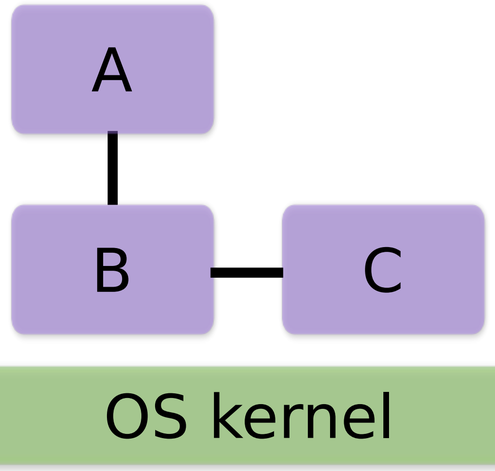
Automated Verification of RPC Stub Code

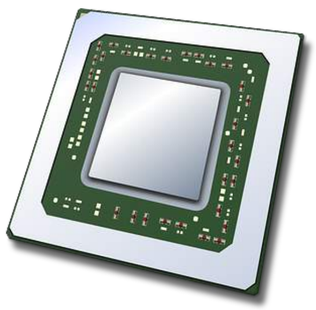
Matthew Fernandez, June Andronick, Gerwin Klein, Ihor Kuz

FM, June 2015

High Assurance Component-based Software

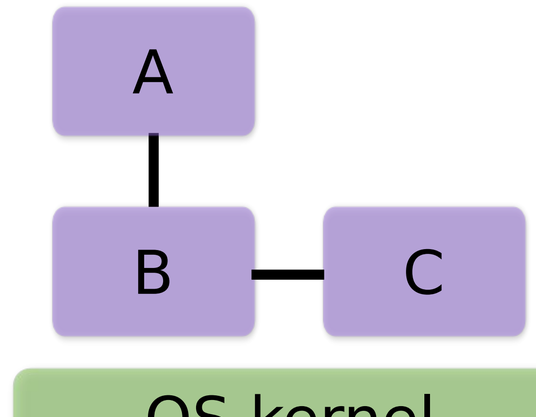
NICTA

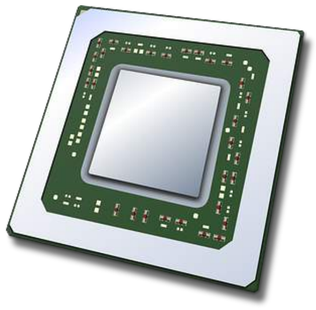




High Assurance Component-based Software

NICTA



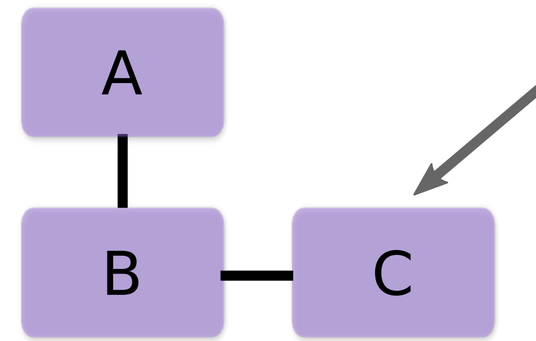


seL4

- 10,000 LoC microkernel
- functional correctness proof
- strong isolation properties
- fast and predictable

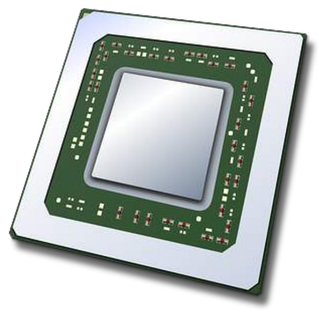
High Assurance Component-based Software

NICTA



components

- functionality
- policy
- >1 MLoC
- varied correctness criteria

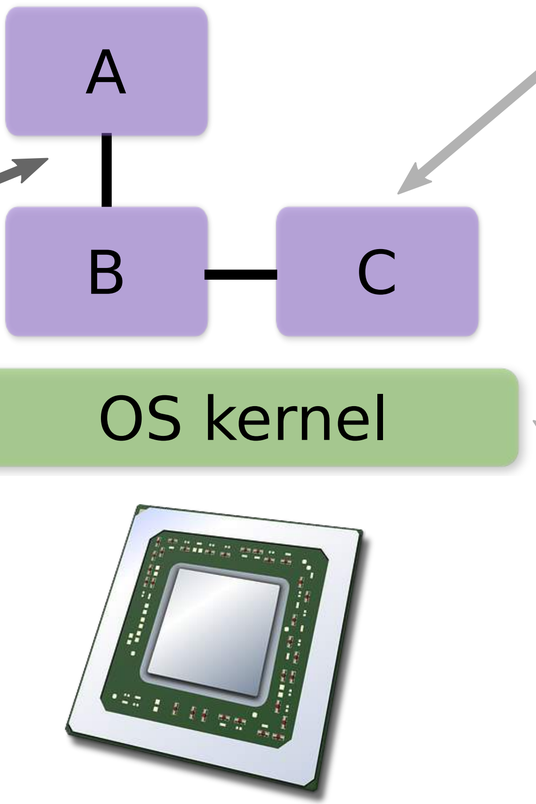


seL4

- 10,000 LoC microkernel
- functional correctness proof
- strong isolation properties
- fast and predictable

High Assurance Component-based Software

NICTA



components

- functionality
- policy
- >1 MLoC
- varied correctness criteria

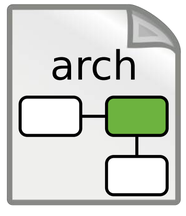
CAmkES

- static component platform
- formal architectural model
- optimised for seL4
- designed for verification

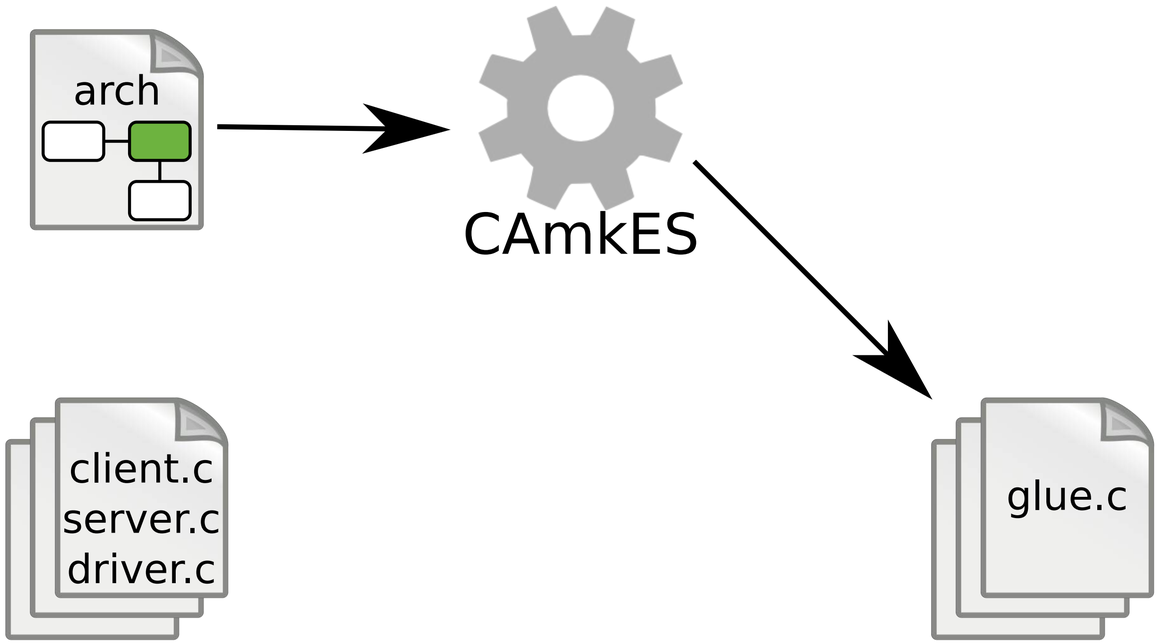
seL4

- 10,000 LoC microkernel
- functional correctness proof
- strong isolation properties
- fast and predictable

