

Automated Verification of RPC Stub Code

Matthew Fernandez, June Andronick, Gerwin Klein, Ihor Kuz

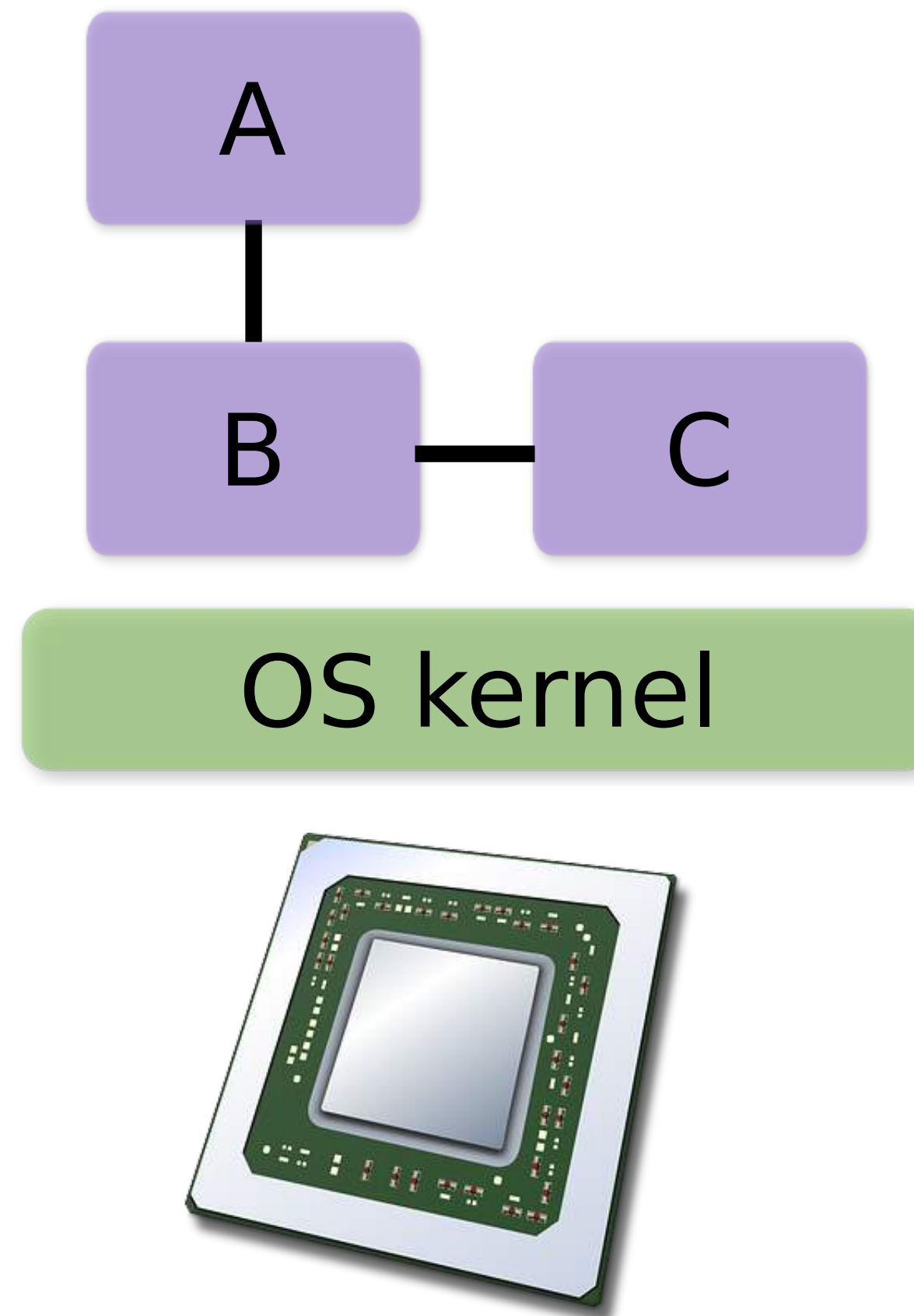
FM, June 2015

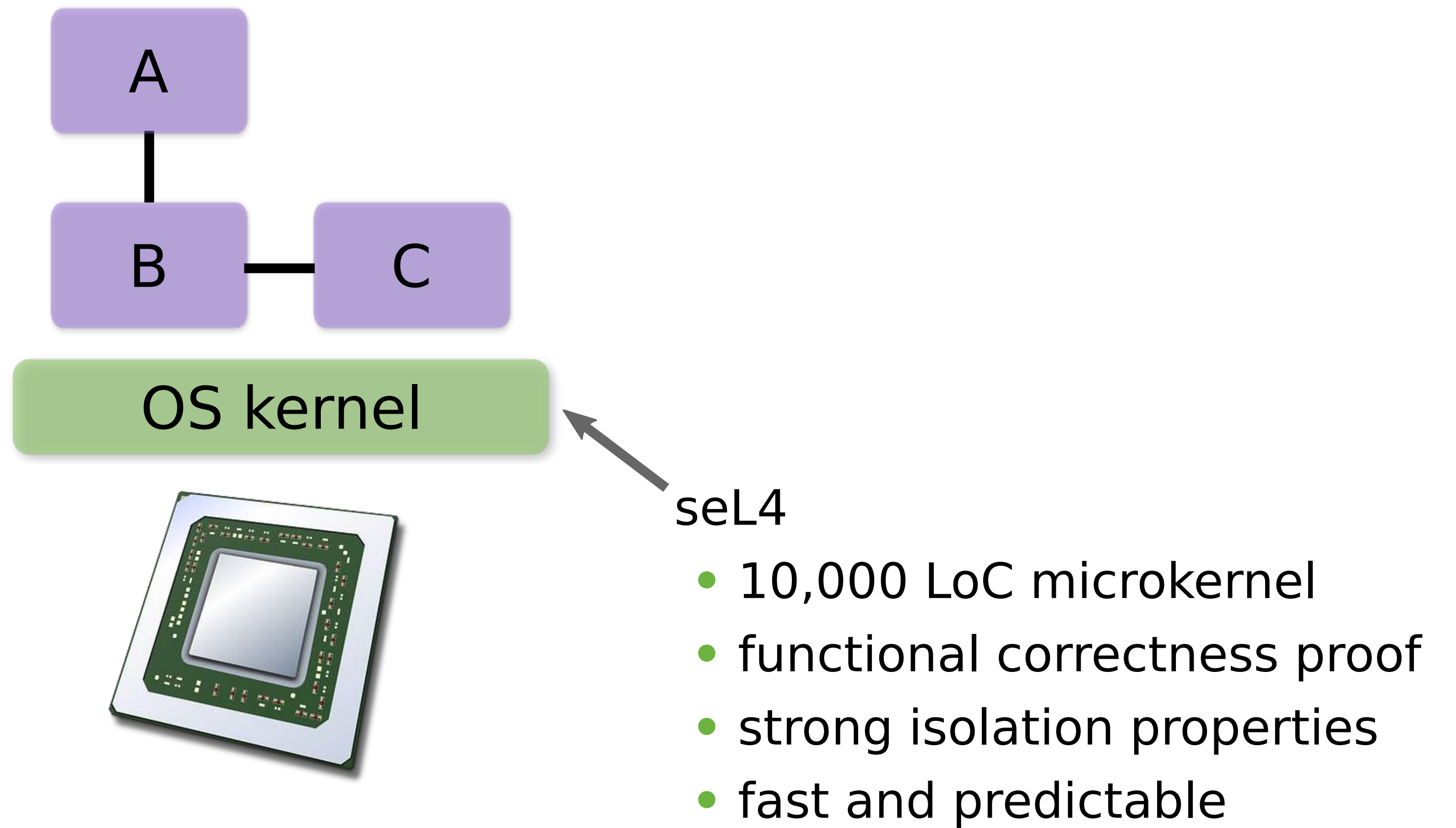


Australian Government

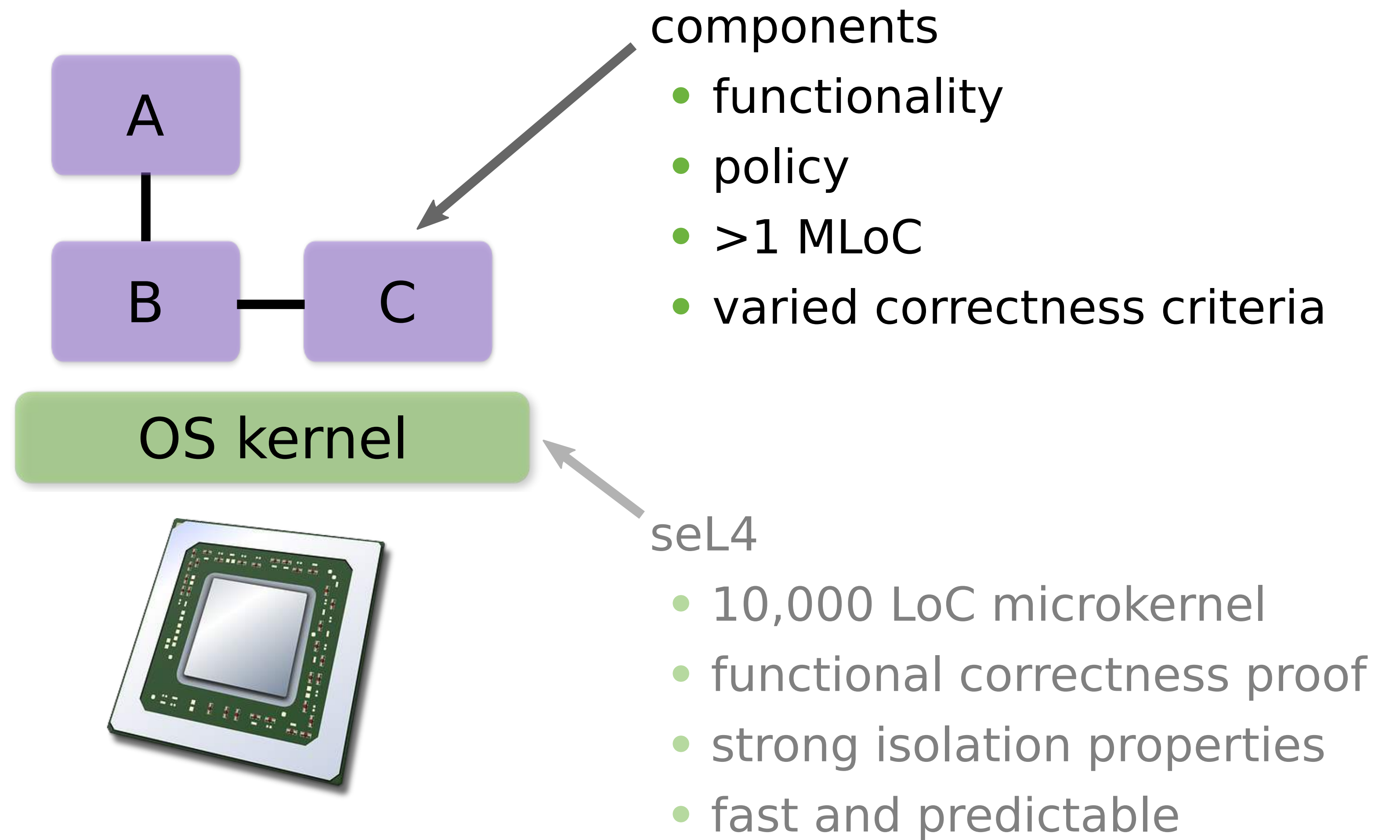


High Assurance Component-based Software

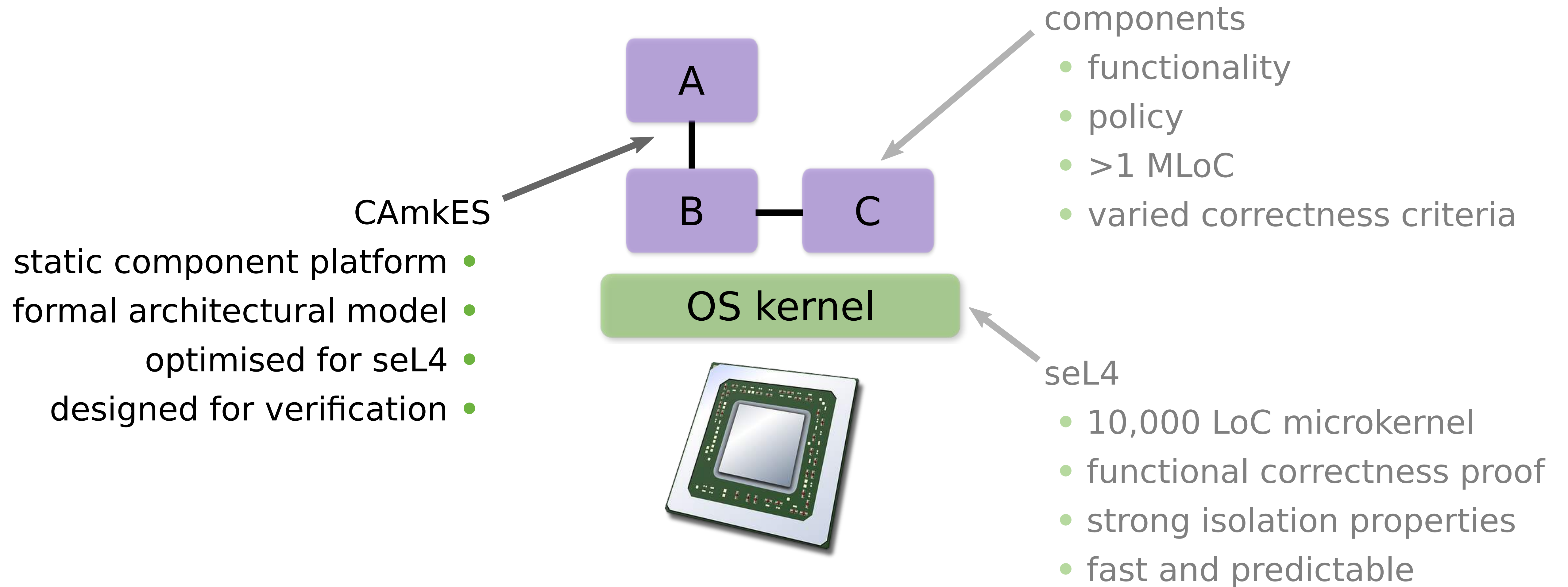




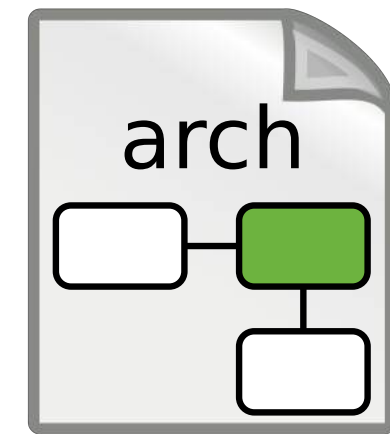
High Assurance Component-based Software



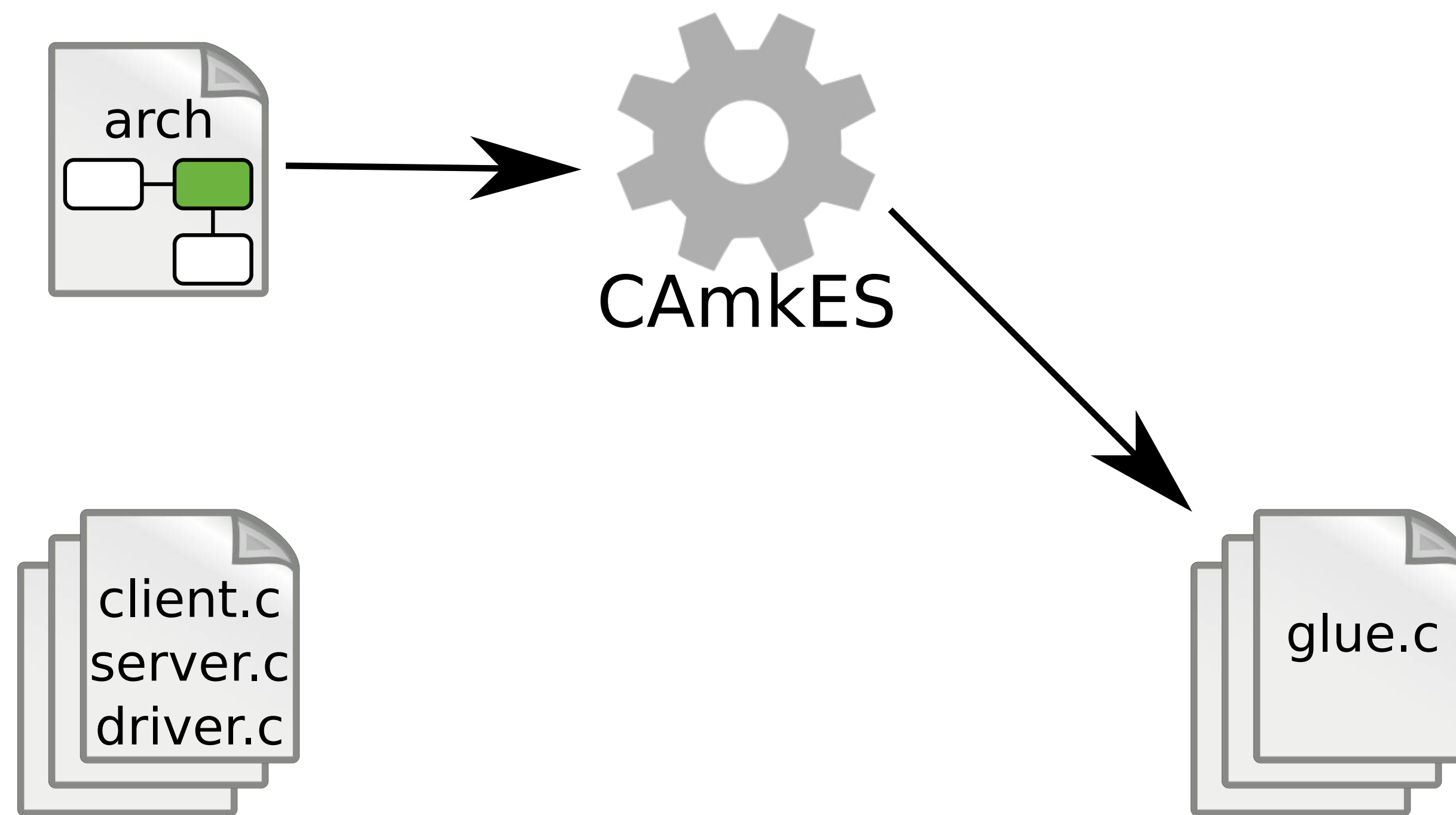
High Assurance Component-based Software



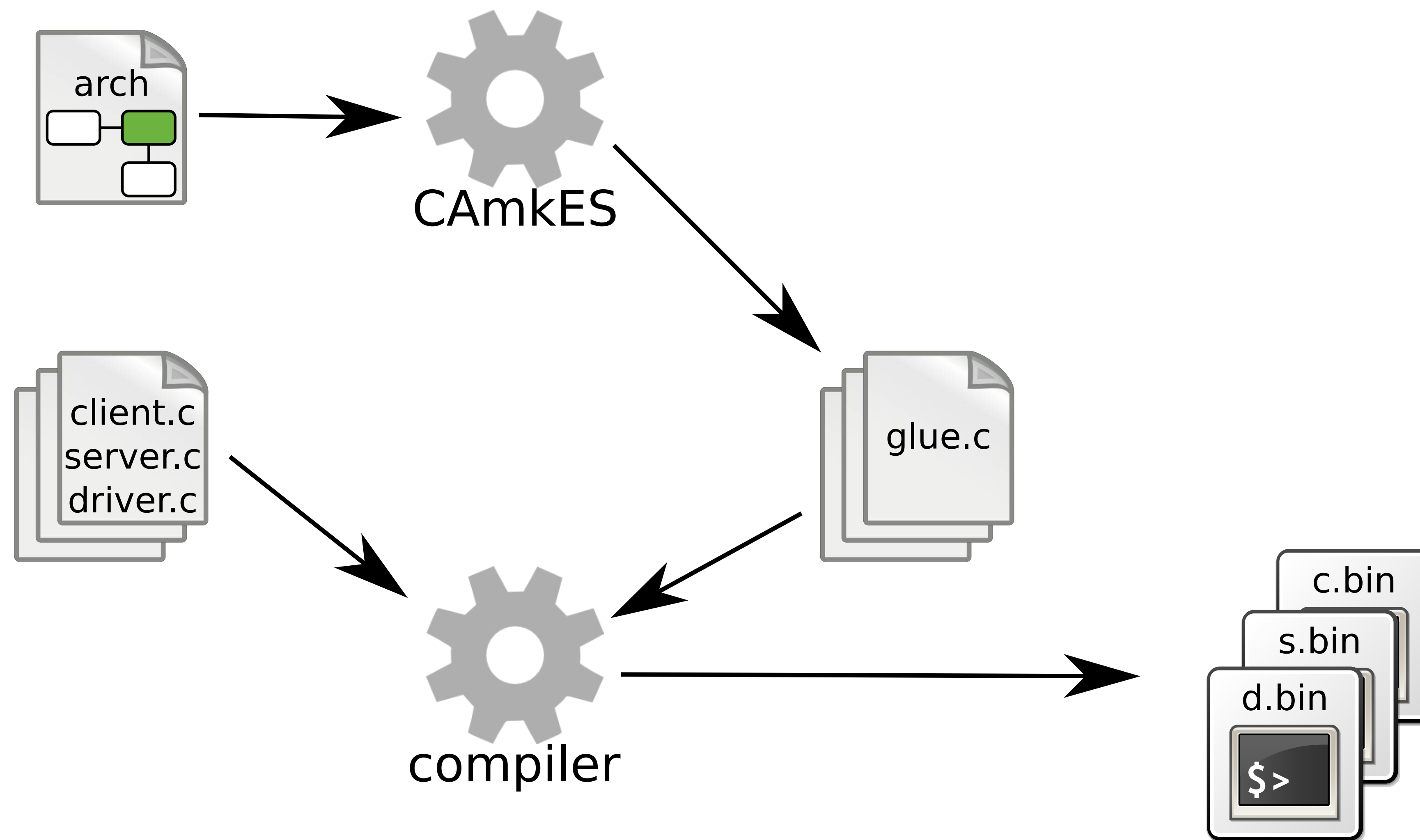
Component Compilation



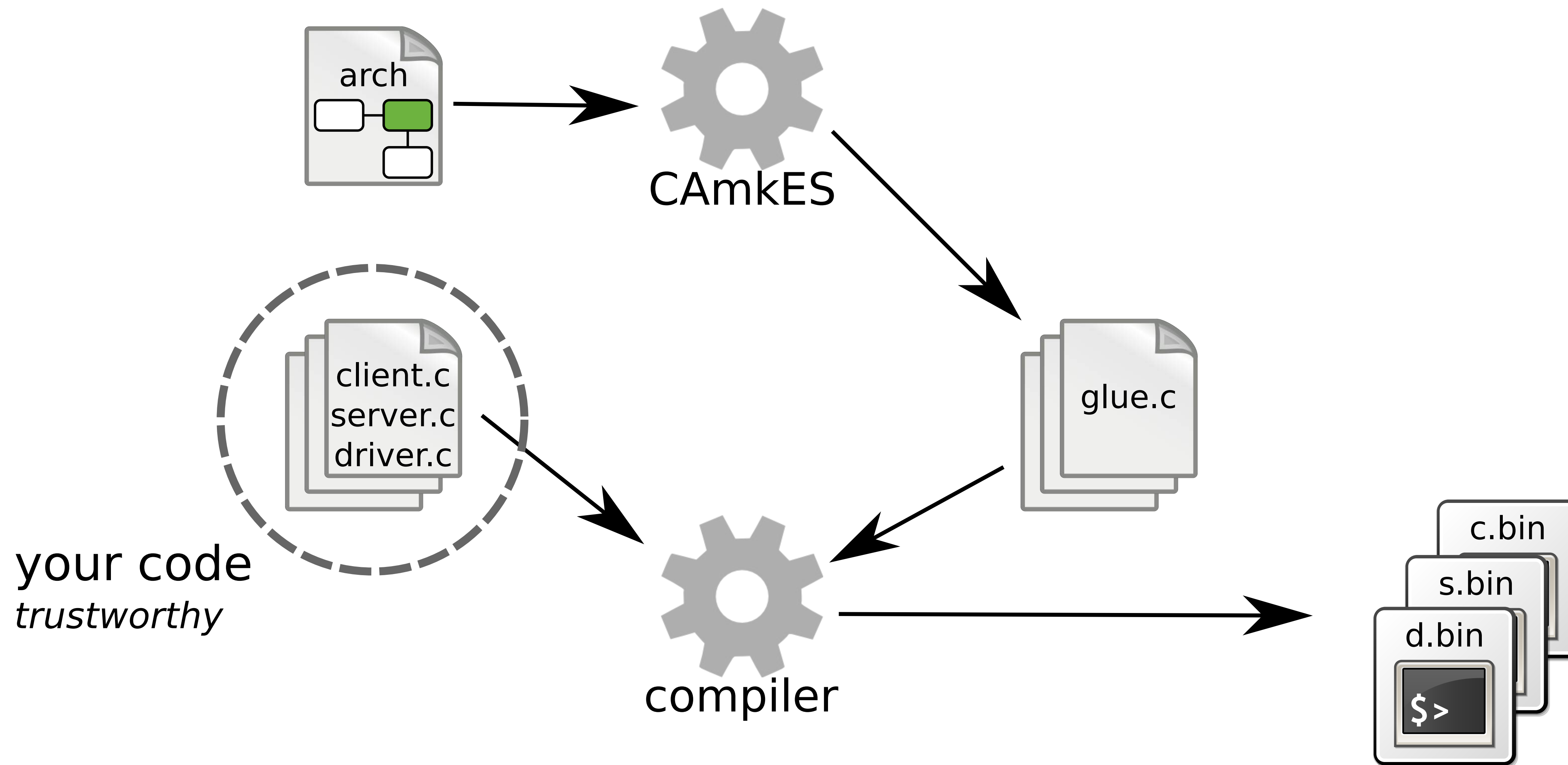
Component Compilation



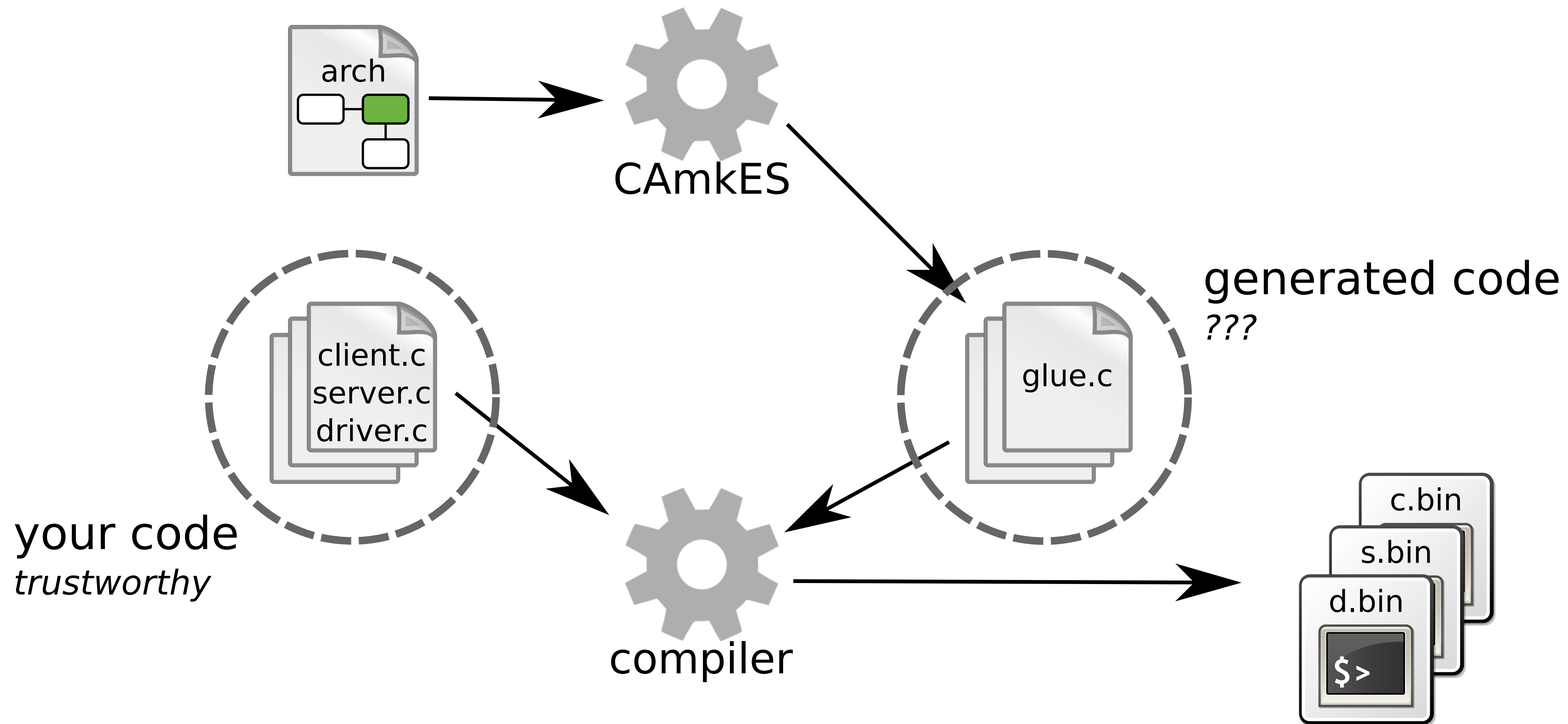
Component Compilation



