevNIC
CFG				Disassemblers
Local				DDT

Execution Consistency Models

Model	FNs w.r.t. unit	FPs w.r.t. unit	# system paths	Uses
Concrete				Valgrind
SC-SE				KLEE
SC-UE				DART
RC				RevNIC
CFG				Disassemblers
Local				DDT

Design your own models

Outline

- Theory
  Execution consistency models
- System
  S²E: Platform for in-vivo multi-path analysis
- Results
  Using S²E in practice

http://s2e.epfl.ch

Outline

- Theory
  Execution consistency models
- System
  S²E: Platform for in-vivo multi-path analysis
- Results
  Using S²E in practice

http://s2e.epfl.ch

Symbolic Execution

Symbolic Execution

```
int func(int a, int b)
{

  if (a > 0) {
     ...
  }

  if (b < 0) {
  ...
  }

}
```

Symbolic Execution

```
a=1 b=2  a=3 b=5  a=5 b=2  a=10 b=22
```

```
int func(int a, int b)
{

  if (a > 0) {
      ...
  }

  if (b < 0) {
  ...
  }

}
```

Symbolic Execution

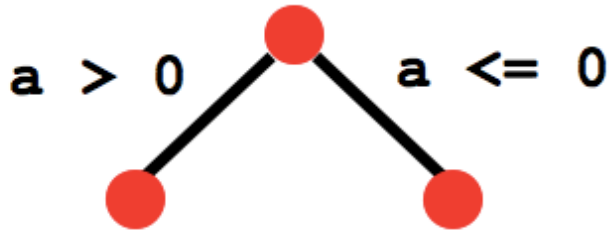
a=λ b=δ

```
int func(int a, int b)
{

  if (a > 0) {
     ...
  }

  if (b < 0) {
  ...
  }

}
```

Symbolic Execution

a=λ b=δ

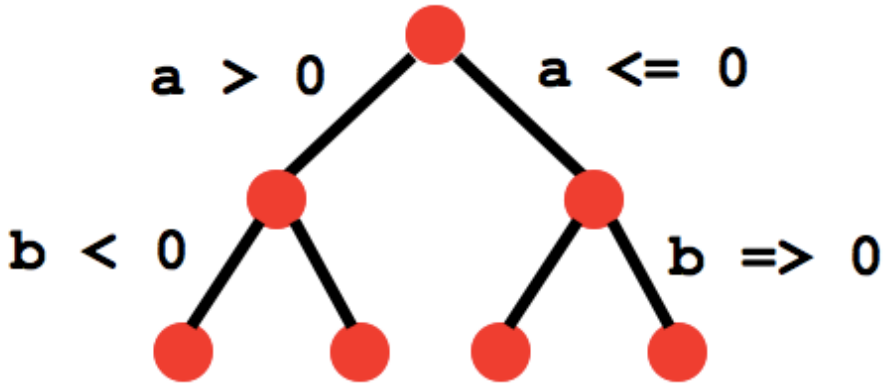
```
int func(int a, int b)
{

  if (a > 0) {
     ...
  }

  if (b < 0) {
  ...
  }

}
```



Symbolic Execution

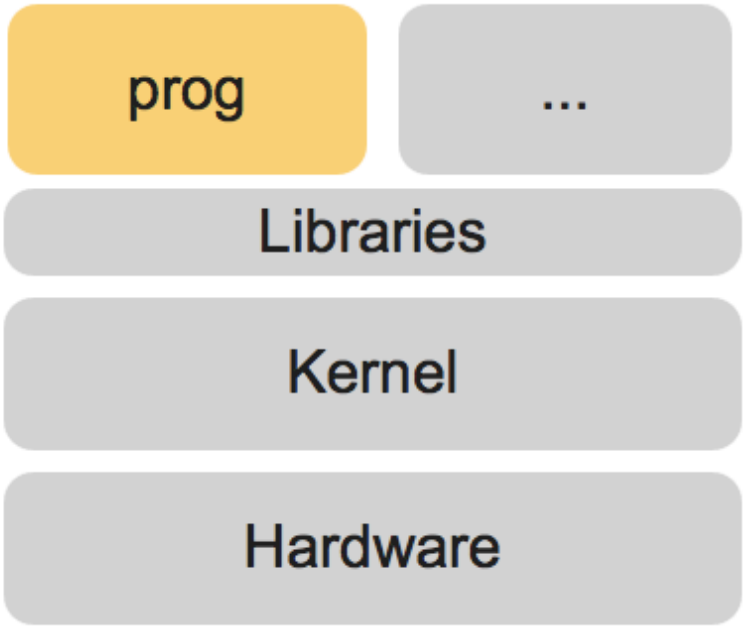
a=λ b=δ

```
int func(int a, int b)
{

  if (a > 0) {
     ...
  }

  if (b < 0) {
  ...
  }

}
```



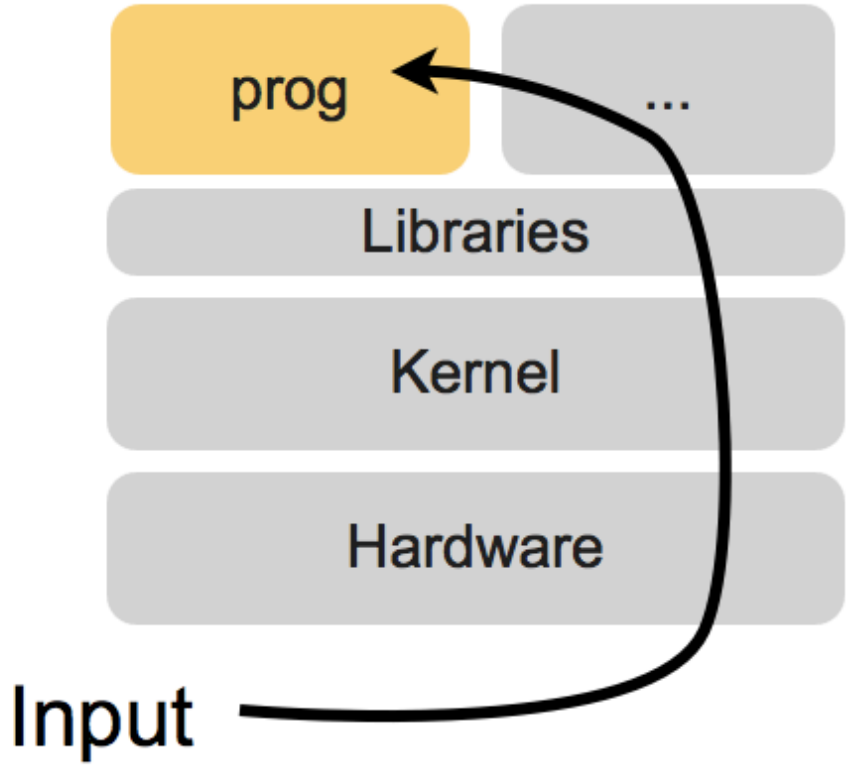
Concrete ➡ Symbolic

```
int main(argc, argv) {

  if (argc == 0) {
     ...
  }

  p = malloc(...);

  if (p == NULL) {
     ...
  }
  ...
}
```



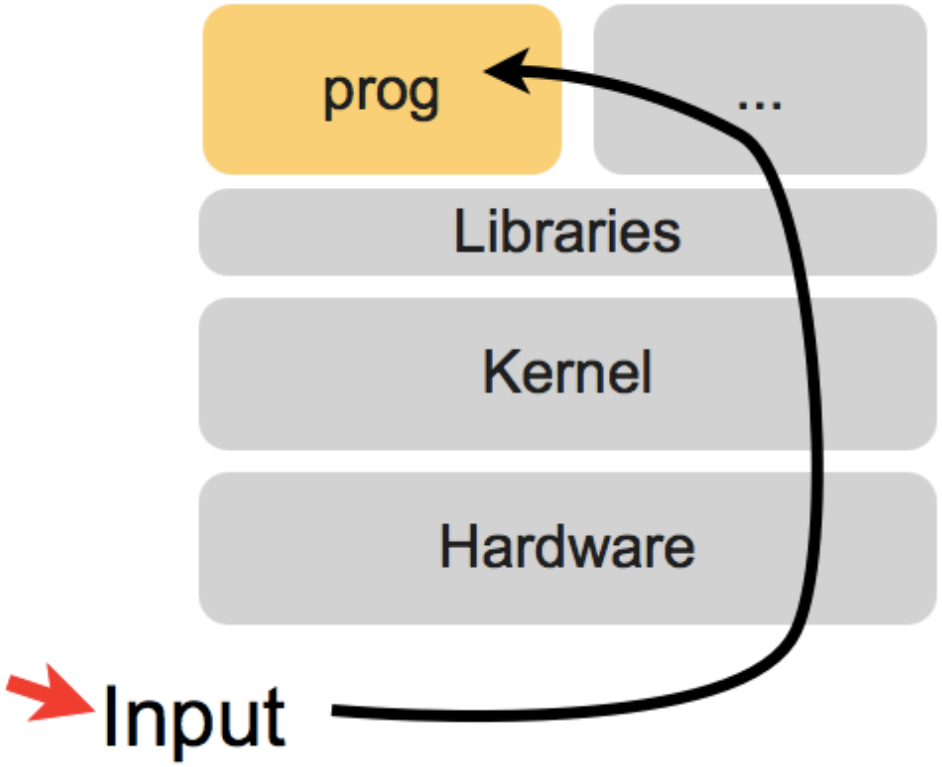
Concrete ➡ Symbolic

```
int main(argc, argv) {

  if (argc == 0) {
     ...
  }

  p = malloc(...);

  if (p == NULL) {
     ...
  }
  ...
}
```