Component Compilation

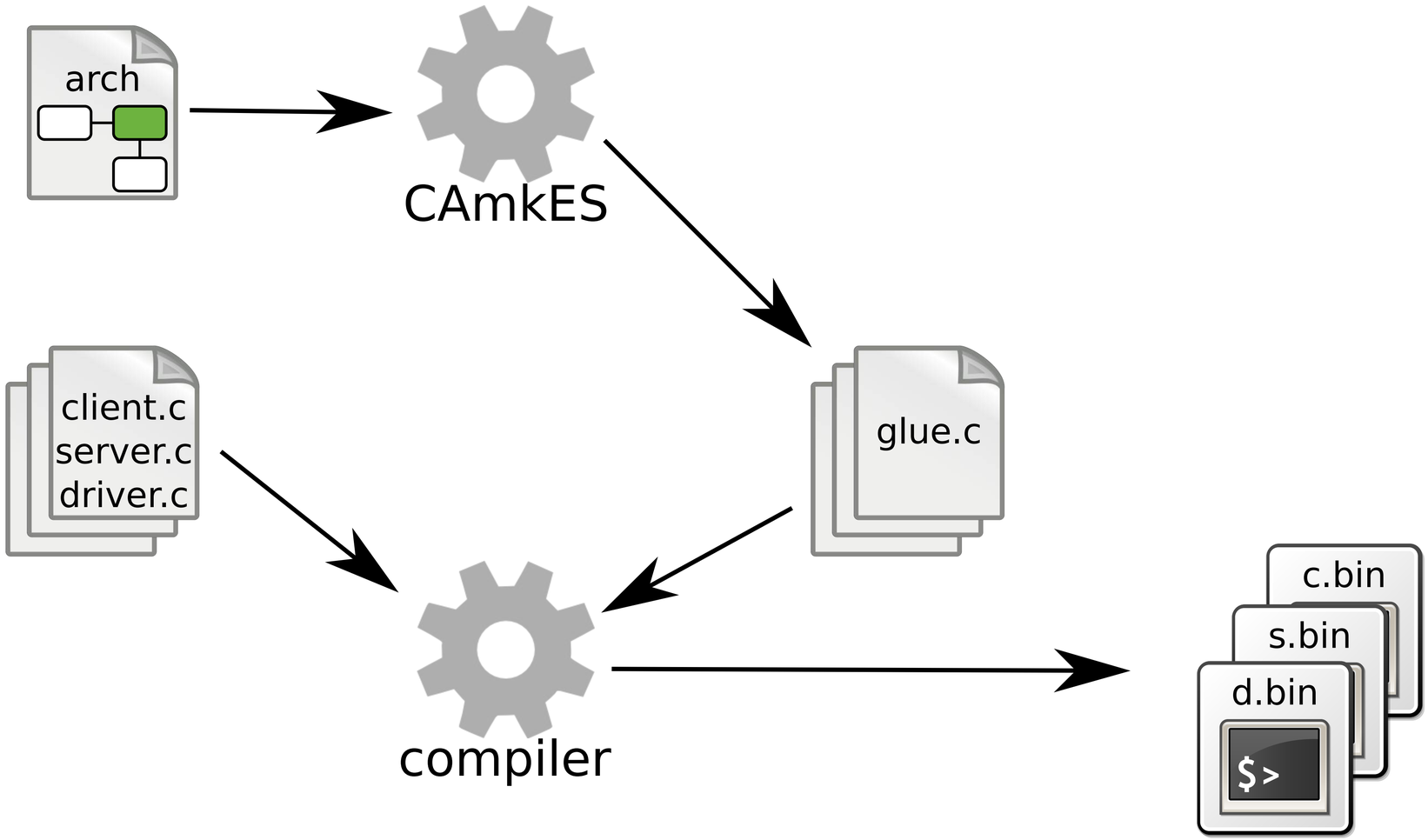


Component Compilation

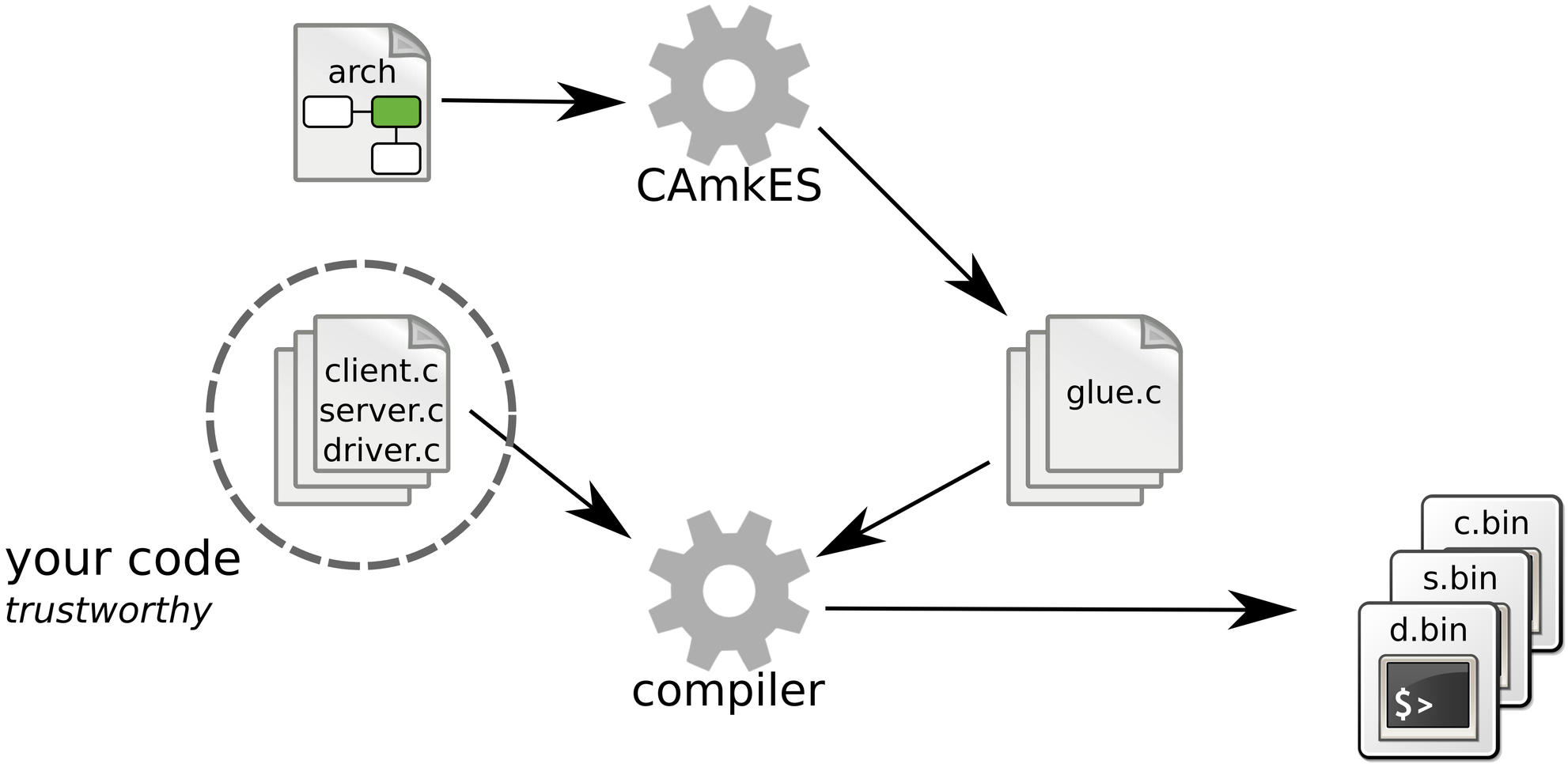


Component Compilation

NICTA

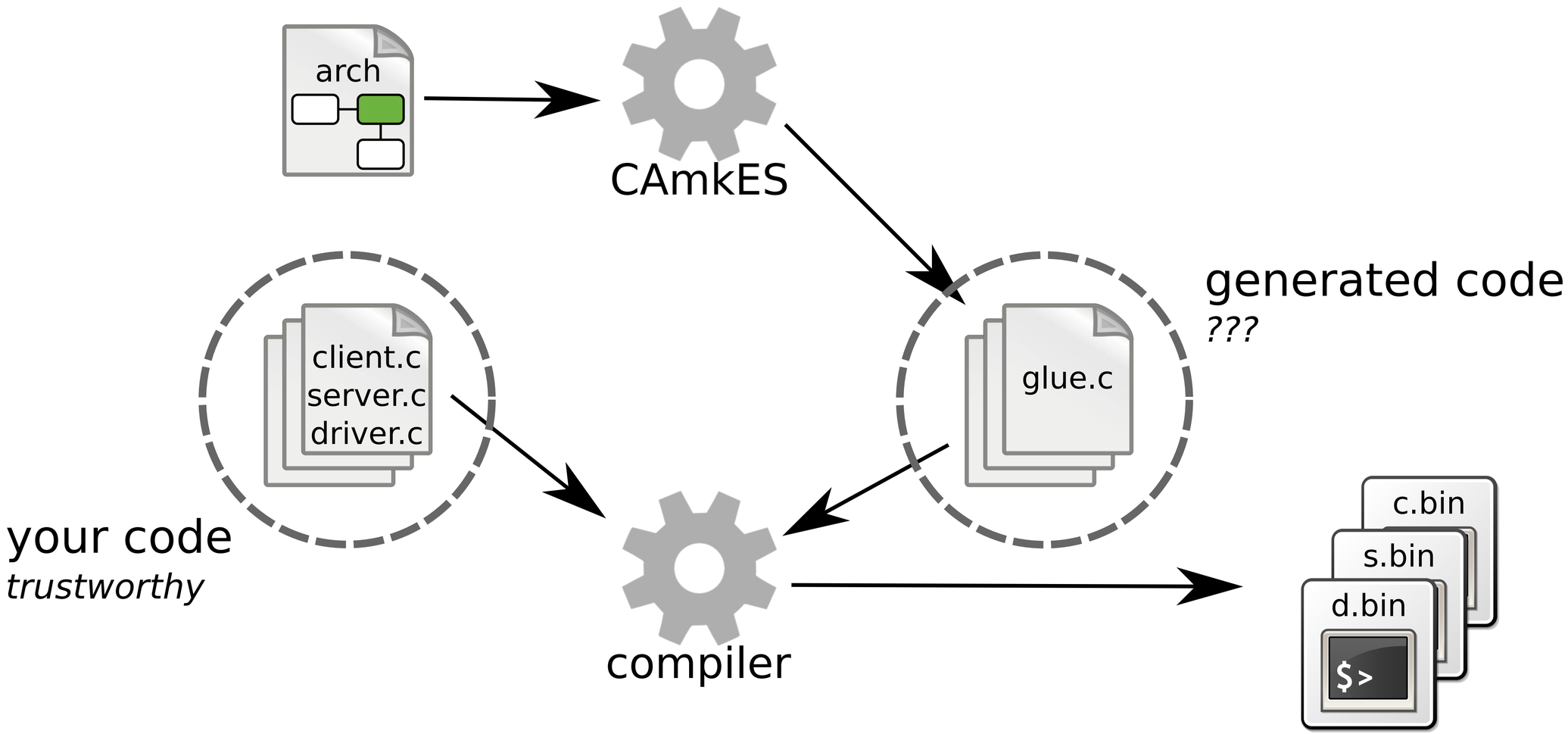


Component Compilation

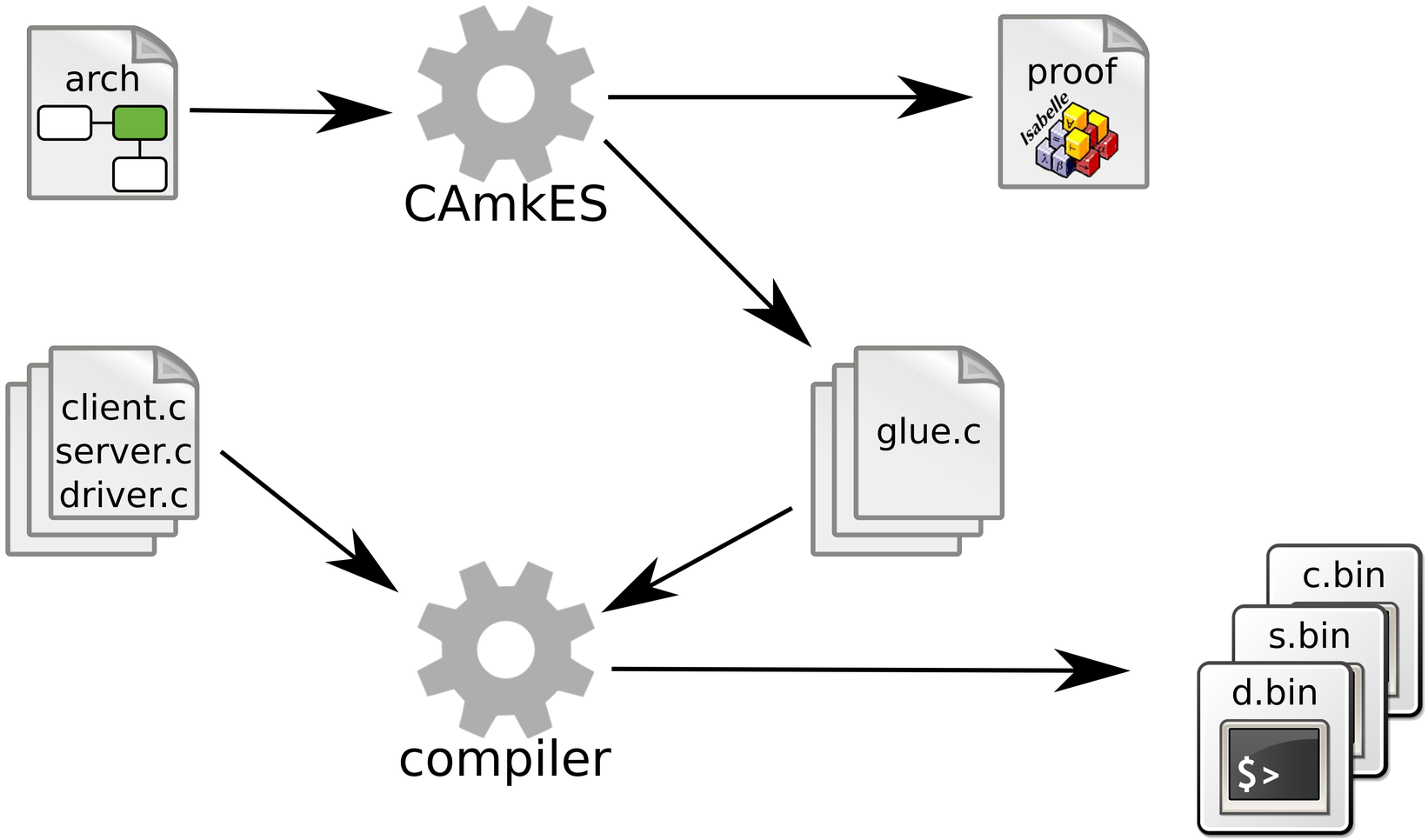


Component Compilation

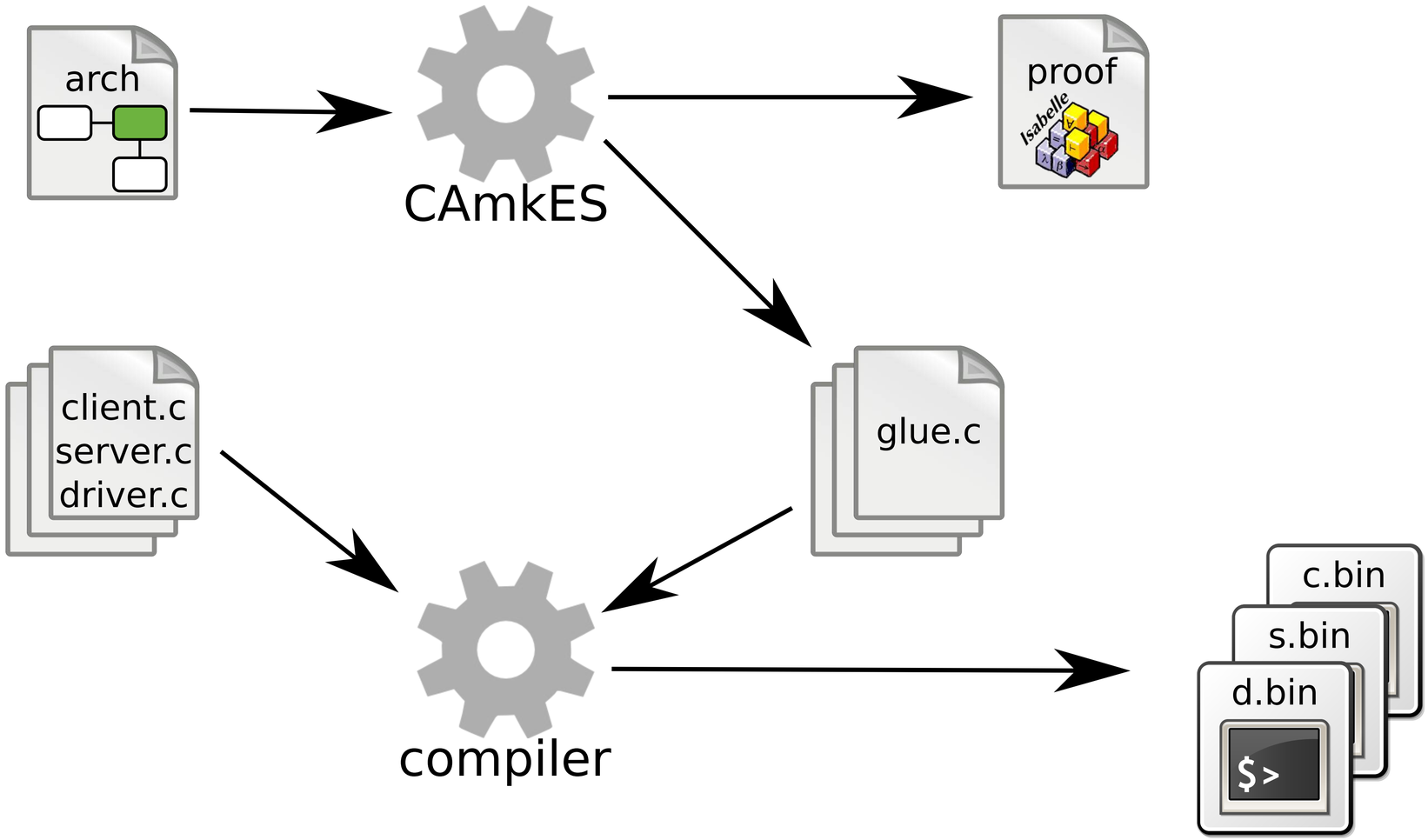
NICTA



Component Compilation



Component Compilation



generated proofs for generated code

Challenges

NICTA

What is correctness?

How do we prove it?

How do we automate it?

Challenges

NICTA

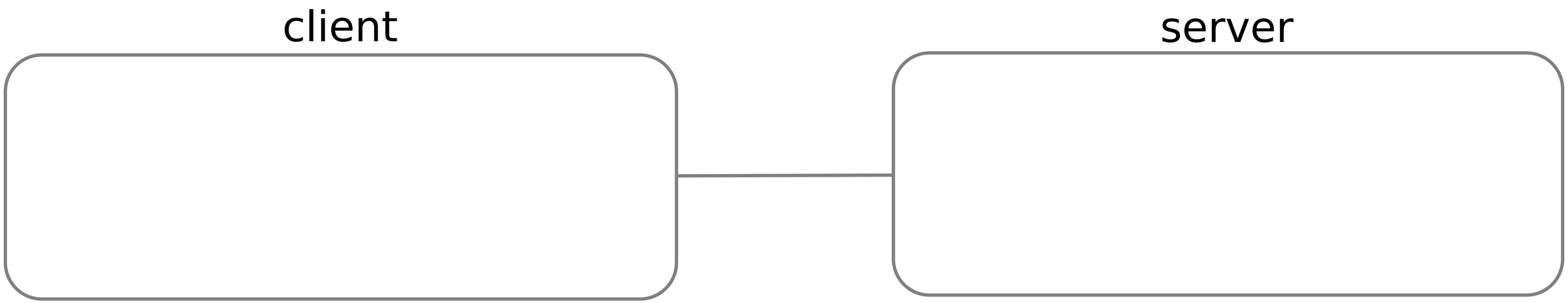
What is correctness?

How do we prove it?

How do we automate it?

Generated Code

NICTA



Generated Code

NICTA

client

```
void run() {
  int x = 2;
  int y = 3;
  int z = c_max(x, y);
  printf("max(%d, %d) = %d\n", x, y, z);
}
```

server

```
int s_max(int a, int b) {
  if (a > b) {
    return a;
  }
  return b;
}
```

Generated Code

NICTA

client

```
void run() {
  int x = 2;
  int y = 3;
  int z = c_max(x, y);
  printf("max(%d, %d) = %d\n", x, y, z);
}
```

server

```
int s_max(int a, int b) {
  if (a > b) {
    return a;
  }
  return b;
}
```

seL4

Generated Code

NICTA

client

```c
void run() {
  int x = 2;
  int y = 3;
  int z = c_max(x, y);
  printf("max(%d, %d) = %d\n", x, y, z);
}
```

server

```c
int s_max(int a, int b) {
  if (a > b) {
    return a;
  }
  return b;
}
```

```c
int c_max(int p, int q) {
  c_marshal(p, q);
  seL4_Call(...);
  int r = c_unmarshal();
  return r;
}
```

seL4

```c
void handle_rpc() {
  seL4_Wait(...);
  s_unmarshal(&i, &j);
  int k = s_max(i, j);
  s_marshal(k);
  seL4_Reply(...);
}
```

Generated Code

NICTA

client

```
void run() {
  int x = 2;
  int y = 3;
  int z = c_max(x, y);
  printf("max(%d, %d) = %d\n", x, y, z);
}
```

server

```
int s_max(int a, int b) {
  if (a > b) {
    return a;
  }
  return b;
}
```

```
int c_max(int p, int q) {
  c_marshal(p, q);
  seL4_Call(...);
  int r = c_unmarshal();
  return r;
}
```

seL4

```
void handle_rpc() {
  seL4_Wait(...);
  s_unmarshal(&i, &j);
  int k = s_max(i, j);
  s_marshal(k);
  seL4_Reply(...);
}
```

Generated Code

NICTA

client

```
void run() {
  int x = 2;
  int y = 3;
  int z = c_max(x, y);
  printf("max(%d, %d) = %d\n", x, y, z);
}
```

server

```
int s_max(int a, int b) {
  if (a > b) {
    return a;
  }
  return b;
}
```

```
int c_max(int p, int q) {
  c_marshal(p, q);
  seL4_Call(...);
  int r = c_unmarshal();
  return r;
}
```

seL4

```
void handle_rpc() {
  seL4_Wait(...);
  s_unmarshal(&i, &j);
  int k = s_max(i, j);
  s_marshal(k);
  seL4_Reply(...);
}
```

Generated Code

NICTA

client

```
void run() {
  int x = 2;
  int y = 3;
  int z = c_max(x, y);
  printf("max(%d, %d) = %d\n", x, y, z);
}