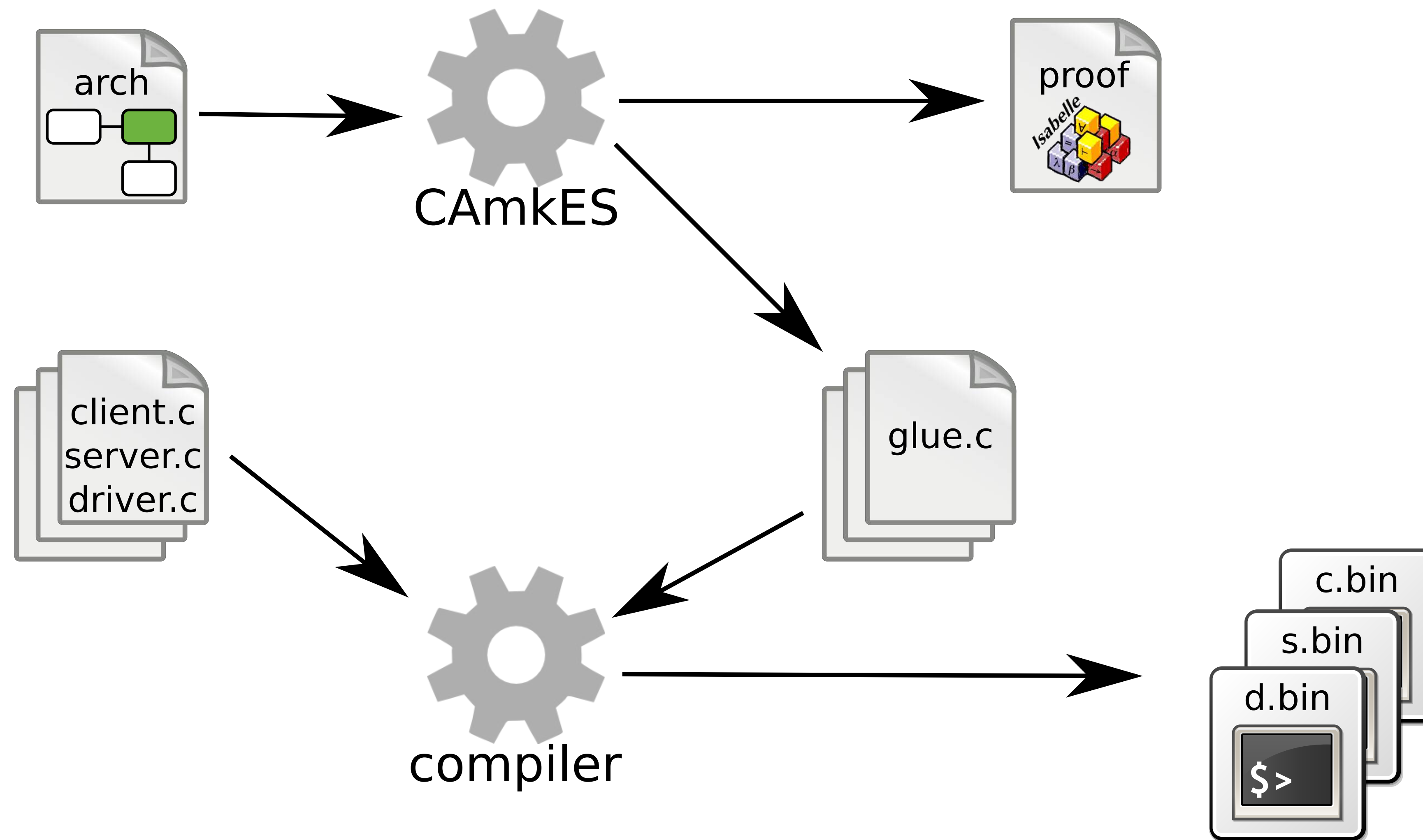
Component Compilation



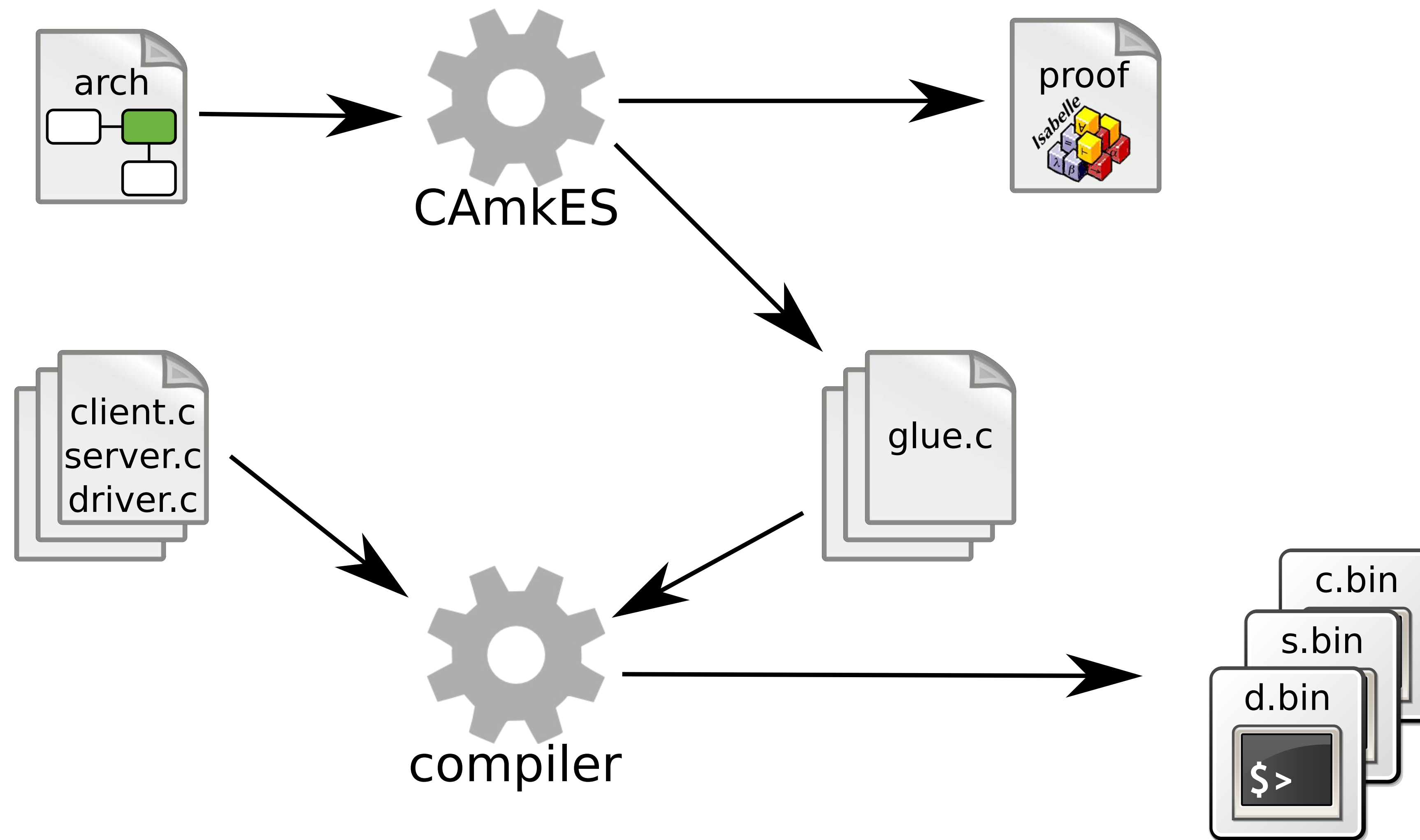
Component Compilation



Component Compilation



Component Compilation

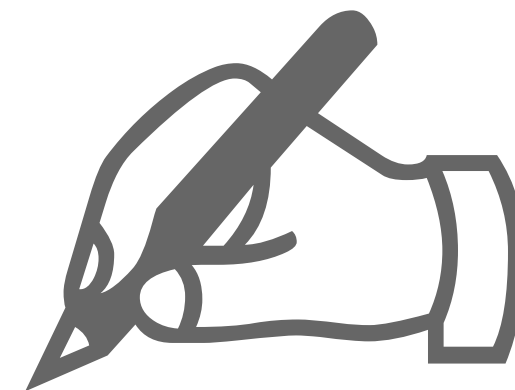


generated proofs for generated code

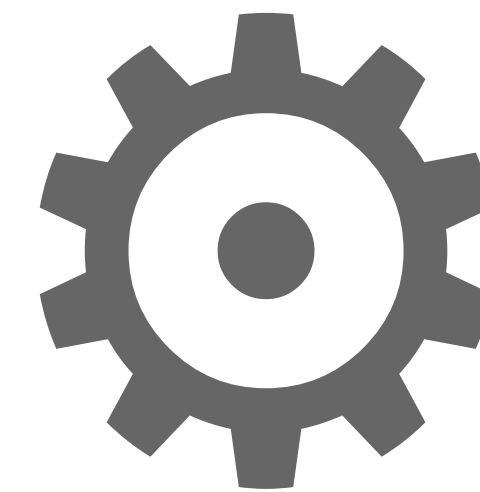
***What is
correctness?***



***How do we
prove it?***



***How do we
automate it?***



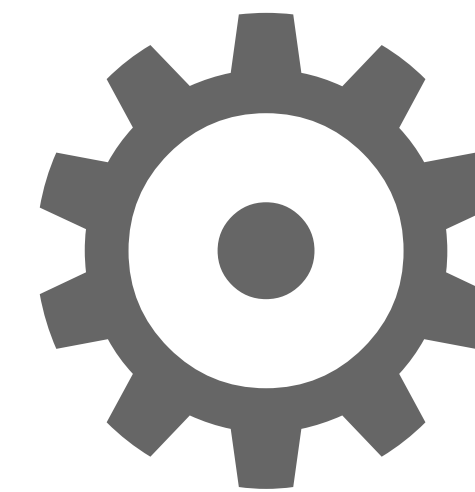
***What is
correctness?***



***How do we
prove it?***



***How do we
automate it?***



Generated Code



client




server




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;  
}
```




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;  
}
```