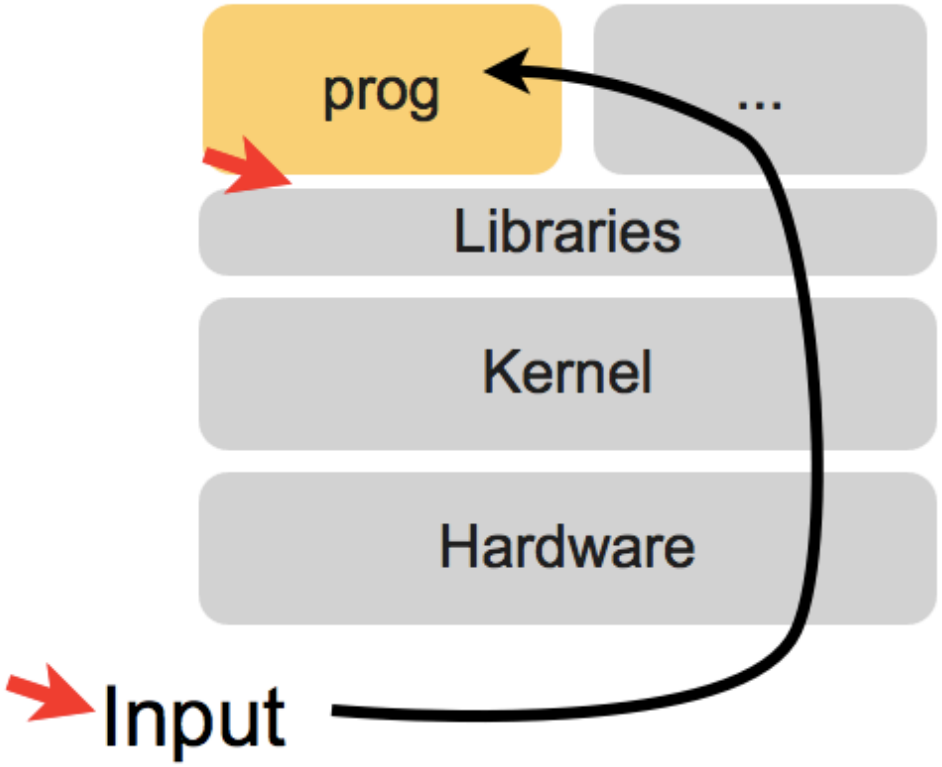
Concrete ➡ Symbolic

```
int main(argc, argv) {

  if (argc == 0) {
     ...
  }

  p = malloc(...);

  if (p == NULL) {
     ...
  }
  ...
}
```



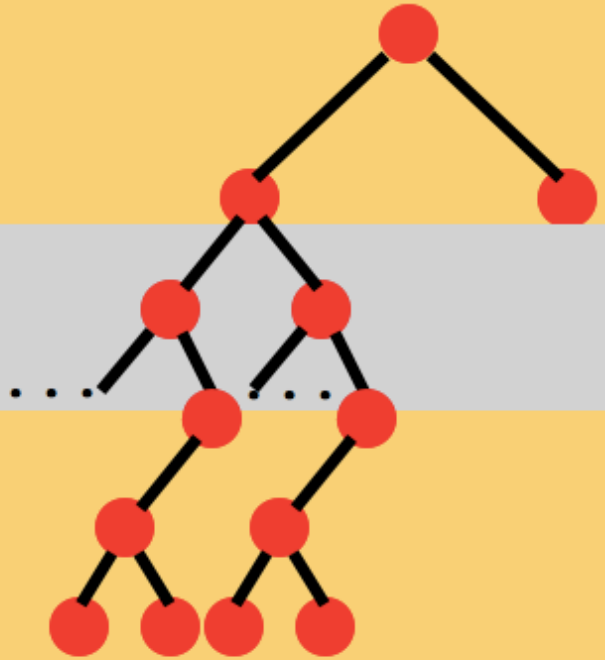
Concrete ➡ Symbolic

```
int main(argc, argv) {

  if (argc == 0) {
     ...
  }

  p = malloc(...);

  if (p == NULL) {
     ...
  }
  ...
}
```