```

server

```
int s_max(int a, int b) {
  if (a > b) {
    return a;
  }
  return b;
}
```

```
int c_max(int p, int q) {
  c_marshal(p, q);
  seL4_Call(...);
  int r = c_unmarshal();
  return r;
}
```

seL4

```
void handle_rpc() {
  seL4_Wait(...);
  s_unmarshal(&i, &j);
  int k = s_max(i, j);
  s_marshal(k);
  seL4_Reply(...);
}
```

Generated Code

NICTA

client

```
void run() {
  int x = 2;
  int y = 3;
  int z = c_max(x, y);
  printf("max(%d, %d) = %d\n", x, y, z);
}
```

server

```
int s_max(int a, int b) {
  if (a > b) {
    return a;
  }
  return b;
}
```

```
int c_max(int p, int q) {
  c_marshal(p, q);
  seL4_Call(...);
  int r = c_unmarshal();
  return r;
}
```

seL4

```
void handle_rpc() {
  seL4_Wait(...);
  s_unmarshal(&i, &j);
  int k = s_max(i, j);
  s_marshal(k);
  seL4_Reply(...);
}
```

Generated Code

NICTA

client

```
void run() {
  int x = 2;
  int y = 3;
  int z = c_max(x, y);
  printf("max(%d, %d) = %d\n", x, y, z);
}
```

```
int c_max(int p, int q) {
  c_marshal(p, q);
  seL4_Call(...);
  int r = c_unmarshal();
  return r;
}
```

seL4

{[**P**]}

server

```
int s_max(int a, int b) {
  if (a > b) {
    return a;
  }
  return b;
}
```

{[**Q**]}

```
void handle_rpc() {
  seL4_Wait(...);
  s_unmarshal(&i, &j);
  int k = s_max(i, j);
  s_marshal(k);
  seL4_Reply(...);
}
```

Generated Code

NICTA

client

```
void run() {
  int x =
  int y =
  int z = c_max(x, y);
  printf(                d) = %d\n", x, y, z);
}
```

{[P]}

{[Q]}

server

{[P]}

```
int s_max(int a, int b) {
  if (a > b) {
    return a;
  }
  return b;
}
```

{[Q]}

```
int c_max(int p, int q) {
  c_marshal(p, q);
  seL4_Call(...);
  int r = c_unmarshal();
  return r;
}
```

seL4

```
void handle_rpc() {
  seL4_Wait(...);
  s_unmarshal(&i, &j);
  int k = s_max(i, j);
  s_marshal(k);
  seL4_Reply(...);
}
```

Generated Code

NICTA

client

```
void run() {
  int x =
  int y =
  int z = c_max(x, y);
  printf(                d) = %d\n", x, y, z);
}
```

{[P]}

{[Q]}

server

{[P]}

```
int s_max(int a, int b) {
  if (a > b) {
    return a;
  }
  return b;
}
```

{[Q]}

```
int c_max(int p, int q) {
  c_marshal(p, q);
  seL4_Call(...);
  int r = c_unmarshal();
  return r;
}
```

seL4

```
void handle_rpc() {
  seL4_Wait(...);
  s_unmarshal(&i, &j);
  int k = s_max(i, j);
  s_marshal(k);
  seL4_Reply(...);
}
```

RPC invocation ≅ local invocation

Challenges

NICTA

What is correctness?

How do we prove it?

How do we automate it?

Toolchain (prior work)

NICTA

```
int max(int a, int b) {
  if (a > b) {
    return a;
  }
  return b;
}
```

Toolchain (prior work)

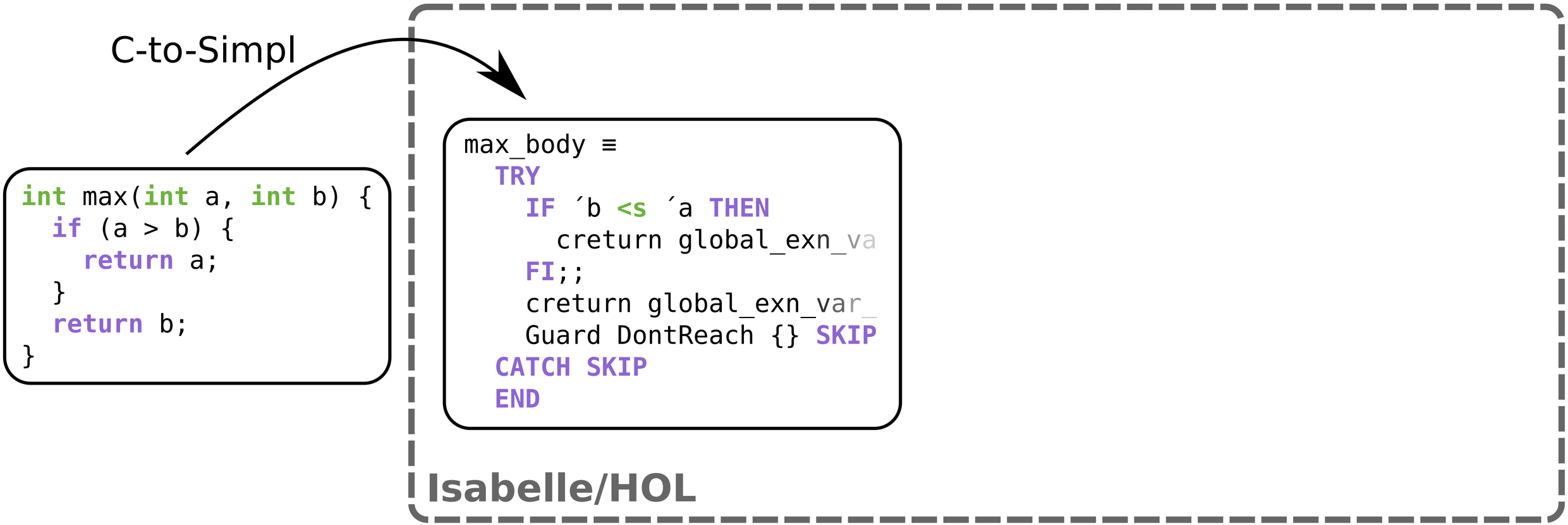
NICTA

- **Isabelle/HOL** — LCF-style interactive theorem prover

Toolchain (prior work)