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

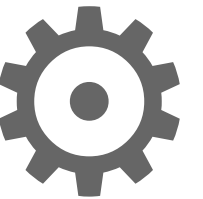
```
int s_max(int a, int b) {  
    if (a > b) {  
        return a;  
    }  
    return b;  
}
```



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




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




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_marshall(p, q);  
  seL4_Call(...);  
  int r = c_unmarshall();  
  return r;  
}
```

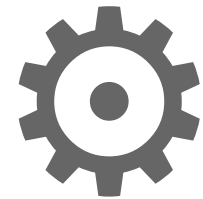


server

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




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




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

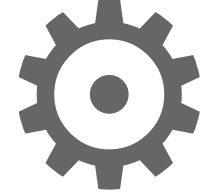
```
int s_max(int a, int b) {  
  if (a > b) {  
    return a;  
  }  
  return b;  
}
```



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



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



client

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

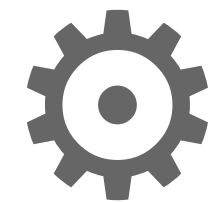


server

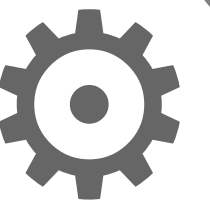
```
int s_max(int a, int b) {  
  if (a > b) {  
    return a;  
  }  
  return b;  
}
```



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



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



client

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

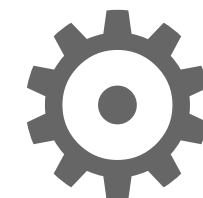


server

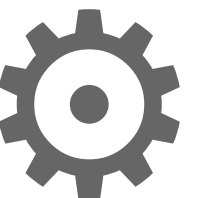
```