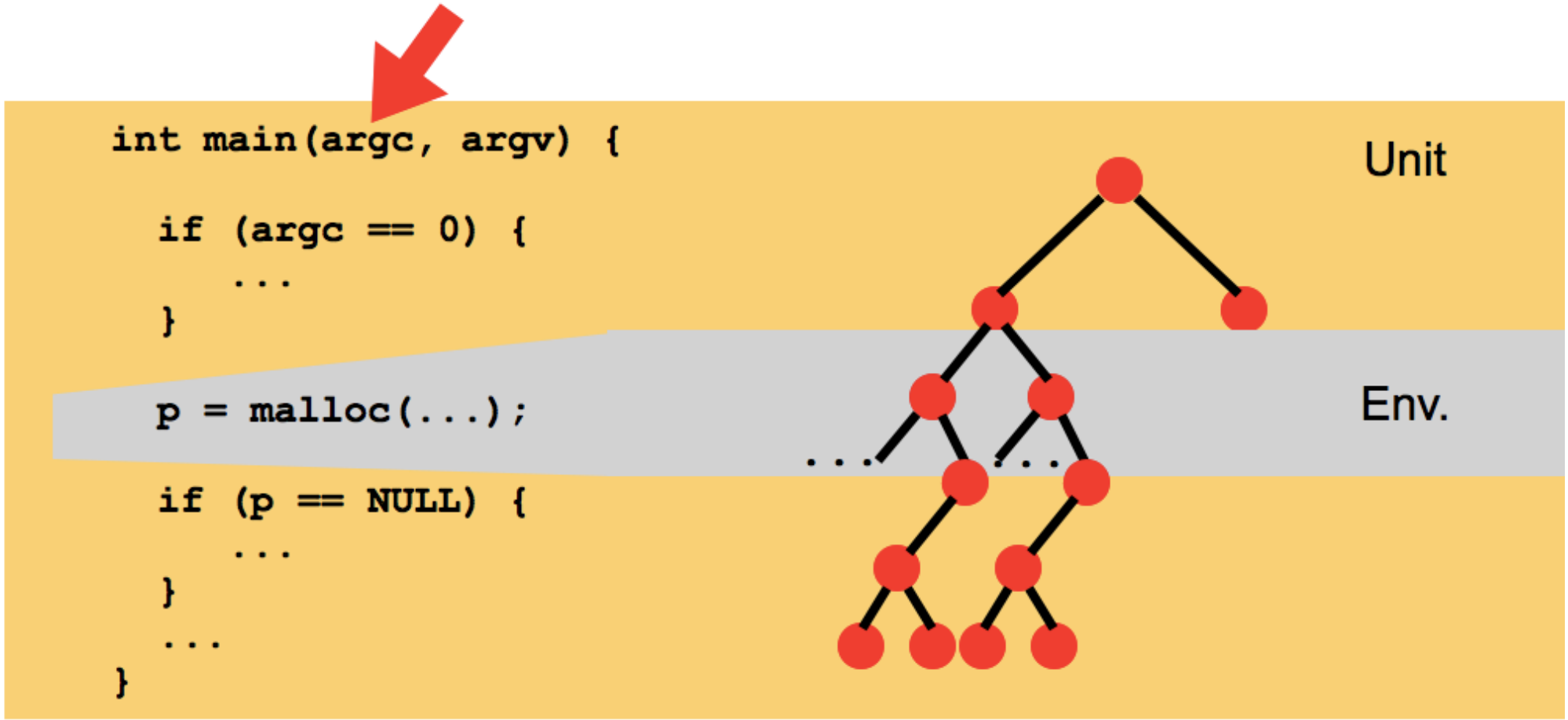


Concrete ➡ Symbolic

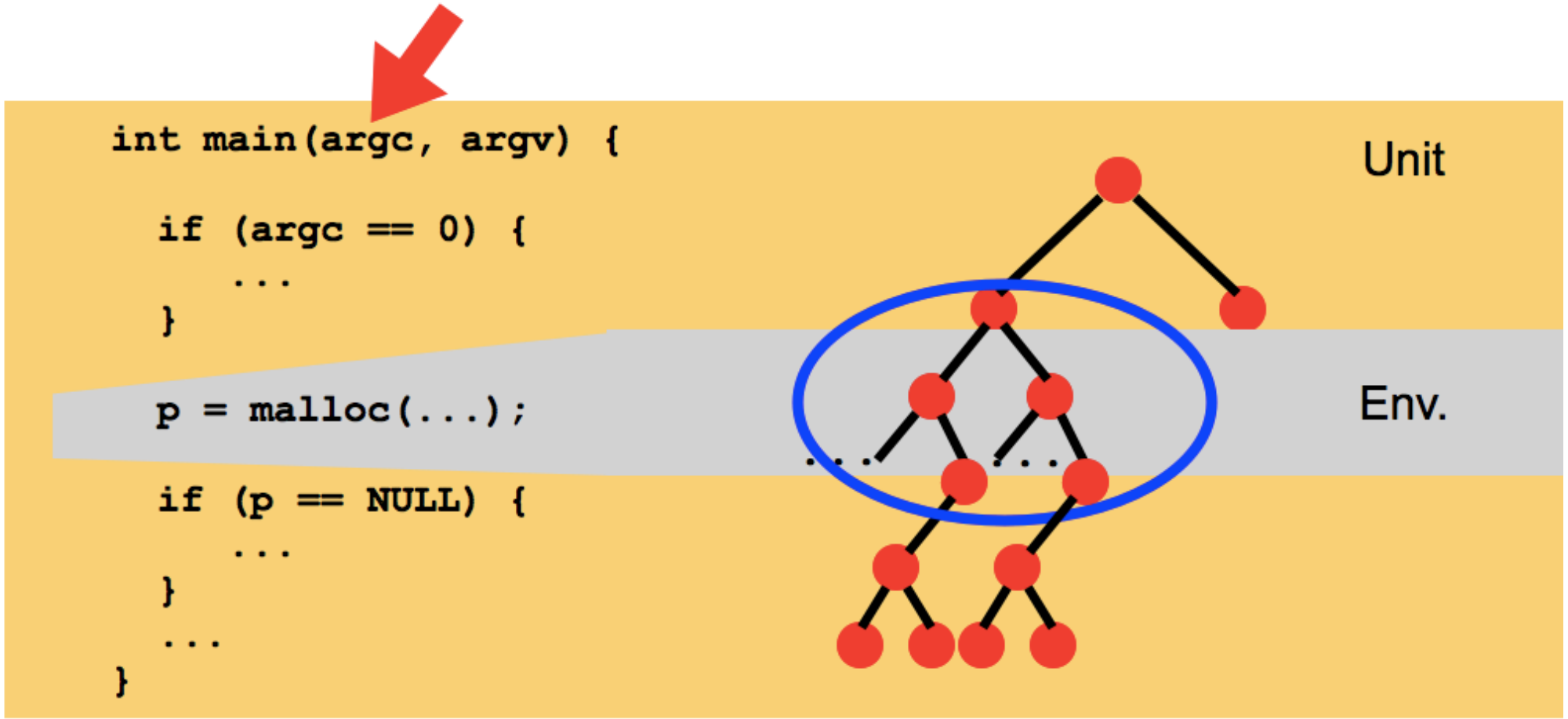
```
int main(argc, argv) {

  if (argc == 0) {
     ...
  }

  p = malloc(...);

  if (p == NULL) {
     ...
  }
  ...
}
```