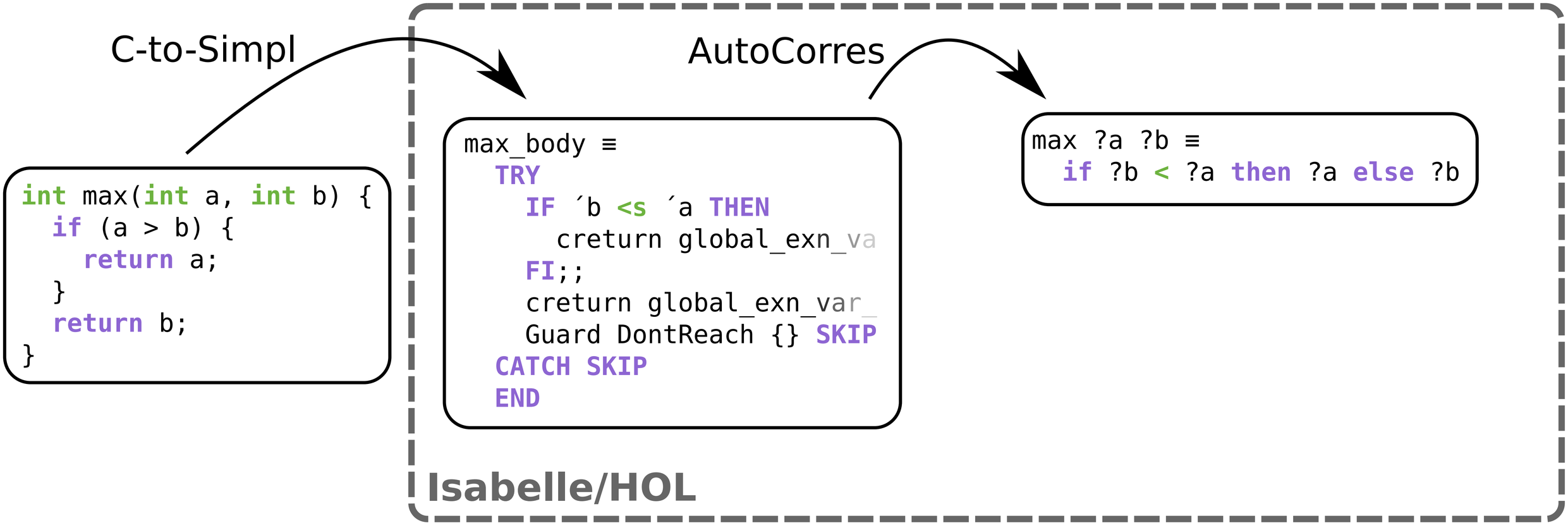
NICTA

- **Isabelle/HOL** — LCF-style interactive theorem prover
- **C-to-Simpl Parser** — Semantics-preserving translation of C to Simpl



Toolchain (prior work)

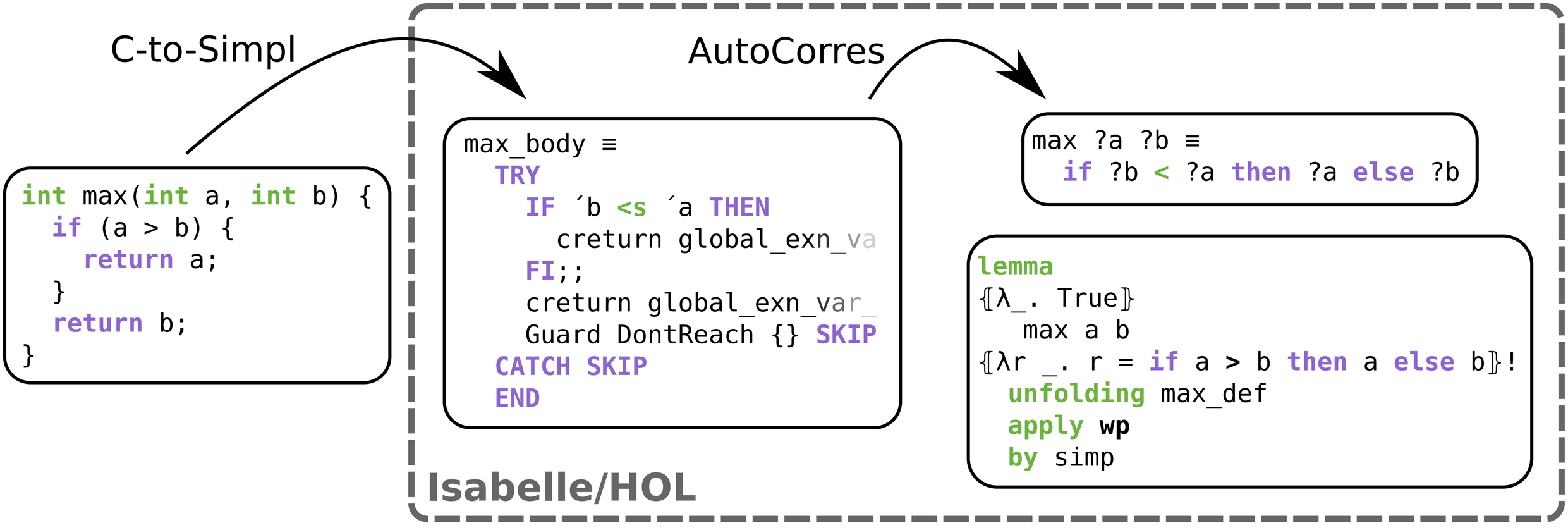
- **Isabelle/HOL** — LCF-style interactive theorem prover
- **C-to-Simpl Parser** — Semantics-preserving translation of C to Simpl
- **AutoCorres** — Verified abstraction of Simpl



Toolchain (prior work)

NICTA

- **Isabelle/HOL** — LCF-style interactive theorem prover
- **C-to-Simpl Parser** — Semantics-preserving translation of C to Simpl
- **AutoCorres** — Verified abstraction of Simpl



Toolchain (prior work)

NICTA

- **Isabelle/HOL** — LCF-style interactive theorem prover
- **C-to-Simpl Parser** — Semantics-preserving translation of C to Simpl
- **AutoCorres** — Verified abstraction of Simpl
- **WP** — weakest-precondition reasoning for Hoare triples

C-to-Simpl

```
int max(int a, int b) {
  if (a > b) {
    return a;
  }
  return b;
}
```

AutoCorres

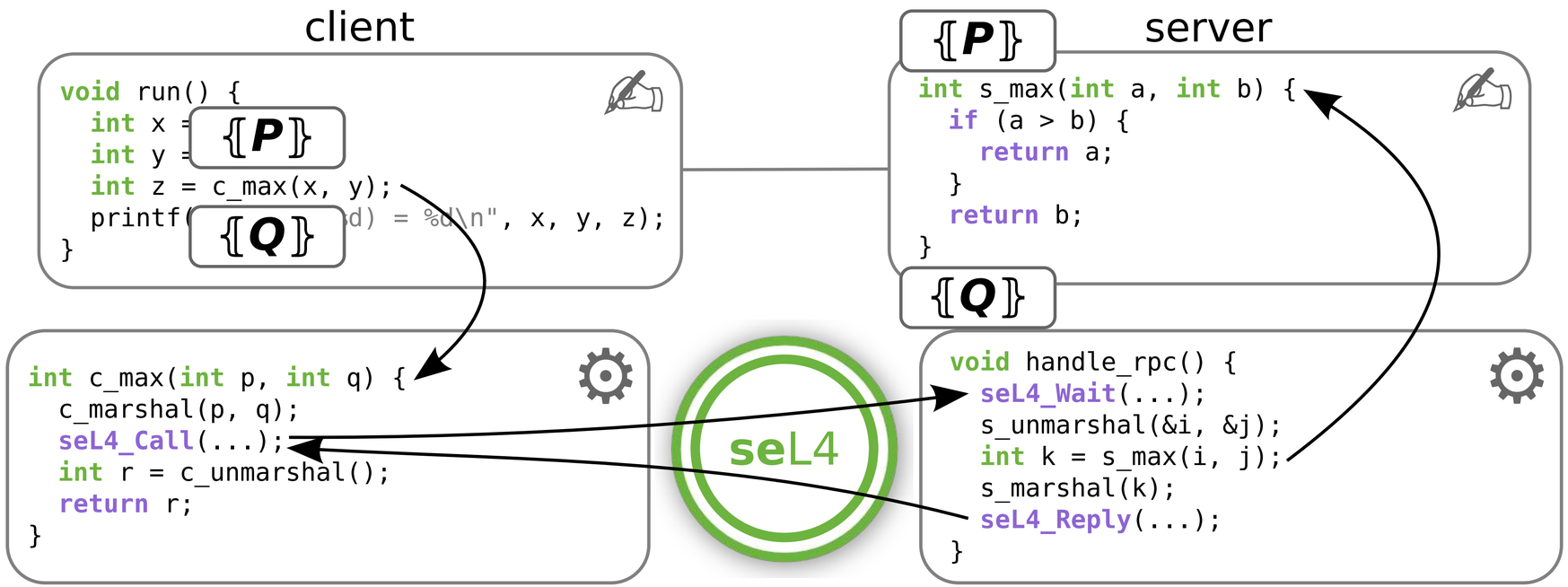
```
max_body ≡
  TRY
    IF ´b <s ´a THEN
      creturn global_exn_va
    FI;;
    creturn global_exn_var_
    Guard DontReach {} SKIP
  CATCH SKIP
  END
```

```
max ?a ?b ≡
  if ?b < ?a then ?a else ?b
```

```
lemma
⦇λ_. True⦈
   max a b
⦇λr _. r = if a > b then a else b⦈!
  unfolding max_def
  apply wp
  by simp
```

Isabelle/HOL

Verifying Generated Code

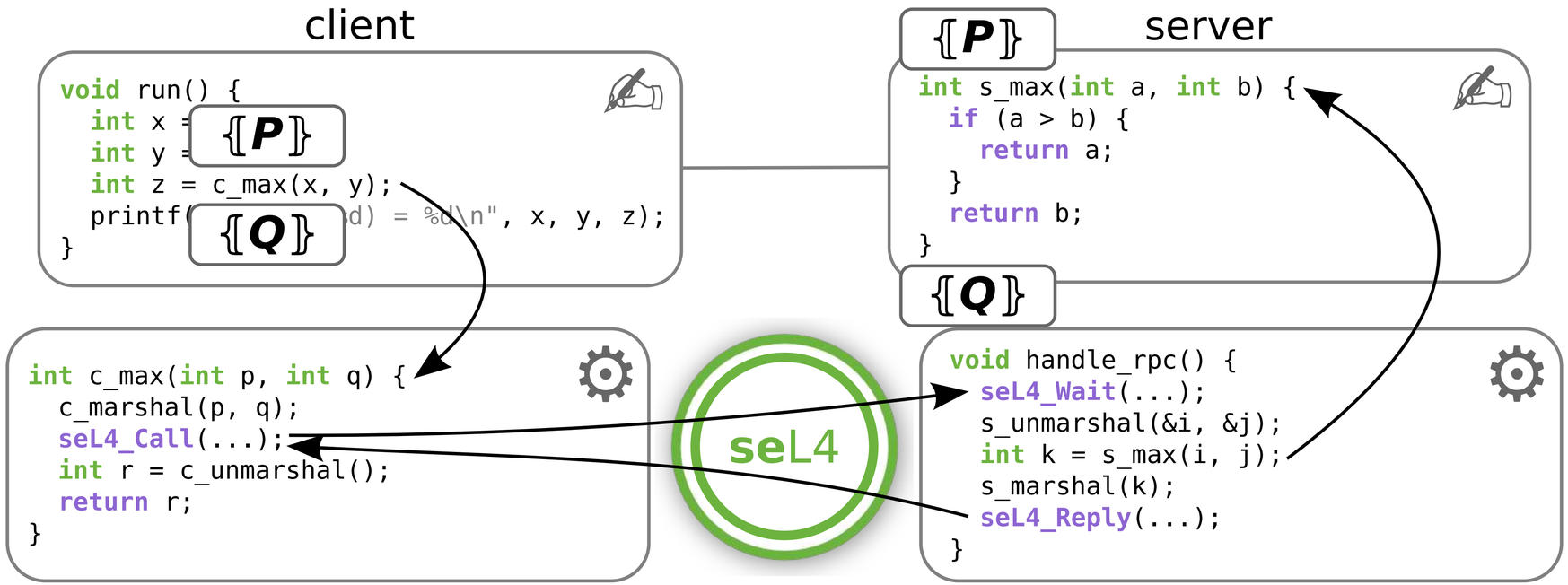


RPC invocation ≅ local invocation

Verifying Generated Code

NICTA

$$\{\!|P|\!\}\ s\_max\ a\ b\ \{\!|Q|\!\}! \quad \Rightarrow \quad \{\!|P|\!\}\ c\_max\ a\ b\ \{\!|Q|\!\}!$$