int s_max(int a, int b) {  
  if (a > b) {  
    return a;  
  }  
  return b;  
}
```



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



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



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_marshall(p, q);  
  seL4_Call(...);  
  int r = c_unmarshall();  
  return r;  
}
```

{P}

server

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

{Q}

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



client

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

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

server

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

{Q}

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



client

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

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

{P}

server

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

{Q}

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

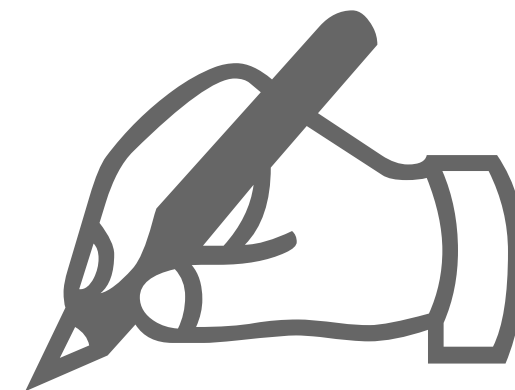


RPC invocation \cong local invocation

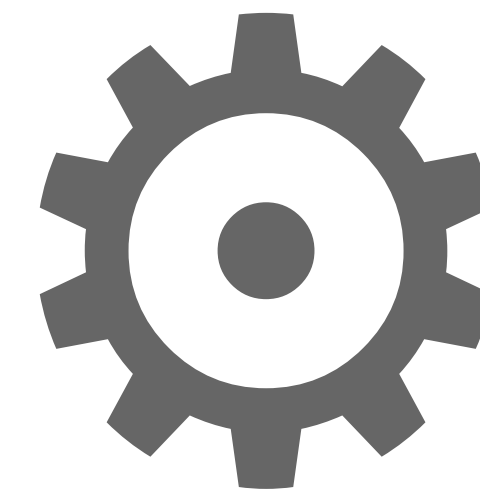
***What is
correctness?***



***How do we
prove it?***



***How do we
automate it?***



Toolchain (prior work)



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

Toolchain (prior work)



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

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

Isabelle/HOL

Toolchain (prior work)