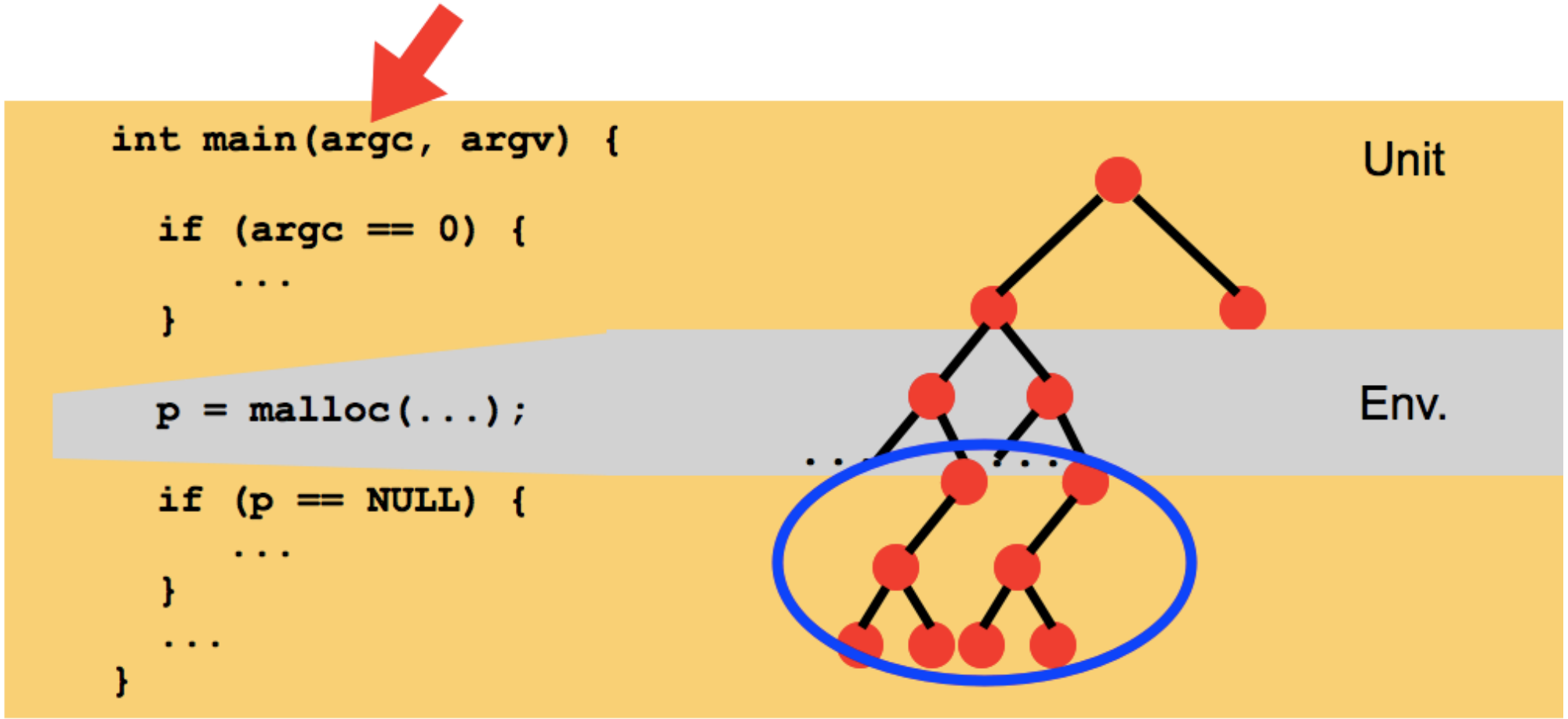
Concrete ➡ Symbolic



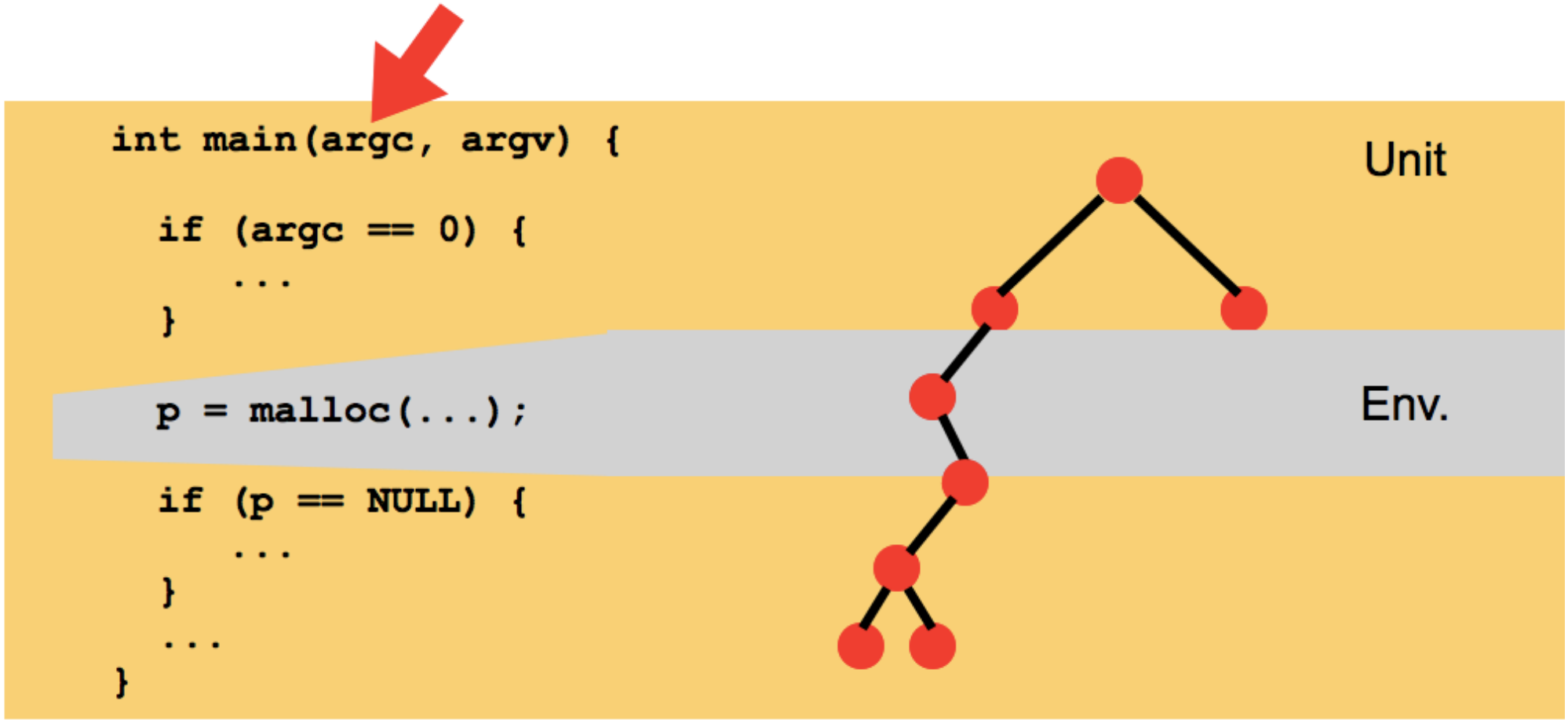
Concrete ➡ Symbolic



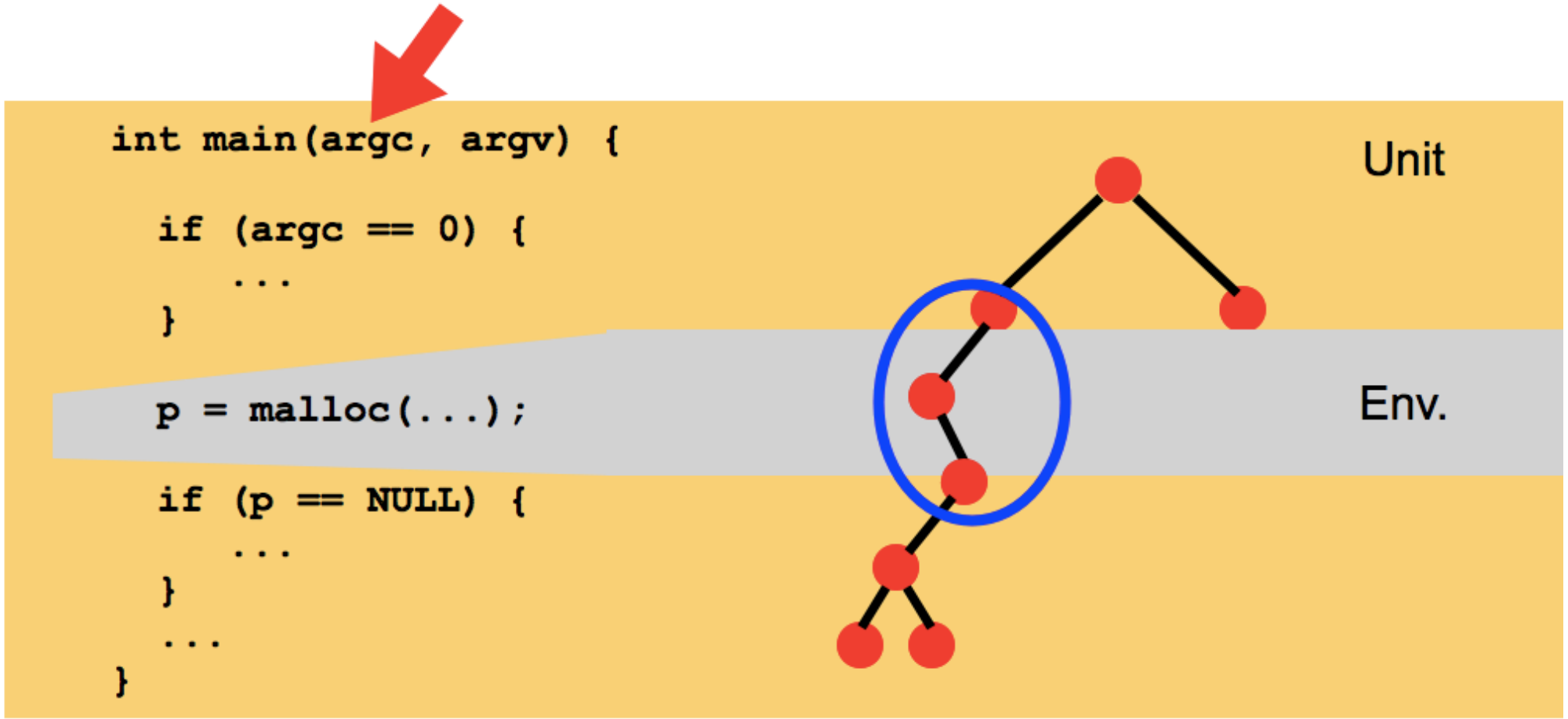
Concrete ➡ Symbolic



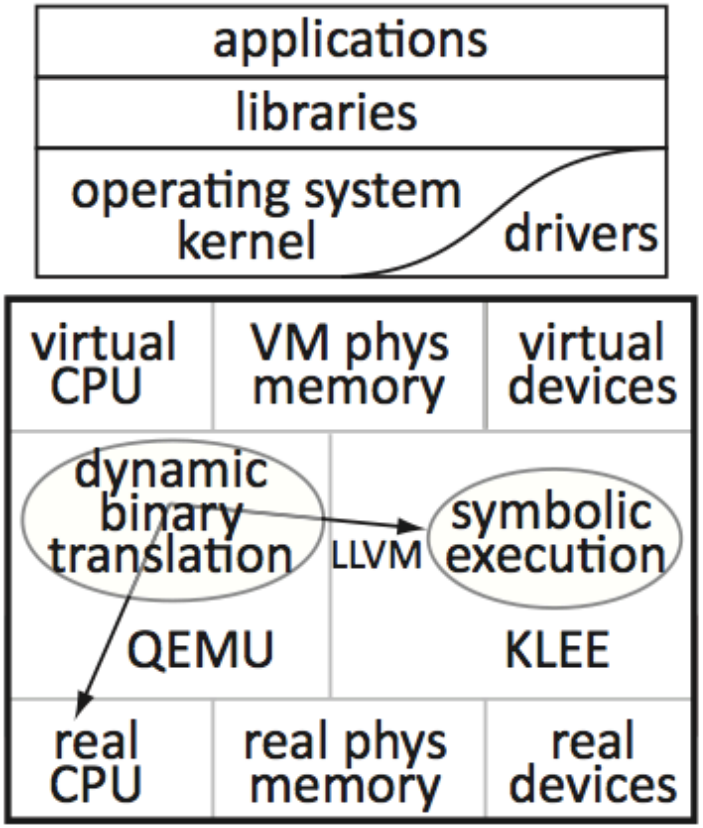
Symbolic ➡ Concrete



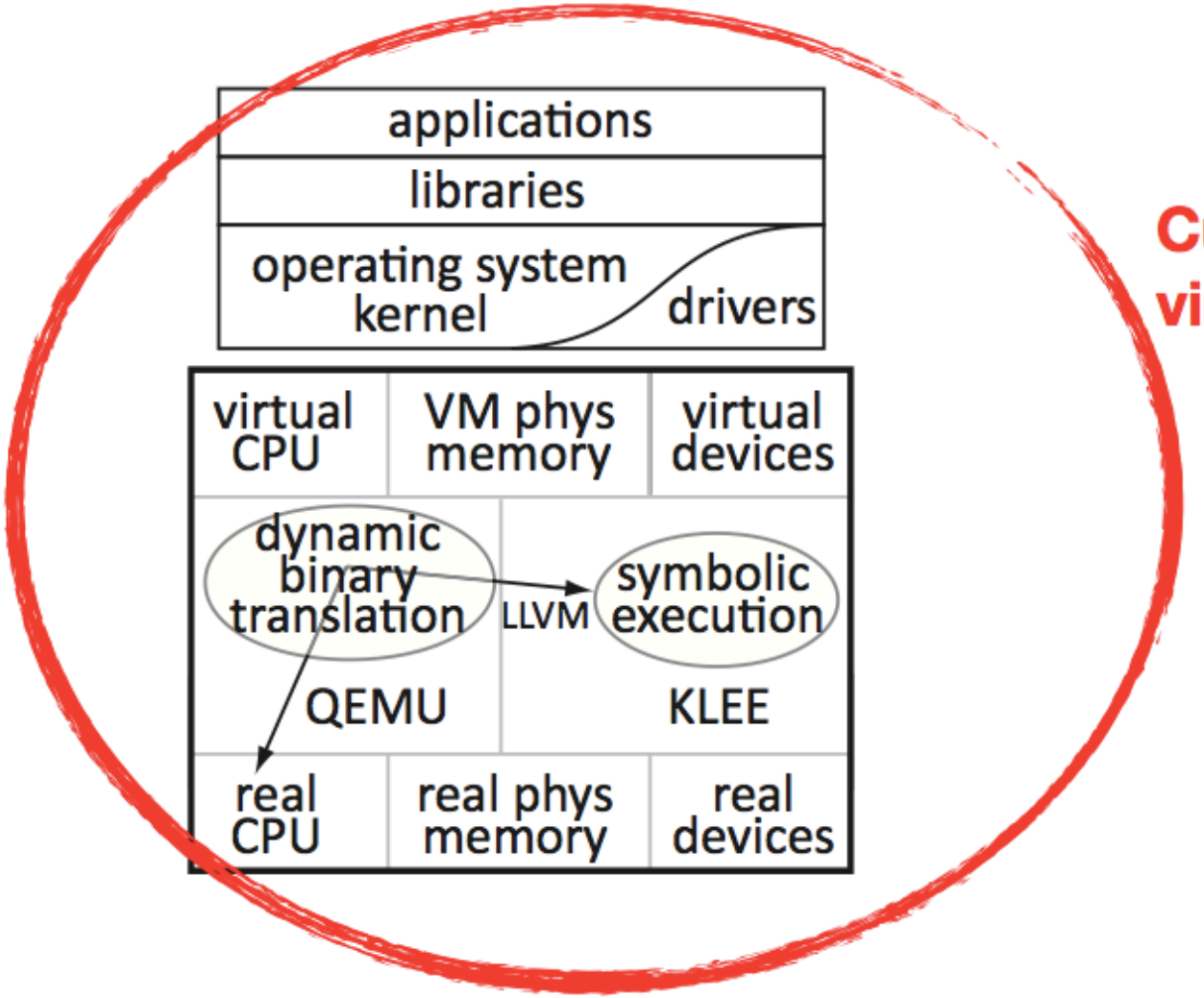
Symbolic ➡ Concrete



S2E Is A Virtual Machine