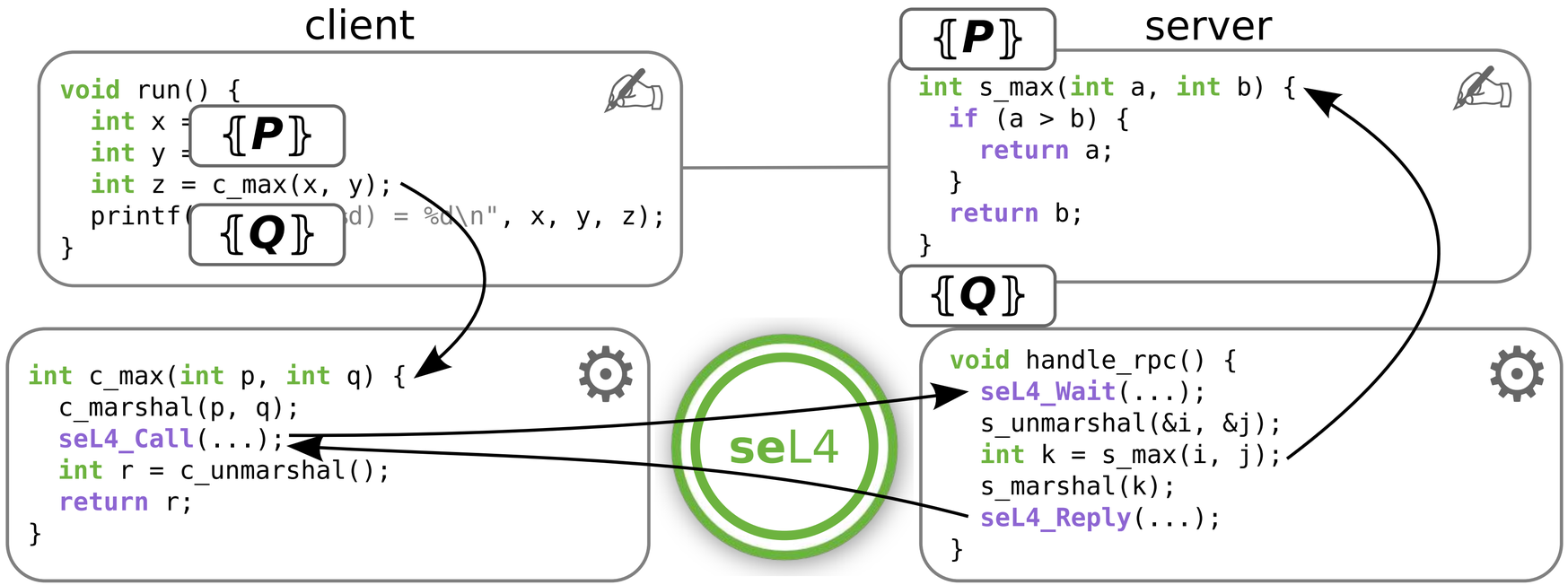


RPC invocation ≅ local invocation

Verifying Generated Code

NICTA

$$\{\!|P|\!\}\ s\_max\ a\ b\ \{\!|Q|\!\}! \quad \Rightarrow \quad \{\!|P|\!\}\ \textbf{do}\ c\_marshall\ a\ b;\ seL4\_Call\ ...;\ c\_unmarshal\ r\ \textbf{od}\ \{\!|Q|\!\}!$$



RPC invocation ≅ local invocation

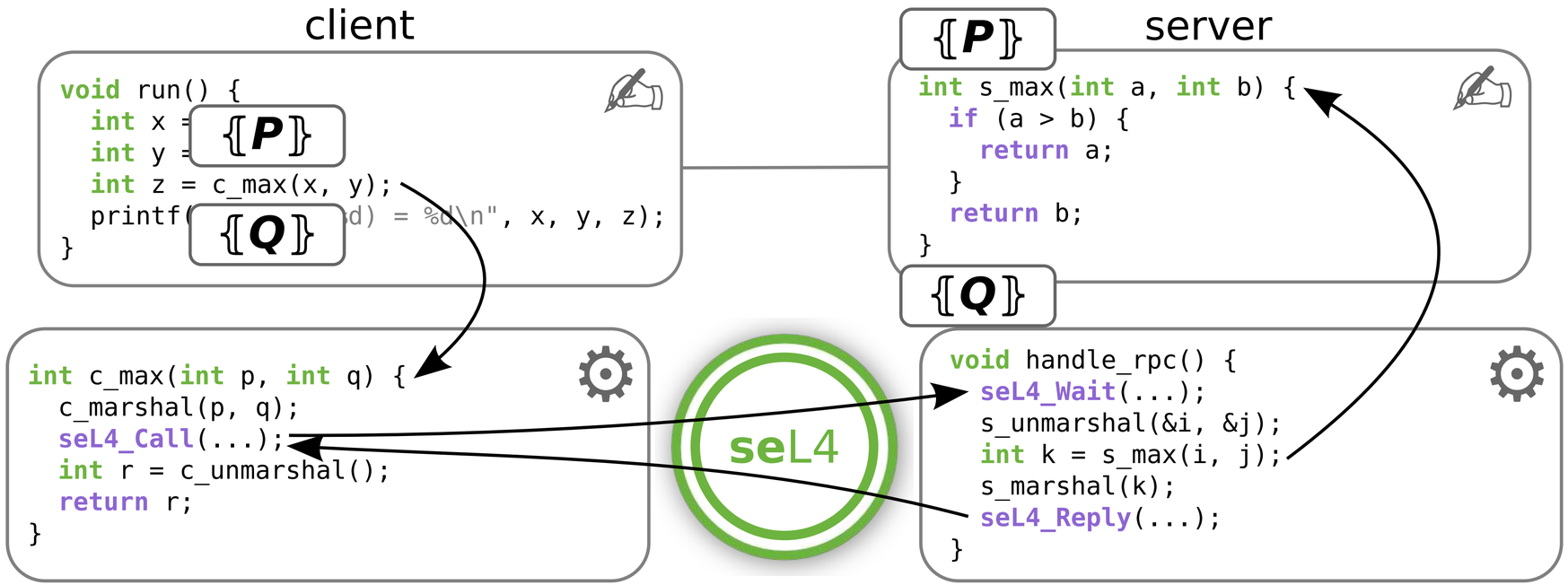
Verifying Generated Code

NICTA

$$\{\!|P|\!\}\ s\_max\ a\ b\ \{\!|Q|\!\}! \quad \Rightarrow \quad \{\!|P|\!\}$$

```
do c_marshall a b;
   s_unmarshal a b;
   r ← s_max a b;
   s_marshal r;
   c_unmarshal r
od
```

$\{\!|Q|\!\}!$



RPC invocation ≅ local invocation

Challenges

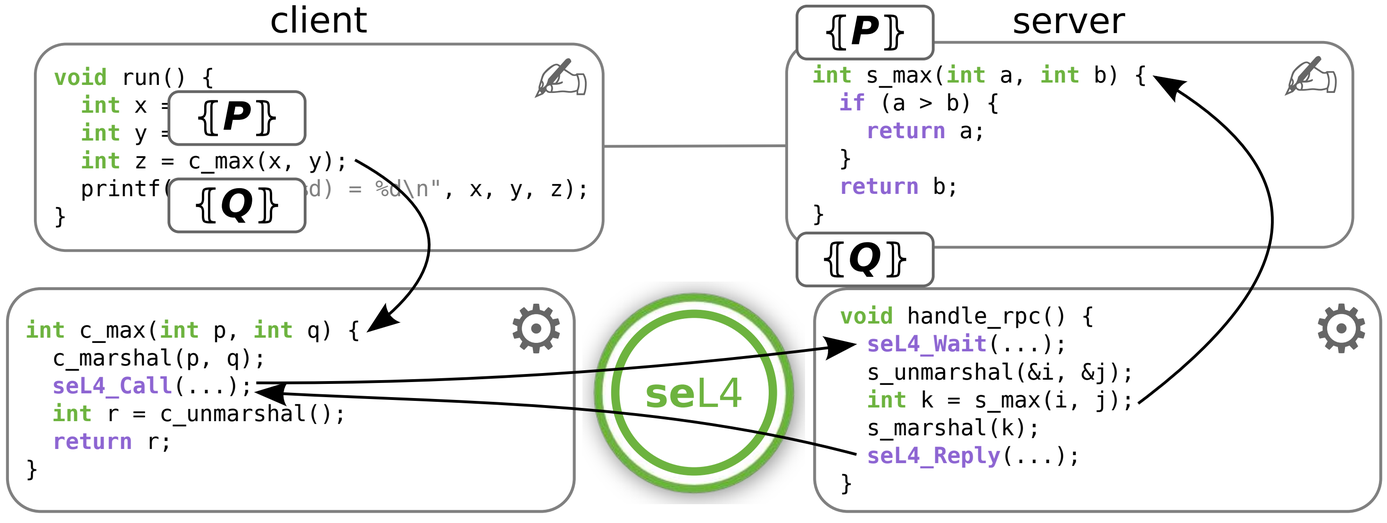
NICTA

What is correctness?

How do we prove it?

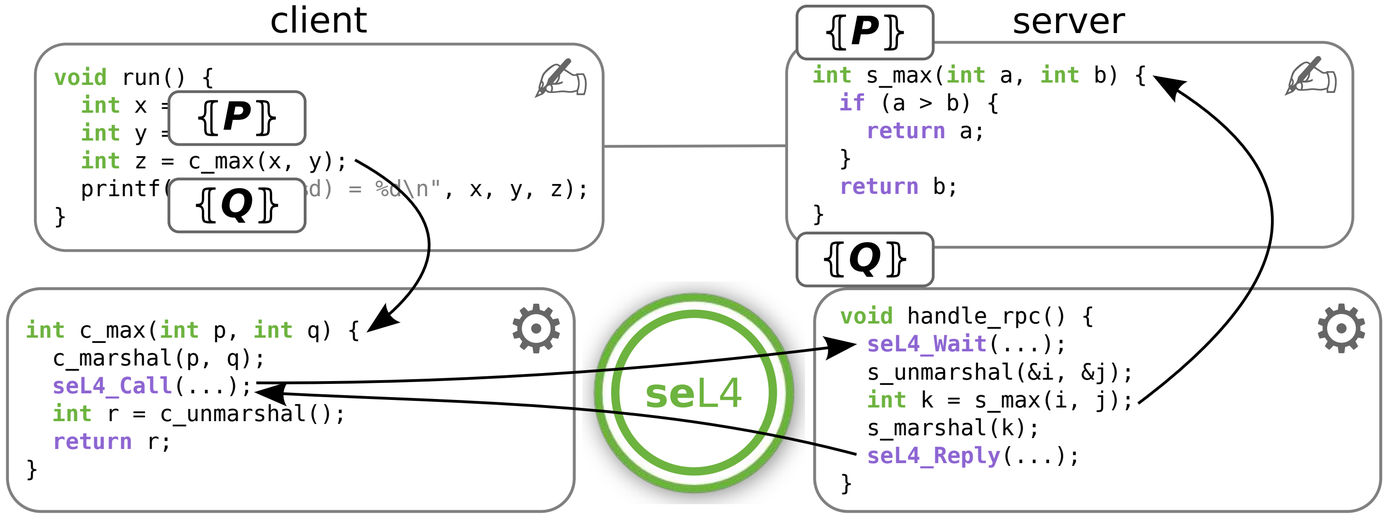
How do we automate it?