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

C-to-Simpl

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

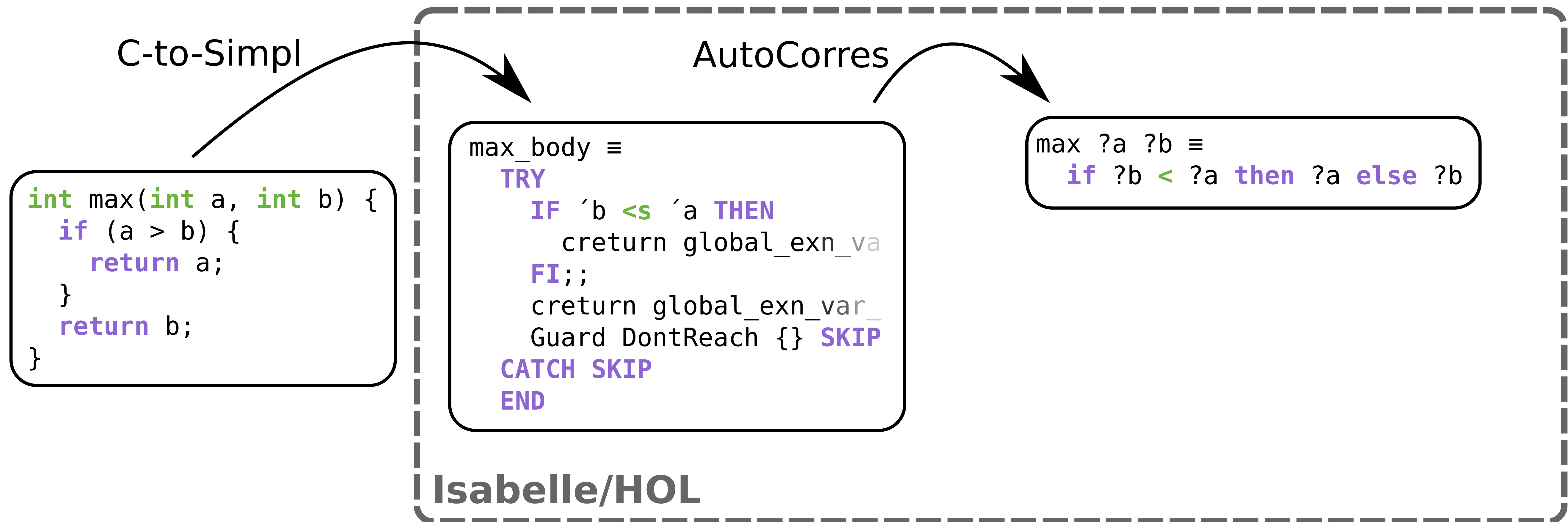
```
max_body ≡  
  TRY  
    IF 'b <s 'a THEN  
      creturn global_exn_va  
    FI;;  
    creturn global_exn_var_  
    Guard DontReach {} SKIP  
  CATCH SKIP  
  END
```

Isabelle/HOL

Toolchain (prior work)



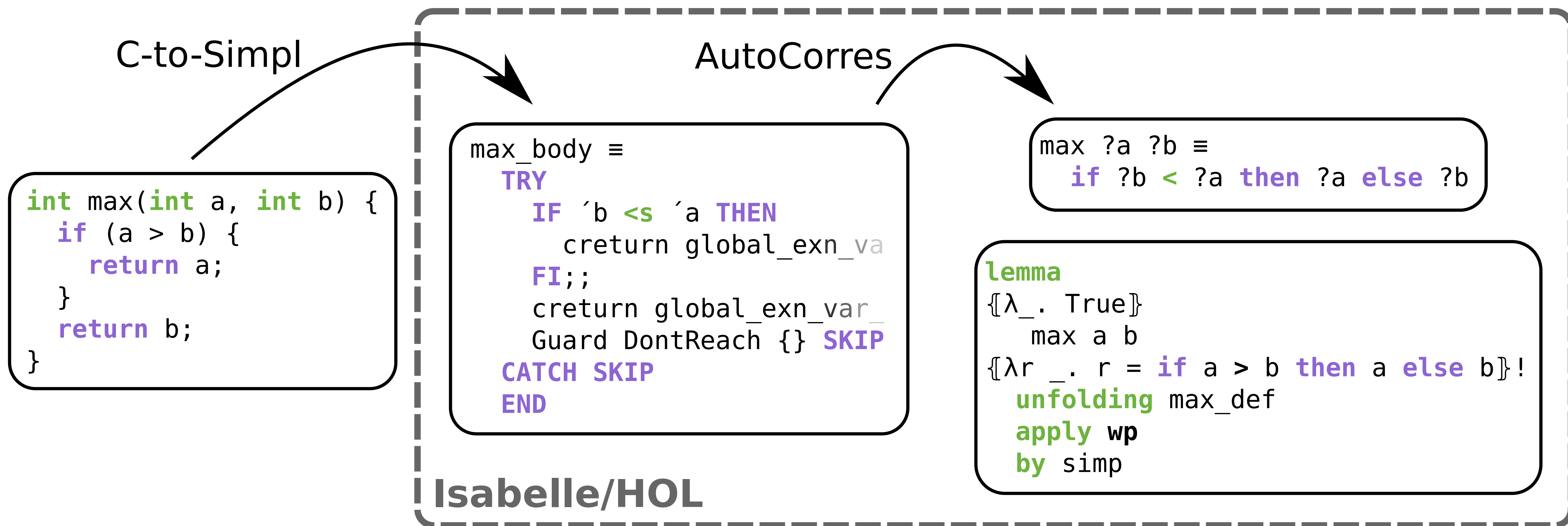
- **Isabelle/HOL** — LCF-style interactive theorem prover
- **C-to-Simpl Parser** — Semantics-preserving translation of C to Simpl
- **AutoCorres** — Verified abstraction of Simpl



Toolchain (prior work)



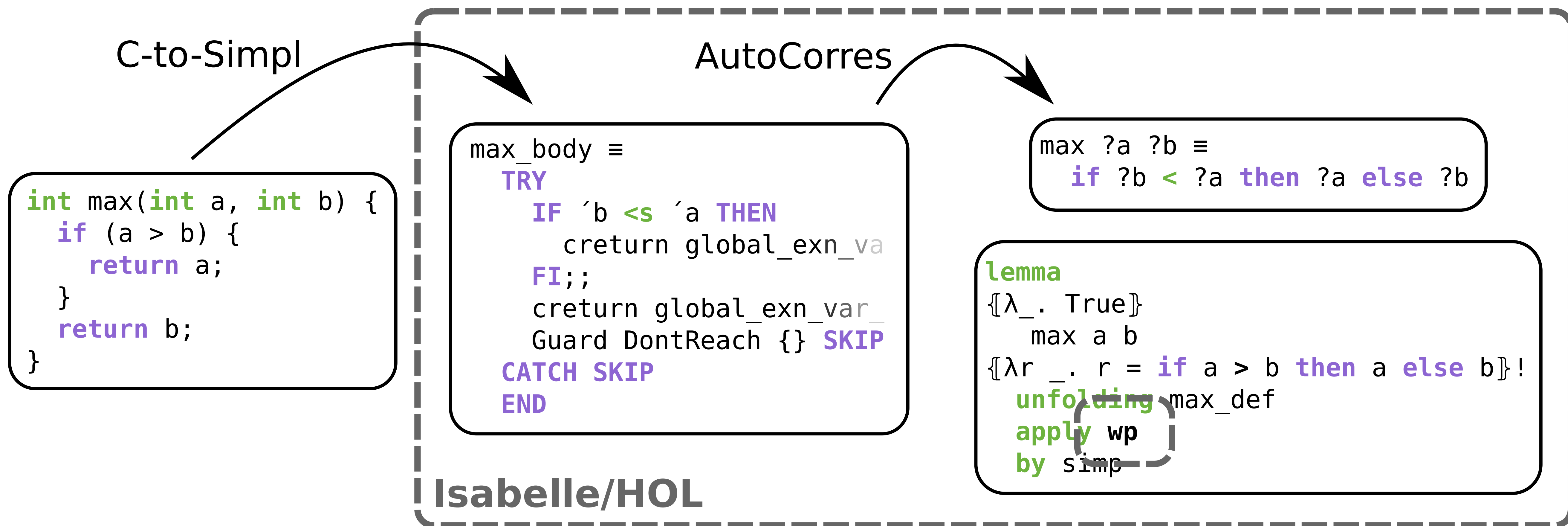
- **Isabelle/HOL** — LCF-style interactive theorem prover
- **C-to-Simpl Parser** — Semantics-preserving translation of C to Simpl
- **AutoCorres** — Verified abstraction of Simpl