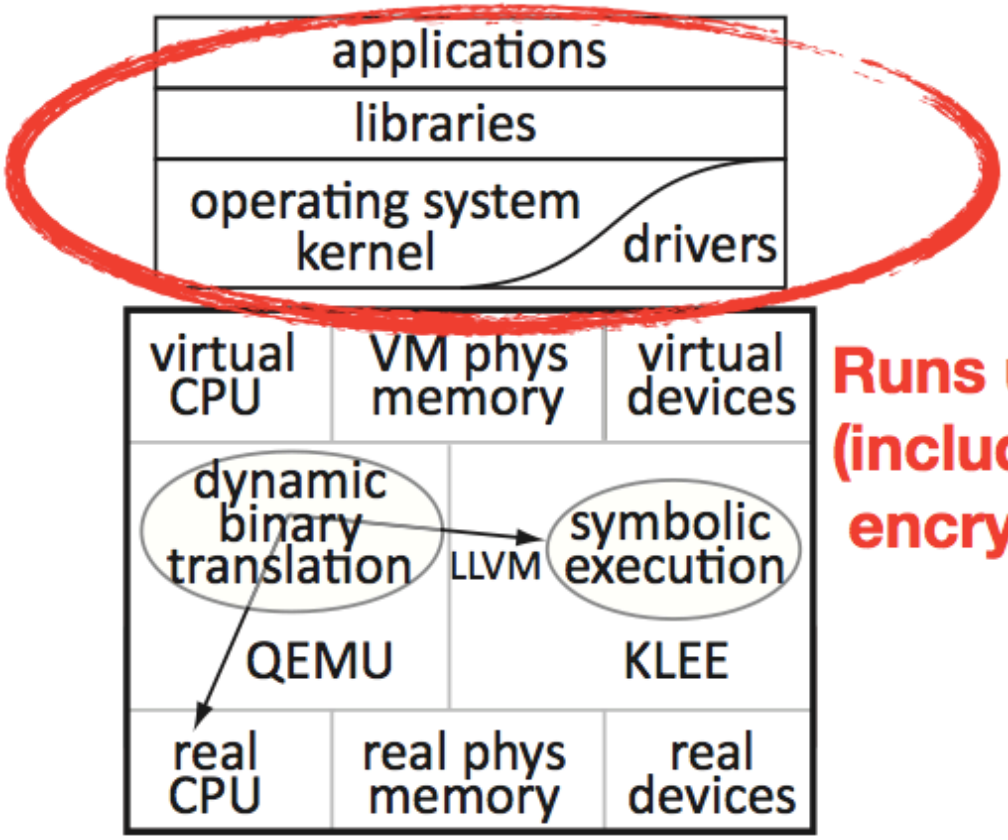


S2E Is A Virtual Machine



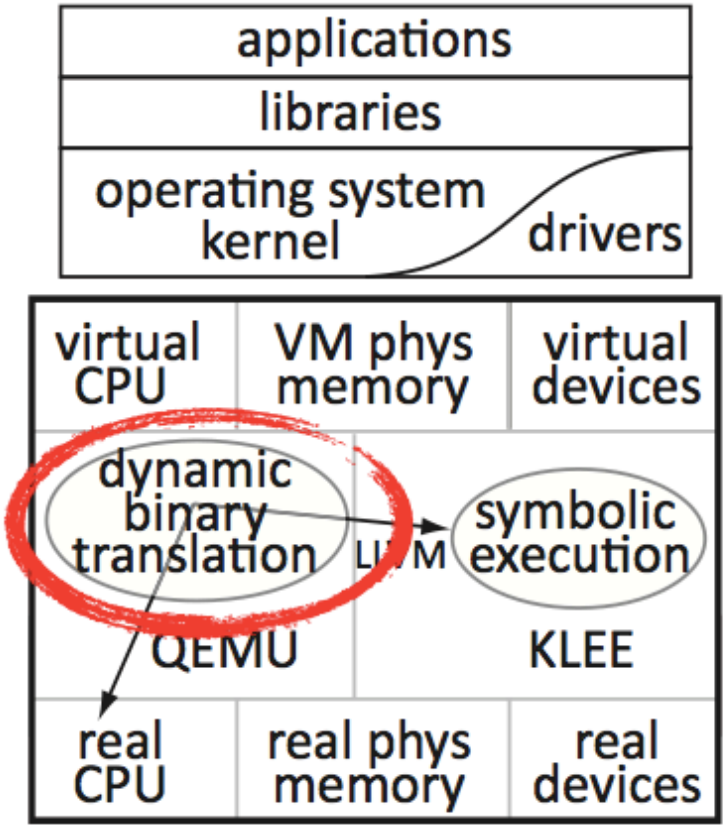
Customized virtual machine

S2E Is A Virtual Machine



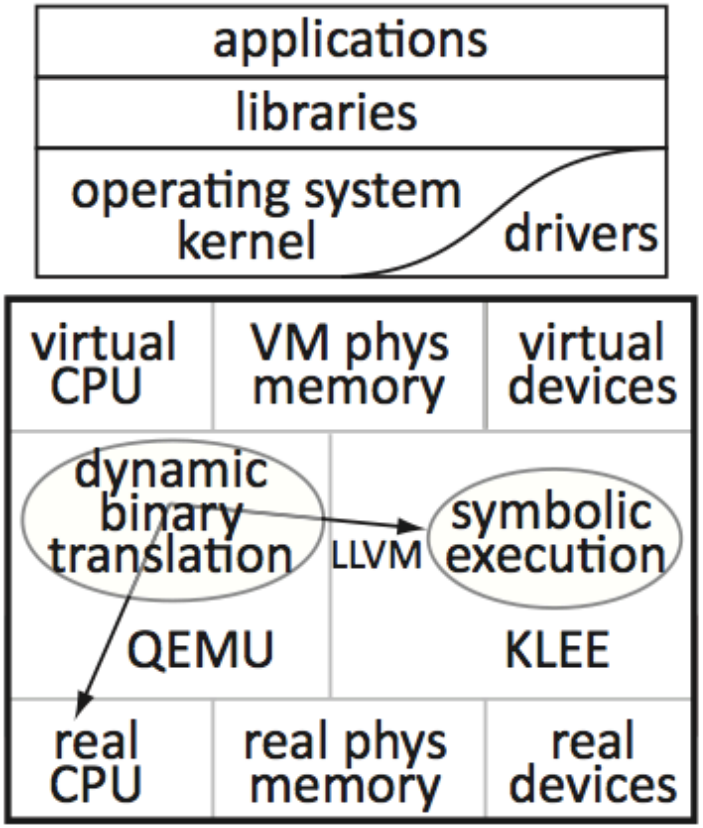
Runs unmodified x86 binaries (including proprietary/obfuscated/ encrypted binaries)

S2E Is A Virtual Machine



Selection done at runtime
Most code runs "natively"