Generated Proofs



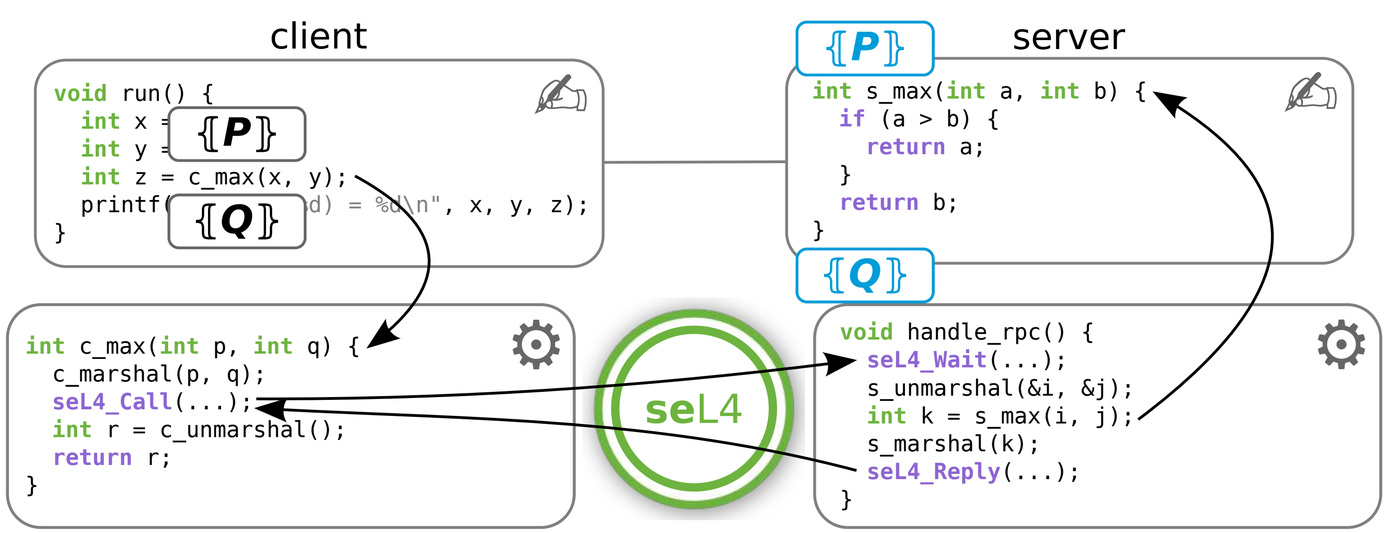
Generated Proofs

locale rpcstubs$_{max}$ =



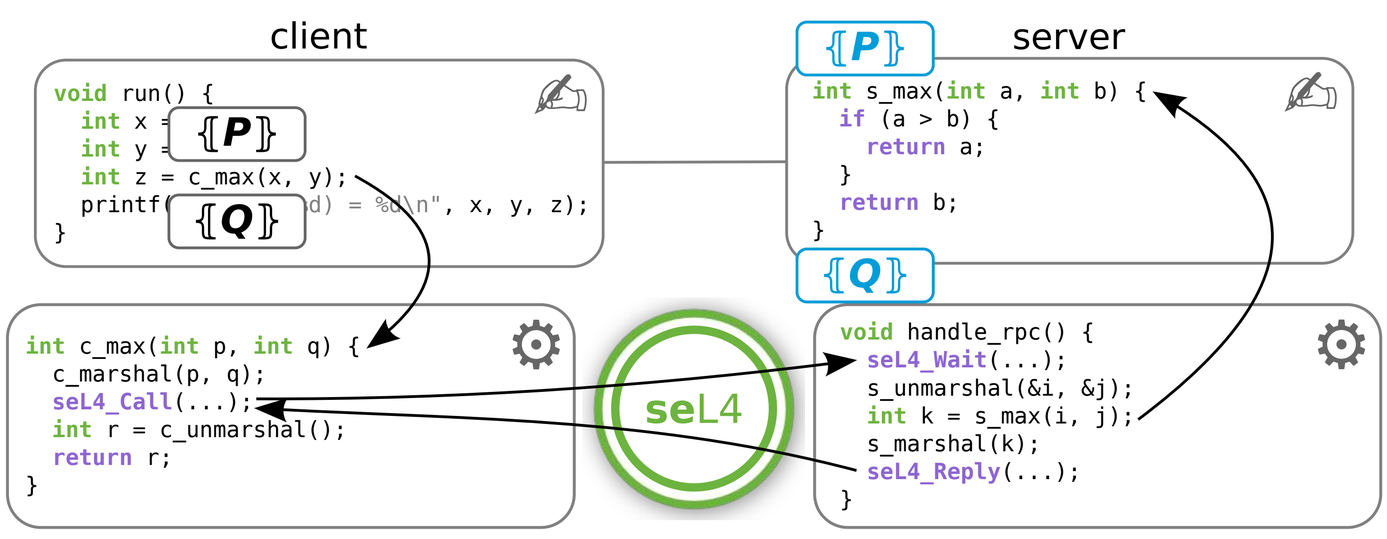
Generated Proofs

locale rpcstubs$_{max}$ =
  fixes P_{max} :: *lifted_globals ⇒ int ⇒ int ⇒ bool*
  fixes Q_{max} :: *lifted_globals ⇒ lifted_globals ⇒ int ⇒ int ⇒ int ⇒ bool*
  assumes
    ⦃λs. P_{max} *s a b⦄*
      s_max a b
    ⦃λr s. Q_{max} *s0 s r a b⦄!*



Generated Proofs

locale rpcstubs$_{max}$ =
  fixes P_{max} :: *lifted_globals ⇒ int ⇒ int ⇒ bool*
  fixes Q_{max} :: *lifted_globals ⇒ lifted_globals ⇒ int ⇒ int ⇒ int ⇒ bool*
  assumes
    ⦃λs. s = s0 ∧ inv s ∧ P_{max} s a b⦄
      s_max a b
    ⦃λr s. inv s ∧ Q_{max} s0 s r a b⦄!



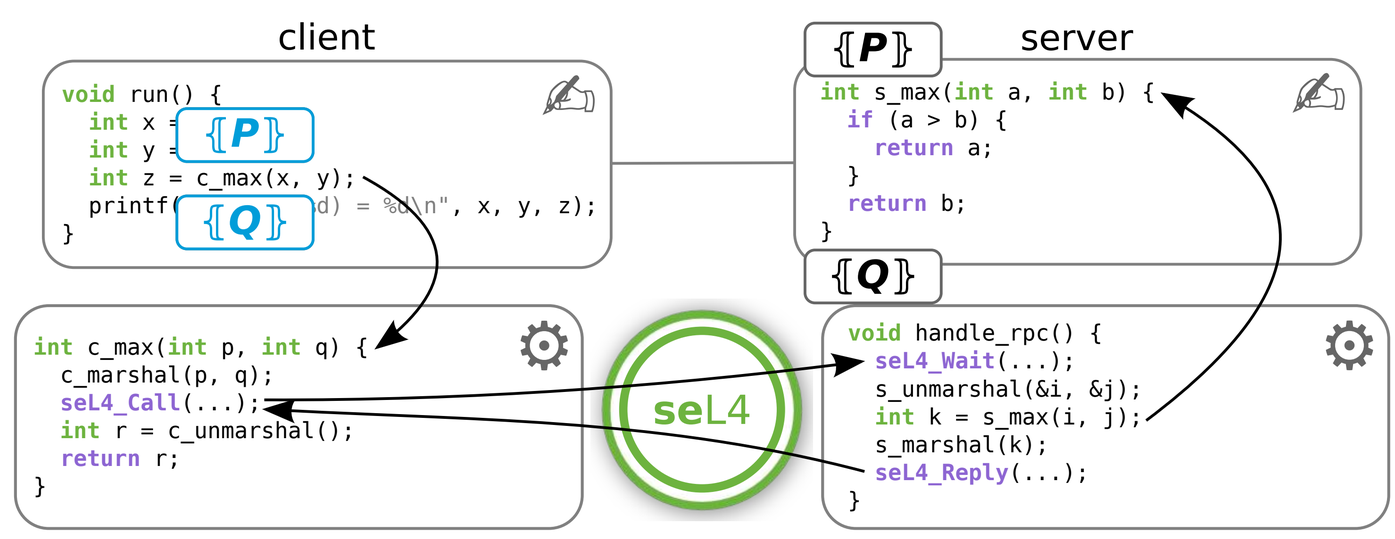
Generated Proofs

locale rpcstubs_{max} =
  fixes P_{max} :: *lifted_globals* ⇒ *int* ⇒ *int* ⇒ *bool*
  fixes Q_{max} :: *lifted_globals* ⇒ *lifted_globals* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *bool*
  assumes
    {|λs. s = s0 ∧ inv s ∧ P_{max} s a b|}
      s_max a b
    {|λr s. inv s ∧ Q_{max} s0 s r a b|}!

theorem

{

}



{

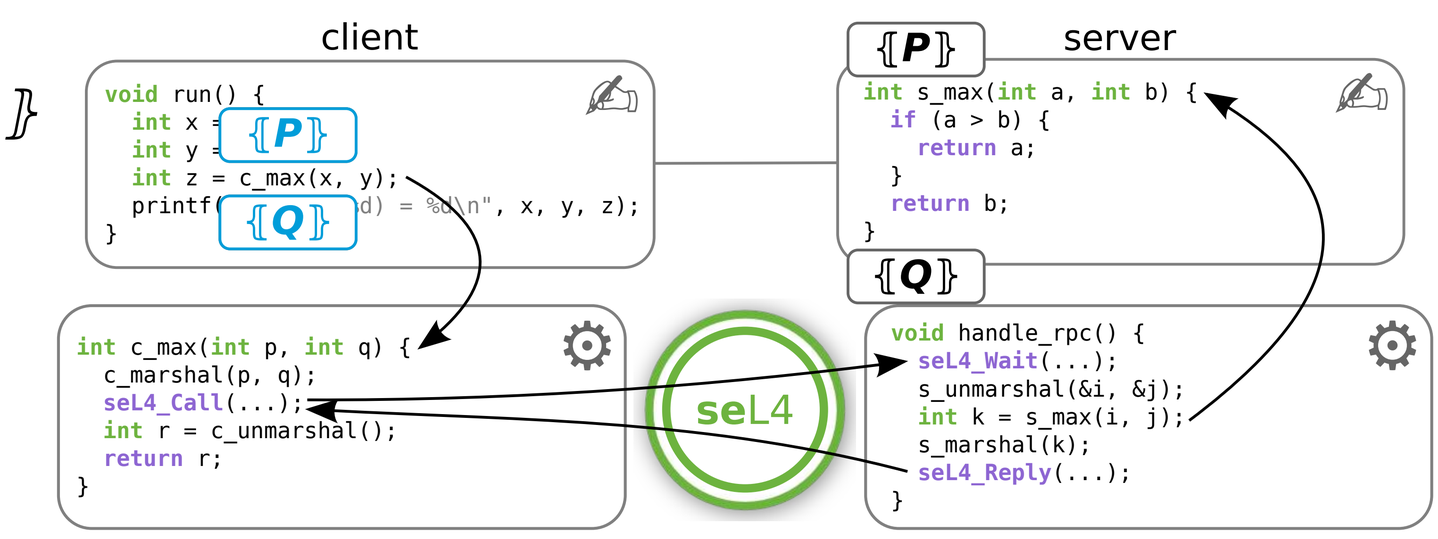
}!