S2E Is A Virtual Machine



Shared concrete/symbolic state representation

Outline

- Theory
  Execution consistency models
- System
  S²E: Platform for in-vivo multi-path analysis
- Results
  Using S²E in practice

http://s2e.epfl.ch

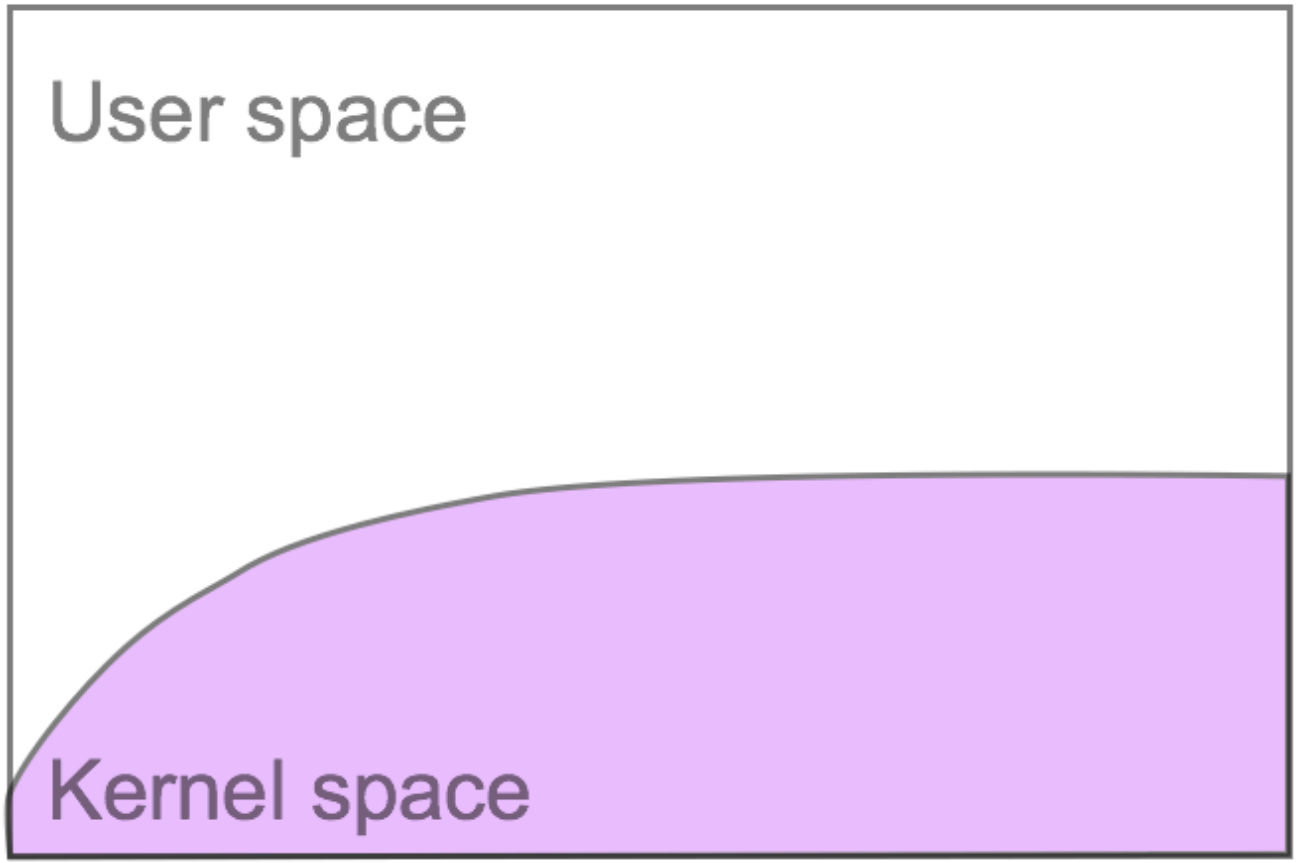
Outline

- Theory
  Execution consistency models
- System
  S²E: Platform for in-vivo multi-path analysis
- Results
  Using S²E in practice