Generated Proofs

locale rpcstubs$_{max}$ =
  fixes P_{max} :: *lifted_globals ⇒ int ⇒ int ⇒ bool*
  fixes Q_{max} :: *lifted_globals ⇒ lifted_globals ⇒ int ⇒ int ⇒ int ⇒ bool*
  assumes
    {|λs. s = s0 ∧ inv s ∧ P_{max} s a b|}
      s_max a b
    {|λr s. inv s ∧ Q_{max} s0 s r a b|}!

theorem
  {
    do *c_marshal a b;*
      s_unmarshal a b;
      r ← s_max a b;
      s_marshal r;
      c_unmarshal r
    od
  {

}

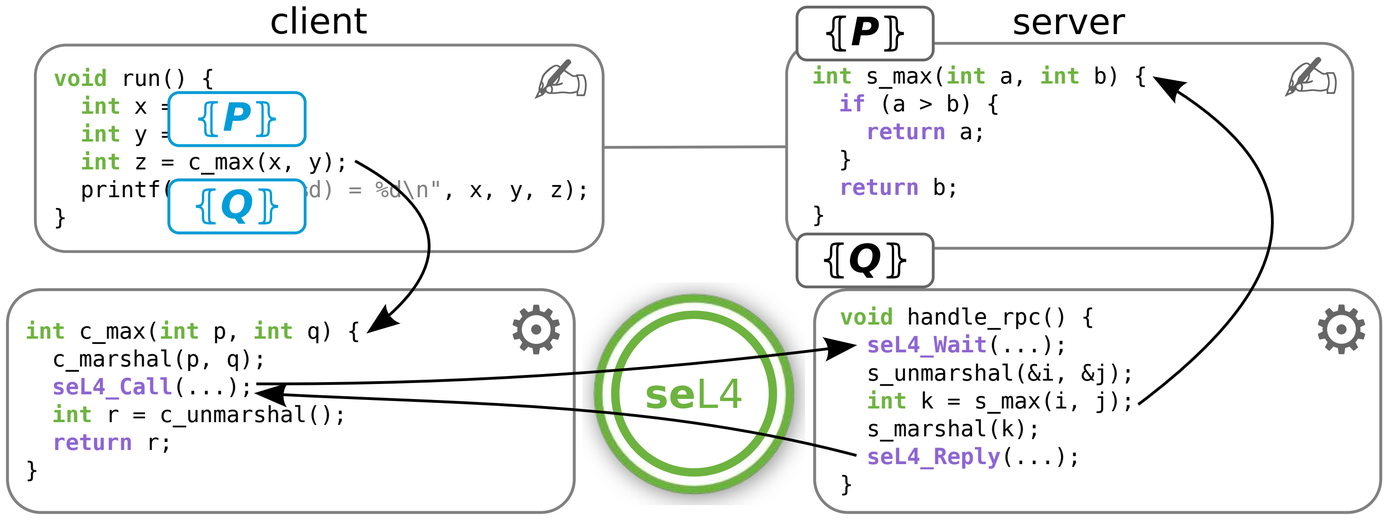


}!

Generated Proofs

locale rpcstubs$_{max}$ =
  fixes P_{max} :: *lifted_globals ⇒ int ⇒ int ⇒ bool*
  fixes Q_{max} :: *lifted_globals ⇒ lifted_globals ⇒ int ⇒ int ⇒ int ⇒ bool*
  assumes
    ⦃λs. s = s0 ∧ inv s ∧ P_{max} s a b⦄
      s_max a b
    ⦃λr s. inv s ∧ Q_{max} s0 s r a b⦄!

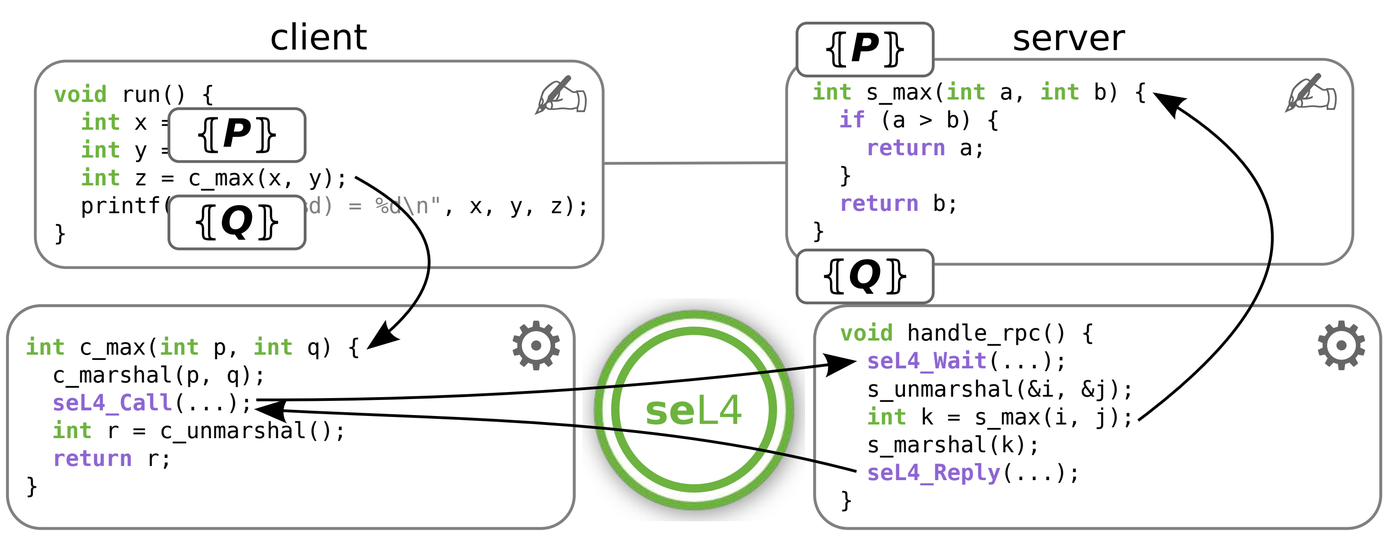
theorem
⦃λs. s = s0 ∧ inv s ∧ P_{max} s a b⦄
  do *c_marshal a b;*
    s_unmarshal a b;
    r ← s_max a b;
    s_marshal r;
    c_unmarshal r
  od
⦃λr s. inv s ∧ Q_{max} s0 s r a b⦄!



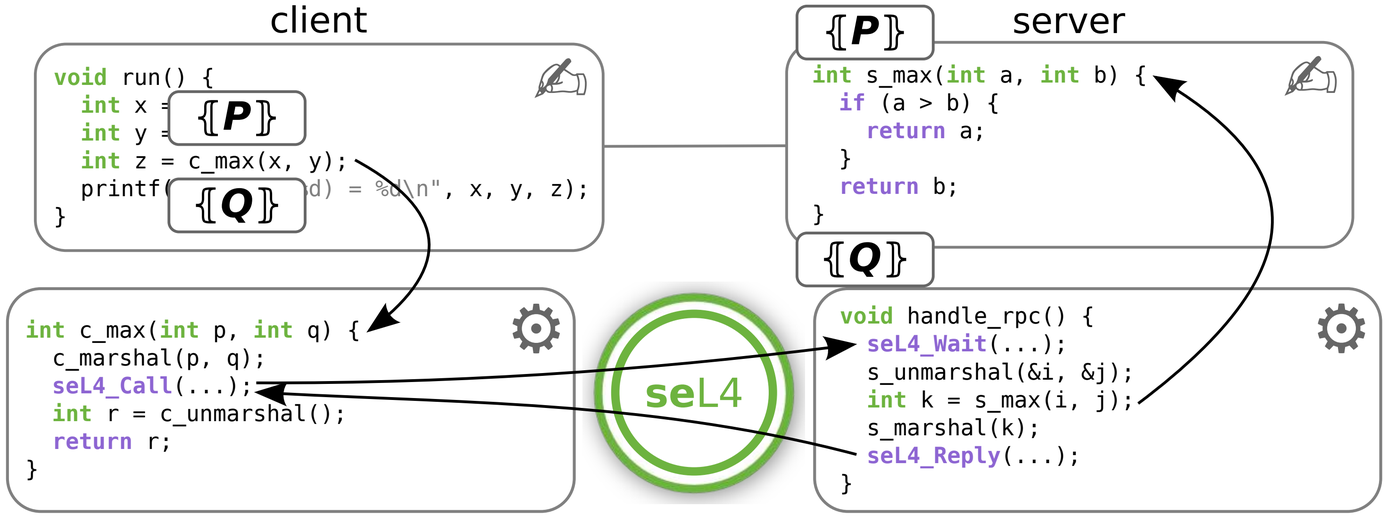
Generated Proofs

locale rpcstubs$_{max}$ =
  fixes P_{max} :: *lifted_globals ⇒ int ⇒ int ⇒ bool*
  fixes Q_{max} :: *lifted_globals ⇒ lifted_globals ⇒ int ⇒ int ⇒ int ⇒ bool*
  assumes
    ⦃λs. s = s0 ∧ inv s ∧ P_{max} s a b⦄
      s_max a b
    ⦃λr s. inv s ∧ Q_{max} s0 s r a b⦄!

theorem
  ⦃λs. s = s0 ∧ inv s ∧ P_{max} s a b⦄
    do *c_marshal a b;*
      s_unmarshal a b;
      r ← s_max a b;
      s_marshal r;
      c_unmarshal r
    od
  ⦃λr s. inv s ∧ Q_{max} s0 s r a b⦄!
    generated proof



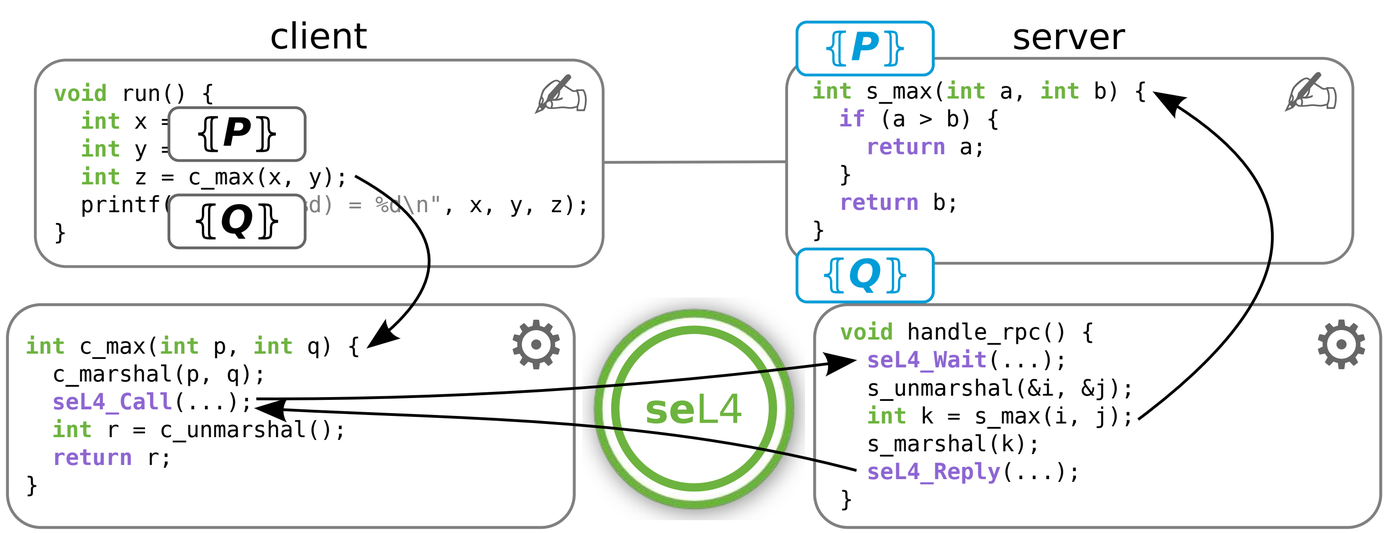
Composing Proofs



Composing Proofs

theorem

$$\{\!|\lambda\_.\ True|\!\}$$
$$\quad s\_max\ a\ b$$
$$\{\!|\lambda r\ \_.\ r = \textbf{if}\ a > b\ \textbf{then}\ a\ \textbf{else}\ b|\!\}!$$