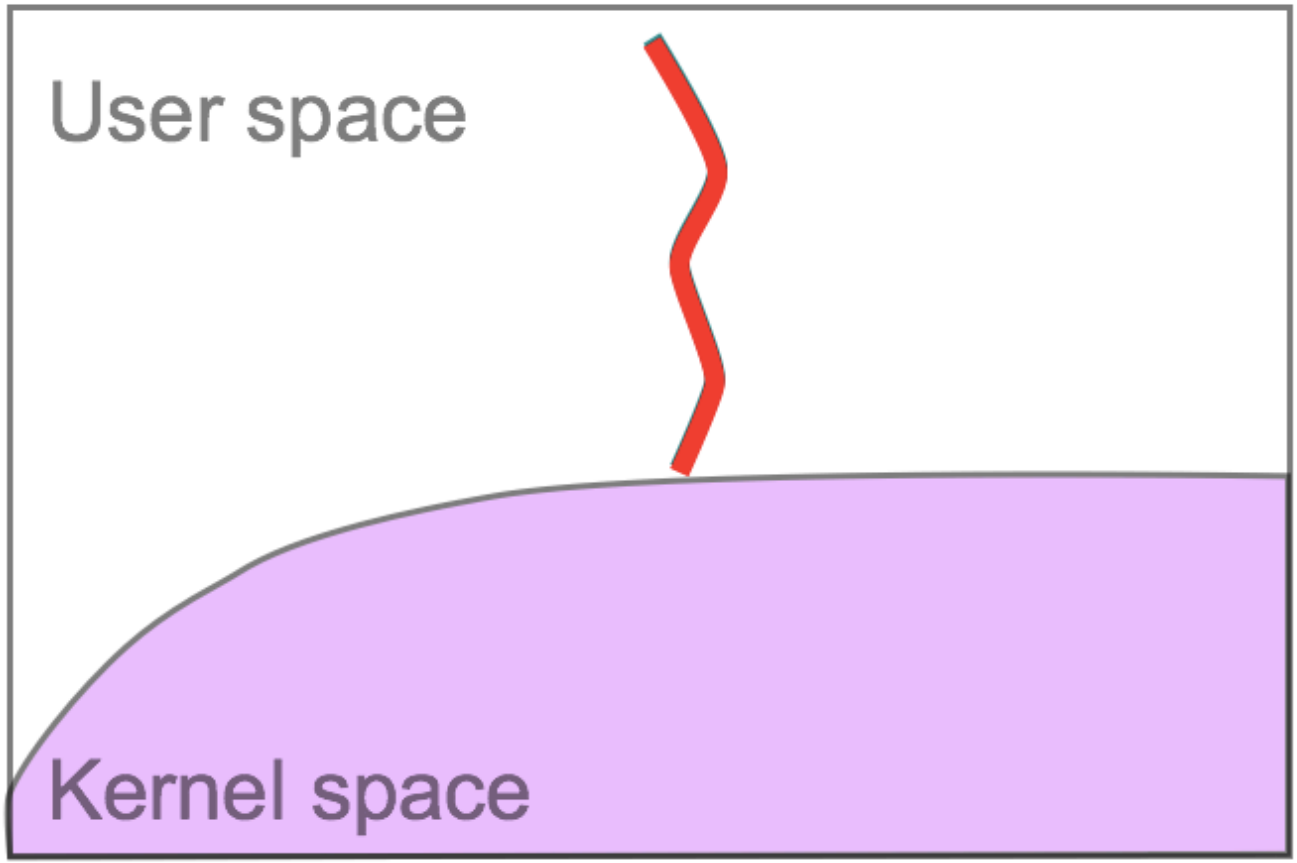
http://s2e.epfl.ch

Multi-Path Performance Profiling

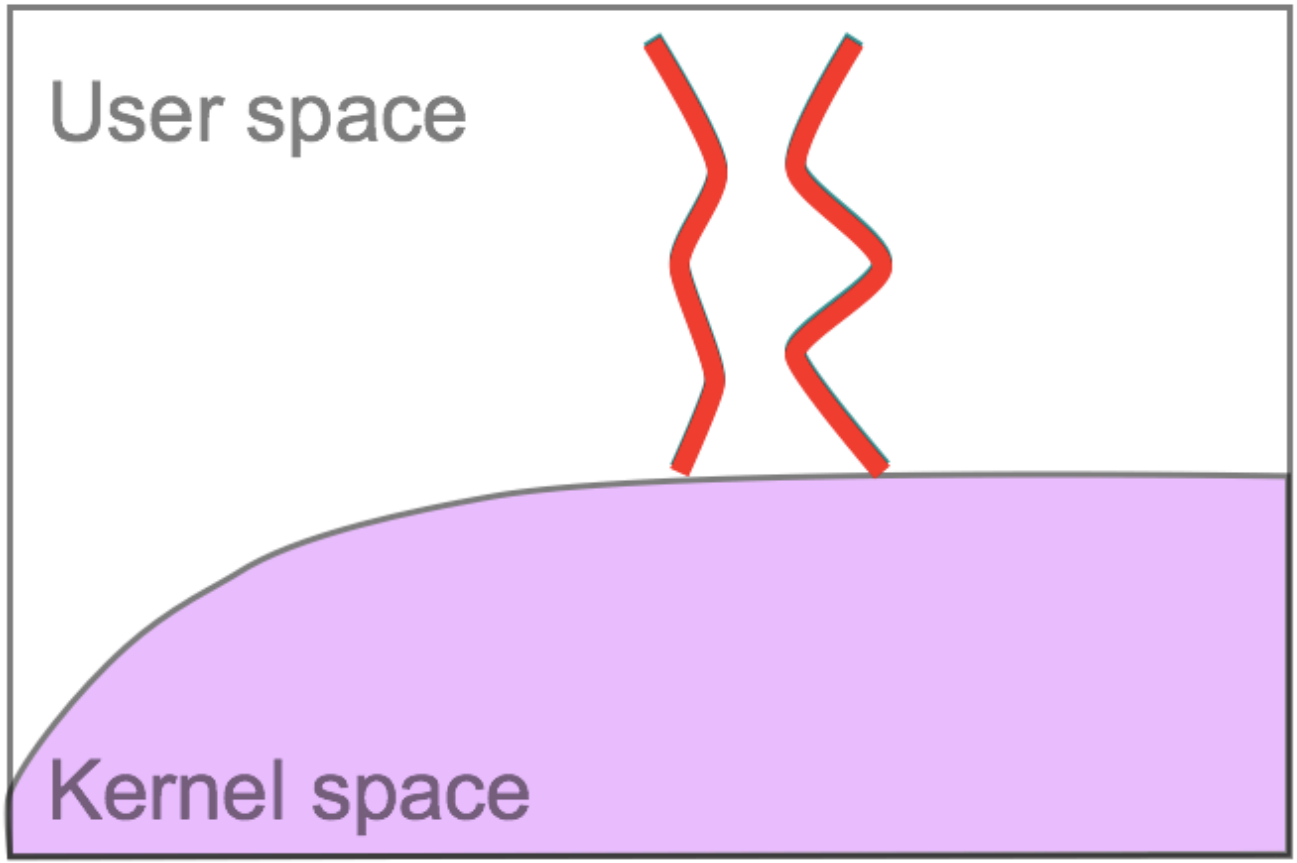
Single-Path Performance Profiling



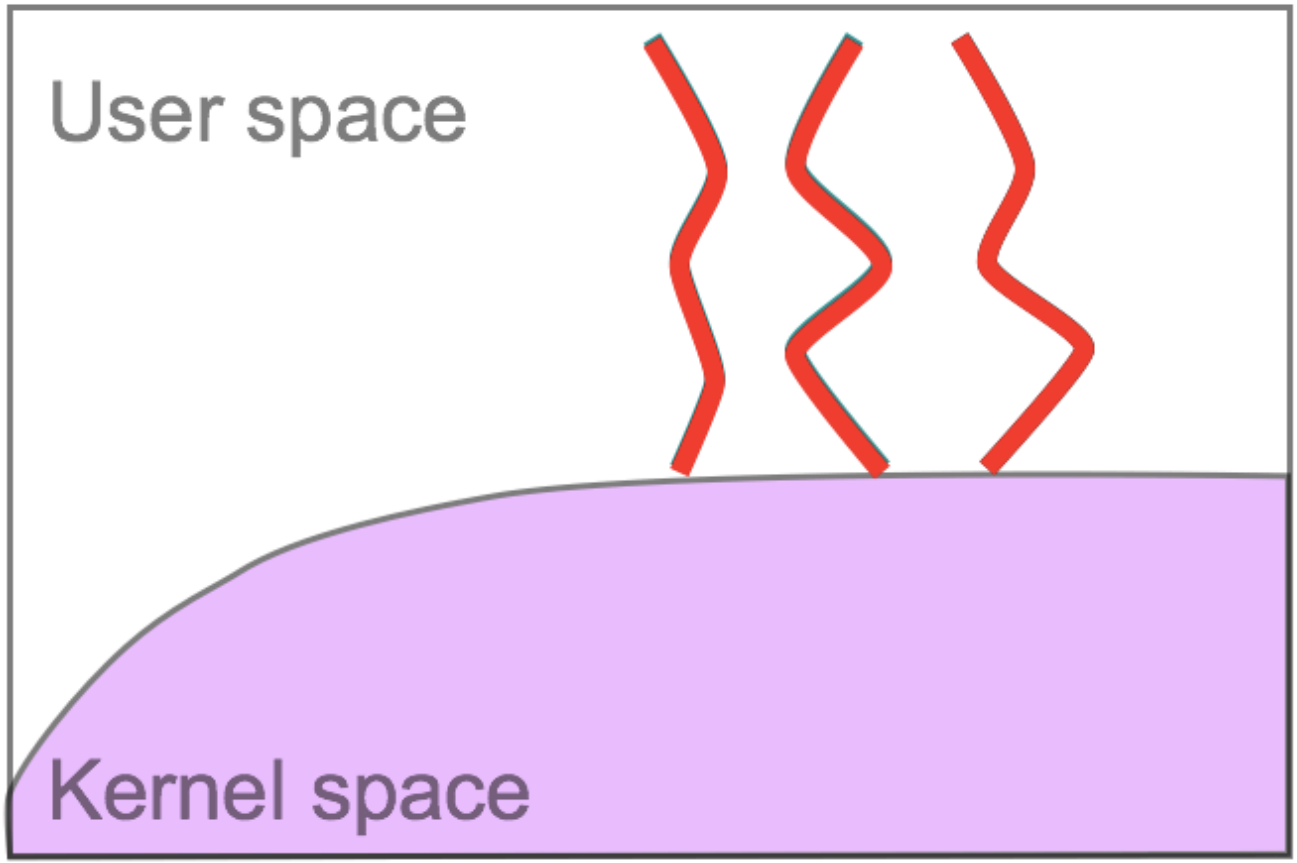
Single-Path Performance Profiling



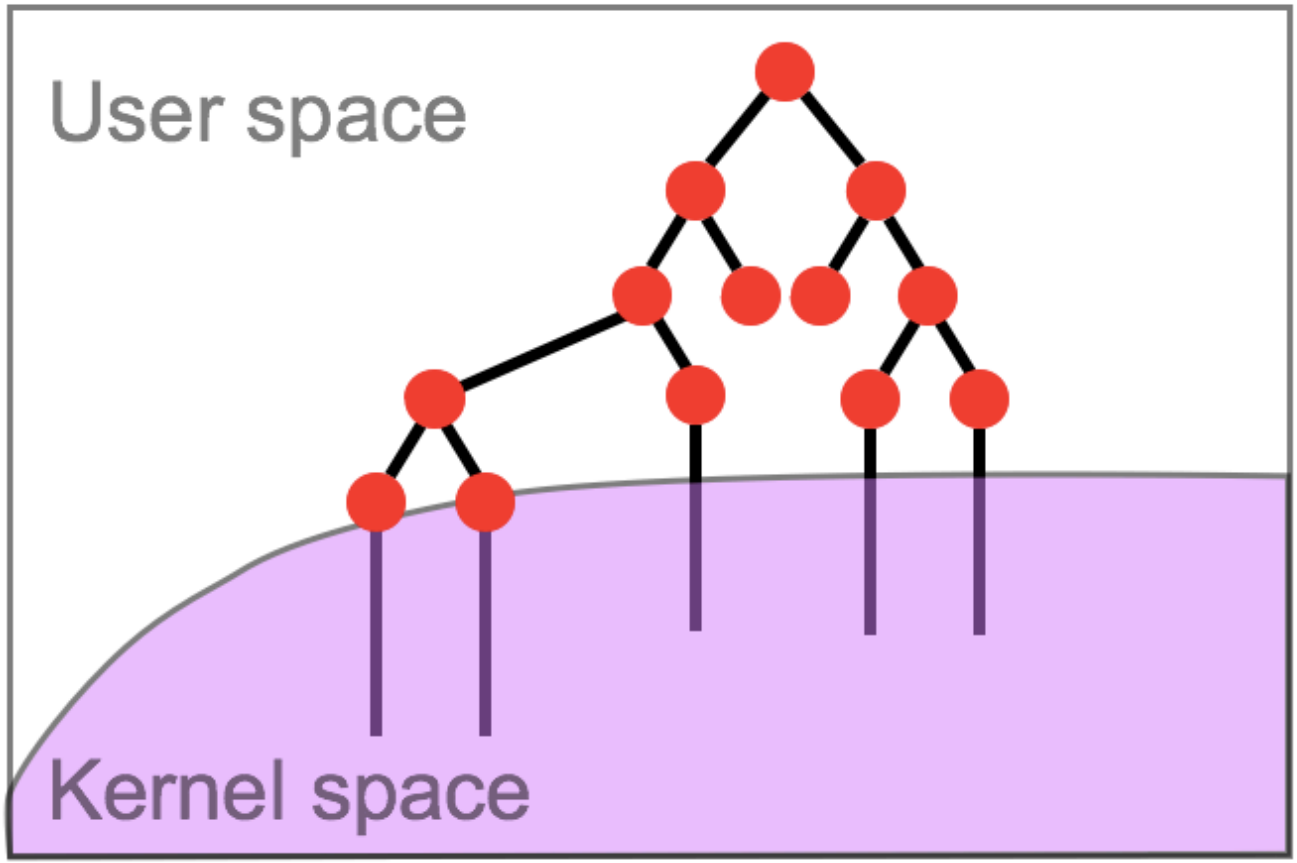
Single-Path Performance Profiling



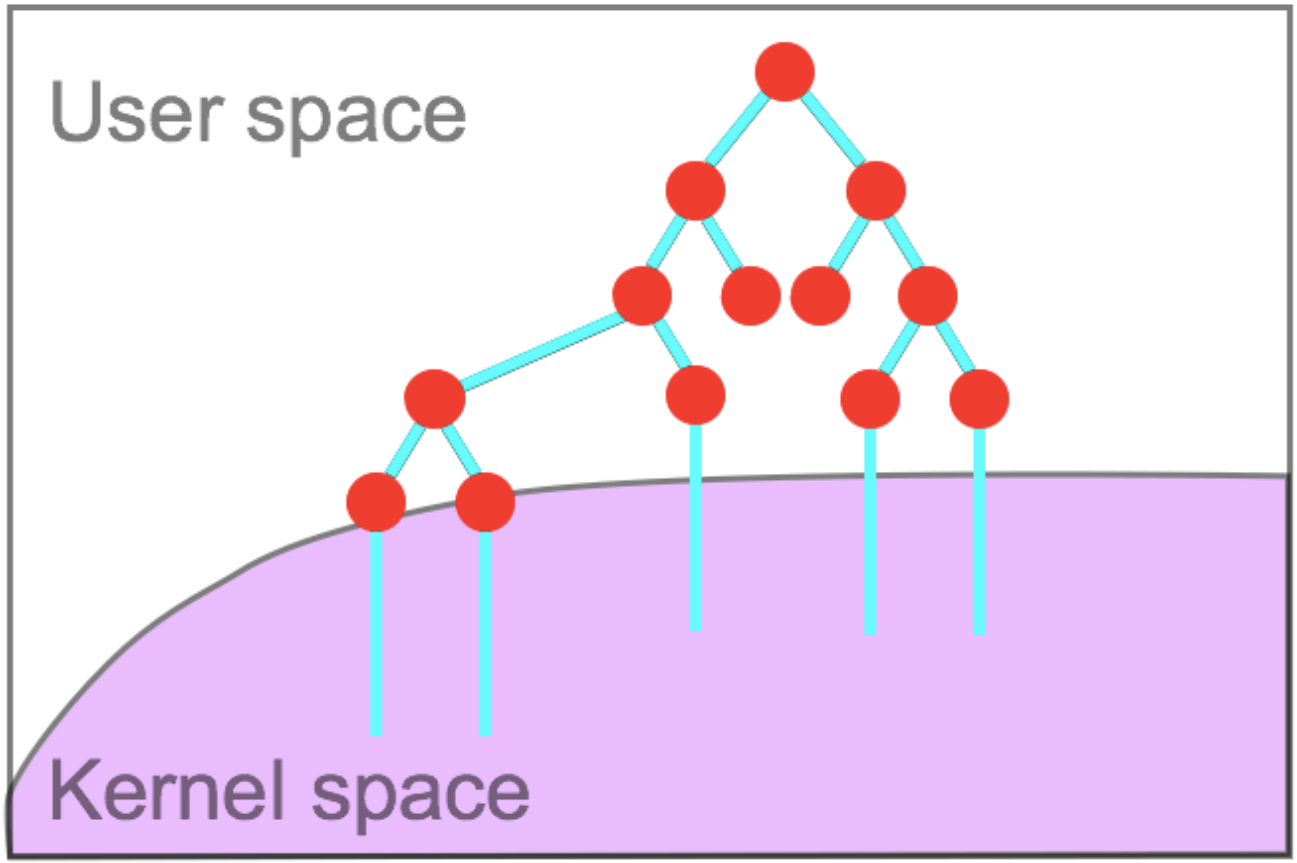
Single-Path Performance Profiling



Multi-Path In-Vivo Profiling



Multi-Path In-Vivo Profiling



PROFs

- Cache Simulator
  Models arbitrary cache hierarchies
- Instruction Counter
  Machine instructions
- MMU Monitor
  Tracks TLB misses and page faults

Finding Performance Envelopes

- Upper and lower bound on performance
- Fastest and slowest execution path
- Metrics?
  - # instructions, cache misses, page faults, ...

Finding Performance Envelopes

ping

Finding Performance Envelopes

ping

Finding Performance Envelopes

ping

Finding Performance Envelopes

ping

>1.5 million instructions

Finding Performance Envelopes

ping

- Unbounded instruction count
- Infinite loop bug

>1.5 million instructions

Infinite Loop in Ping

```
void process_options(optptr...) {
  …
  while (totlen > 0) {
    …
    opt = optptr;
    …
    switch (*opt) {
      case OPTION_ROUTE_RECORD:
        length = *++opt;

        if (length < 4)
          continue;
    }
    …
  }
}
```

Infinite Loop in Ping

```
void process_options(optptr...) {
  …
  while (totlen > 0) {
    …
    opt = optptr;
    …
    switch (*opt) {
      case OPTION_ROUTE_RECORD:
        length = *++opt;

        if (length < 4)
          continue;
    }
    …
  }
}
```

Infinite Loop in Ping

```
void process_options(optptr...) {
  …
  while (totlen > 0) {
    …
    opt = optptr;
    …
    switch (*opt) {
      case OPTION_ROUTE_RECORD:
        length = *++opt;

        if (length < 4)
          continue;
    }
    …
  }
}
```

Infinite Loop in Ping

```
void process_options(optptr...) {
  …
  while (totlen > 0) {
    …
    opt = optptr;
    …
    switch (*opt) {
      case OPTION_ROUTE_RECORD:
        length = *++opt;

        if (length < 4)
          continue;
    }
    …
  }
}
```

Infinite Loop in Ping

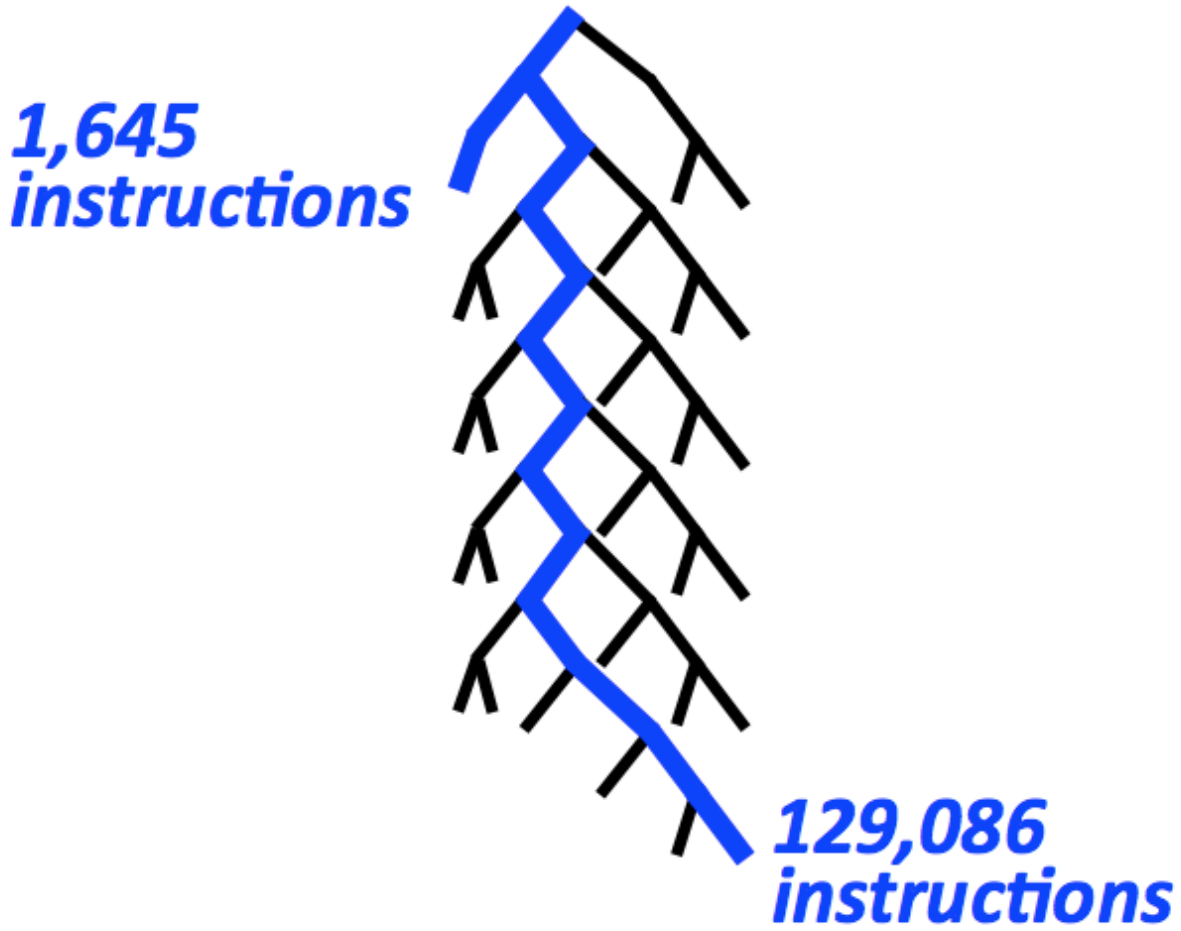
```
void process_options(optptr...) {
  …
  while (totlen > 0) {
    …
    opt = optptr;
    …
    switch (*opt) {
      case OPTION_ROUTE_RECORD:
        length = *++opt;

        if (length < 4)
          continue;
    }
    …
  }
}
```

Infinite Loop in Ping

```
void process_options(optptr...) {
  …
  while (totlen > 0) {
    …
    opt = optptr;
    …
    switch (*opt) {
      case OPTION_ROUTE_RECORD:
        length = *++opt;

        if (length < 4)
          continue;
    }
    …
  }
}
```

Perf. Envelope for Patched Ping

Perf. Envelope for Patched Ping



Other Uses of S2E

- Reverse engineering [Eurosys'10]
- Automated closed-source driver testing [USENIX'10]
- File system corruption impact analysis
  University of Wisconsin-Madison
- Symbolic execution of sensor networks
  RWTH Aachen University
- File system equivalence checking
  Max Planck Institute for Software Systems
- Energy profiling, privacy analysis, ...

Conclusion

- Execution consistency models
- Platform for in-vivo multi-path analysis
- Use of symbolic execution in performance analysis

http://s2e.epfl.ch

Ready-for-use VM, demos, tutorials, source code, documentation