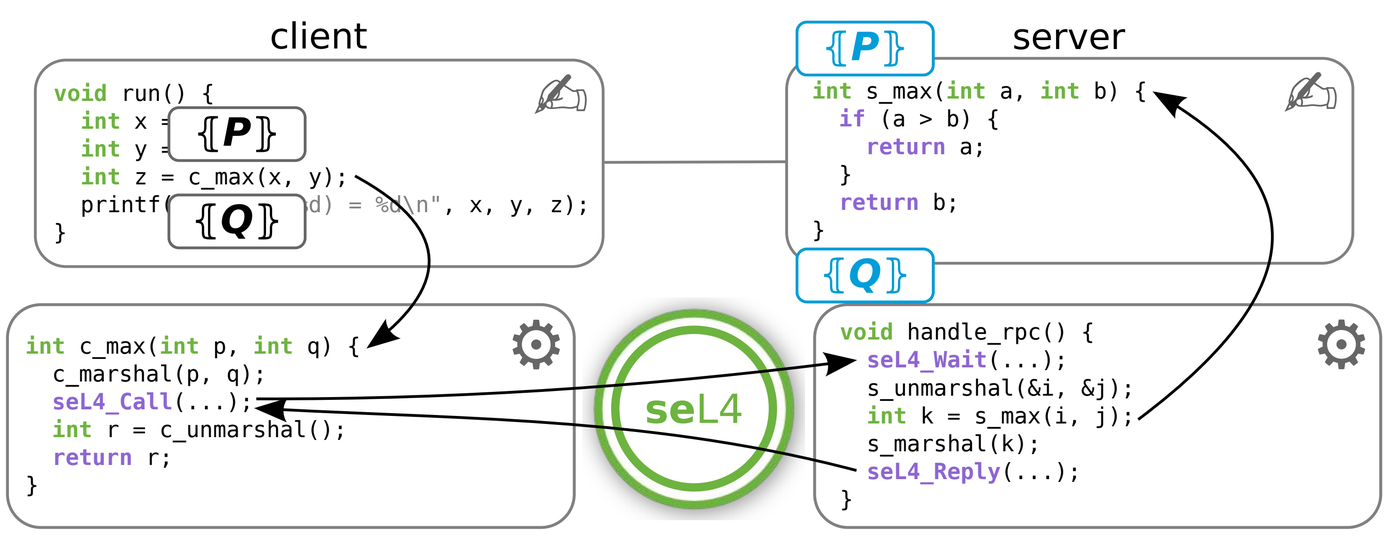
Composing Proofs

theorem

{⟦λ_. True⟧}
* s_max a b*
*{⟦λr _. r = **if** a > b **then** a **else** b⟧}!*

interpretation $rpcstubs_{max}$

λ_ _ _. True -- P_{max}
*λ_ _ r a b. r = **if** a > b **then** a **else** b* -- Q_{max}



Composing Proofs

NICTA

theorem

$\{\!|\lambda\_.\ True|\!\}$
  s_max a b
$\{\!|\lambda r\ \_.\ r = \textbf{if}\ a > b\ \textbf{then}\ a\ \textbf{else}\ b|\!\}!$

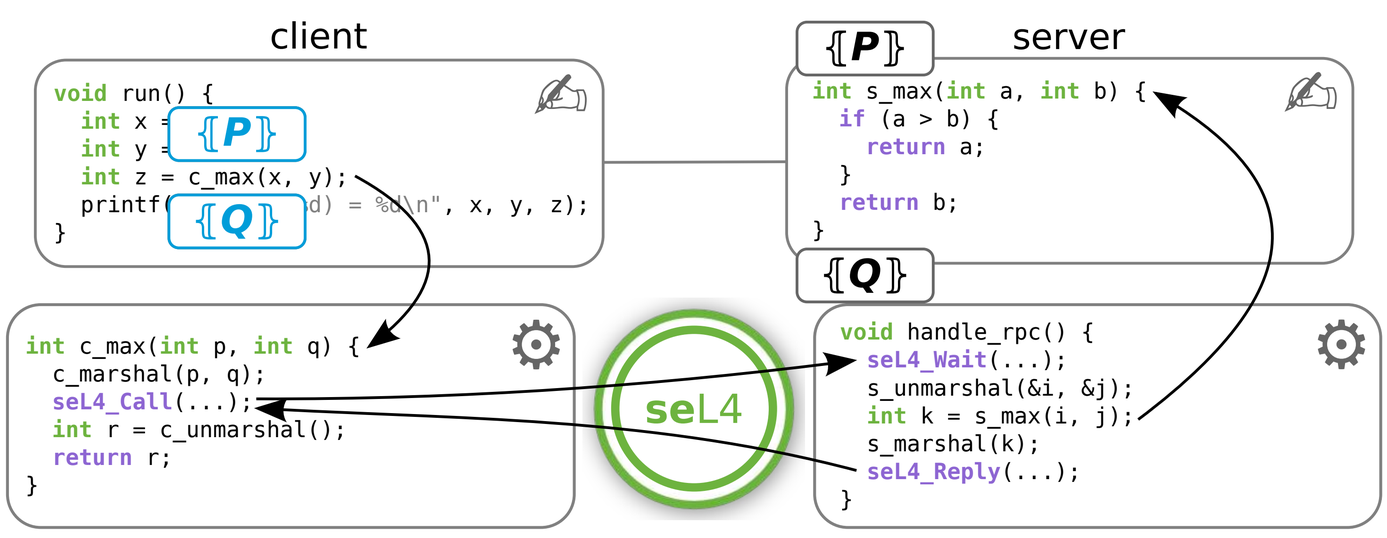
interpretation $rpcstubs_{max}$

$\lambda\_\ \_\ \_.\ True$ -- P_{max}

$\lambda\_\ \_\ r\ a\ b.\ r = \textbf{if}\ a > b\ \textbf{then}\ a\ \textbf{else}\ b$ -- Q_{max}

theorem

$\{\!|\lambda s.\ inv\ s|\!\}$
  do *c_marshal a b;*
    s_unmarshal a b;
    r ← s_max a b;
    s_marshal r;
    c_unmarshal r
  od
$\{\!|\lambda r\ s.\ inv\ s \wedge r = \textbf{if}\ a > b\ \textbf{then}\ a\ \textbf{else}\ b|\!\}!$



Composing Proofs

NICTA

theorem

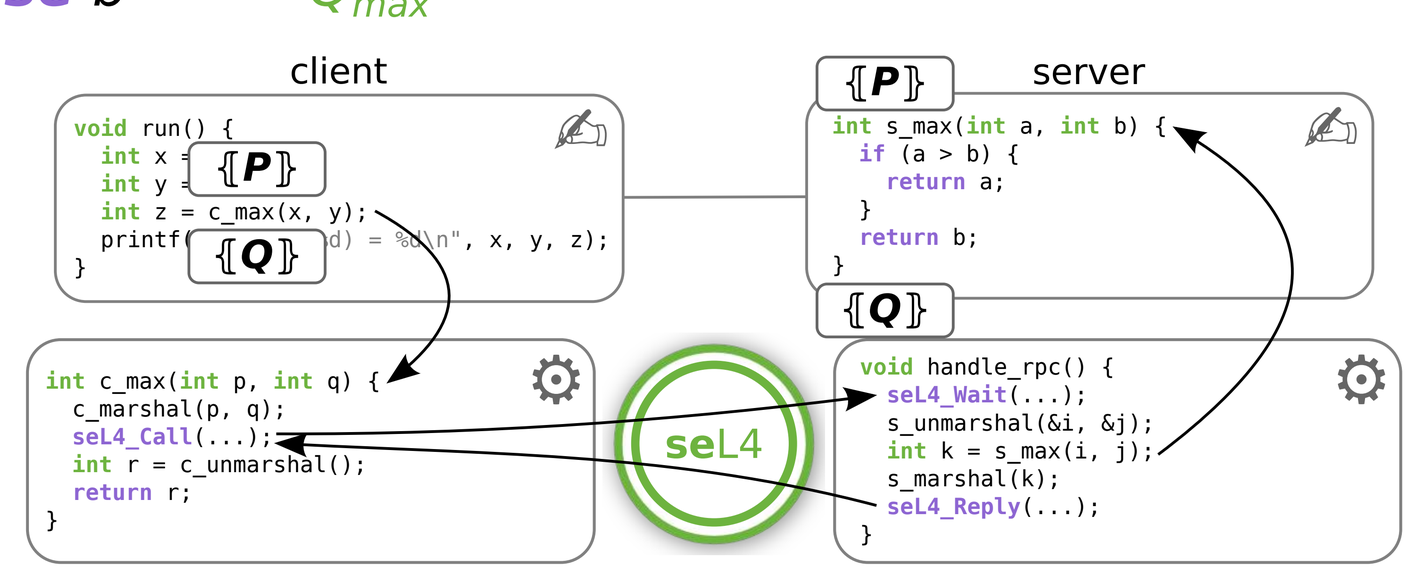
{λ_. True}
s_max a b
*{λr _. r = **if** a > b **then** a **else** b}!*

interpretation $rpcstubs_{max}$

λ_ _ _. True -- P_{max}
*λ_ _ r a b. r = **if** a > b **then** a **else** b* -- Q_{max}

theorem

{λs. inv s}
__do__ c_marshal a b;
s_unmarshal a b;
r ← s_max a b;
s_marshal r;
c_unmarshal r
__od__
*{λr s. inv s ∧ r = **if** a > b **then** a **else** b}!*



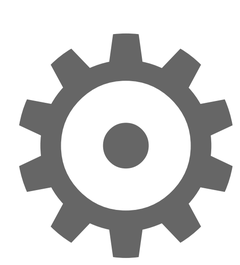
Challenges

NICTA

What is correctness?

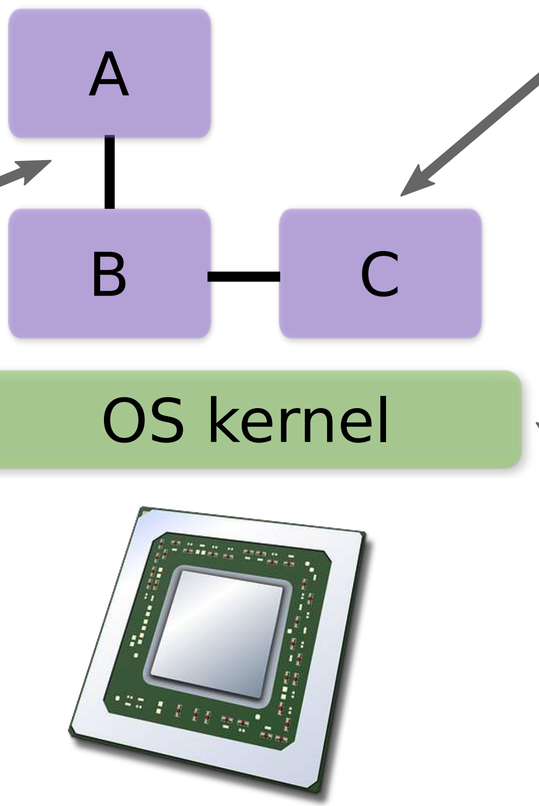
How do we prove it?

How do we automate it?



Generated Proofs for Generated Code

NICTA



components
- functionality
- policy
- >1 MLoC
- varied correctness criteria

CAmkES
- static component platform
- formal architectural model
- optimised for seL4
- designed for verification

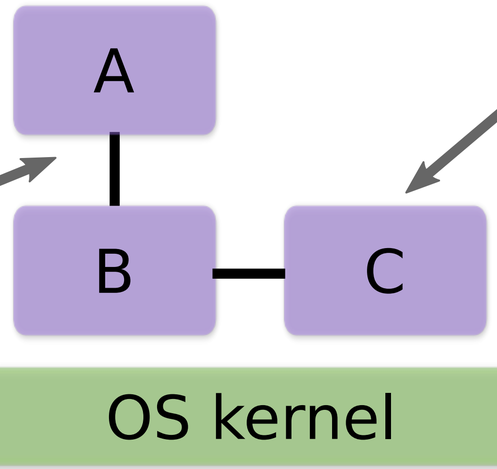
seL4
- 10,000 LoC microkernel
- functional correctness proof
- strong isolation properties
- fast and predictable

Generated Proofs for Generated Code

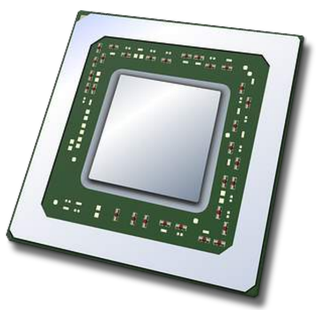
NICTA

proof obligations

components
- functionality
- policy
- >1 MLoC
- varied correctness criteria



CAmkES
- static component platform
- formal architectural model
- optimised for seL4
- designed for verification



seL4
- 10,000 LoC microkernel
- functional correctness proof
- strong isolation properties
- fast and predictable

Generated Proofs for Generated Code

NICTA

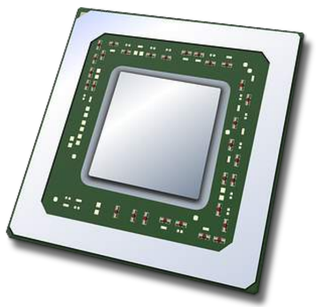
proof obligations

components
- functionality
- policy
- >1 MLoC
- varied correctness criteria

A
B
C

CAmkES
- static component platform
- formal architectural model
- optimised for seL4
- designed for verification

OS kernel



seL4
- 10,000 LoC microkernel
- functional correctness proof
- strong isolation properties
- fast and predictable