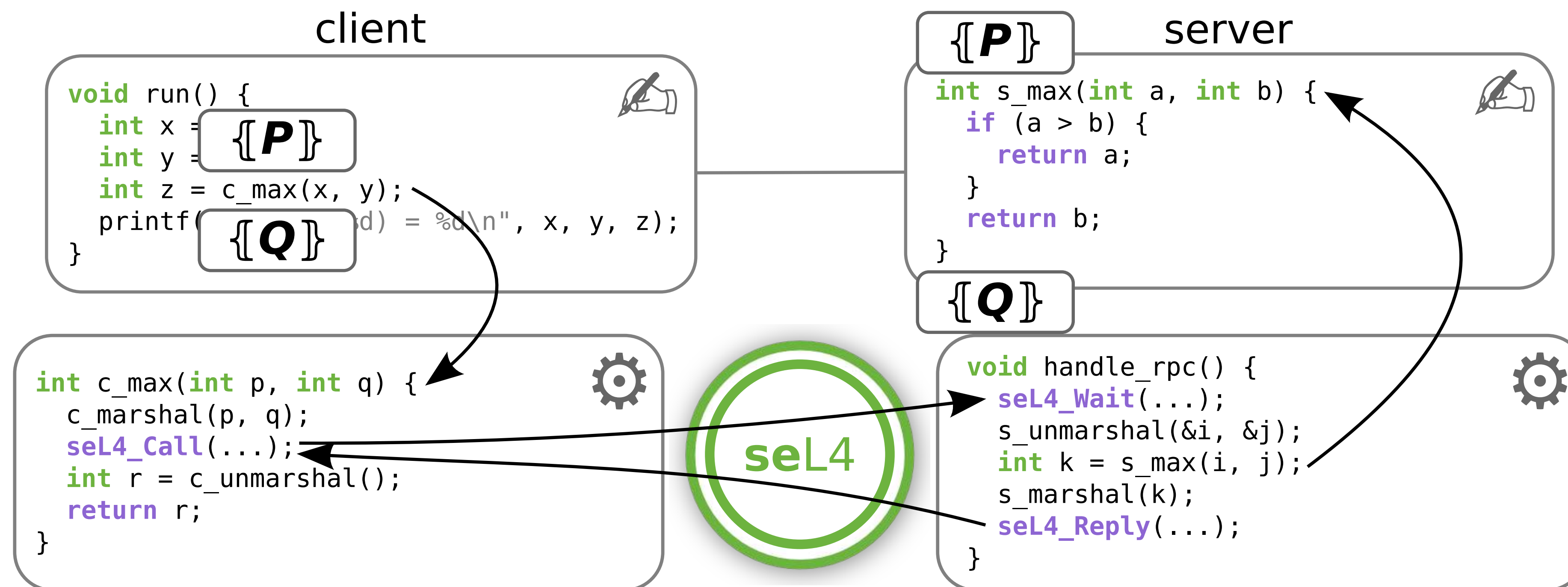
Toolchain (prior work)



- **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



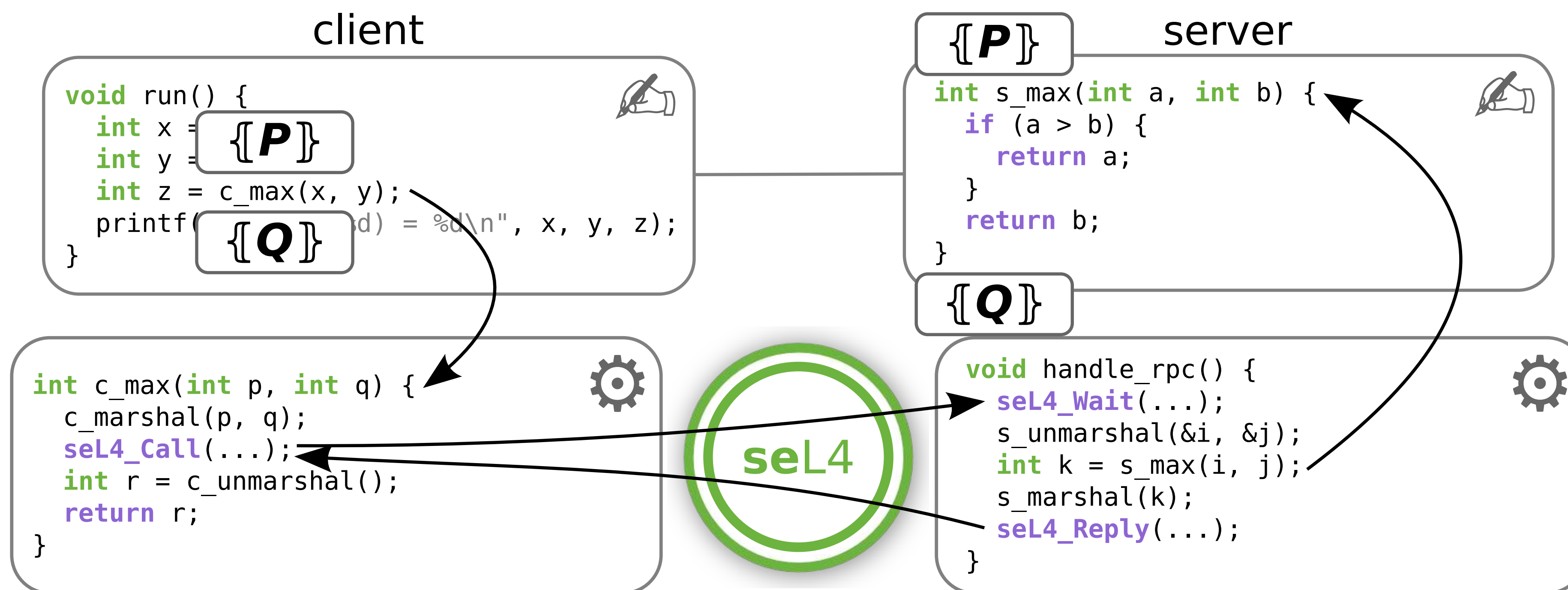
Verifying Generated Code



RPC invocation \cong local invocation

Verifying Generated Code

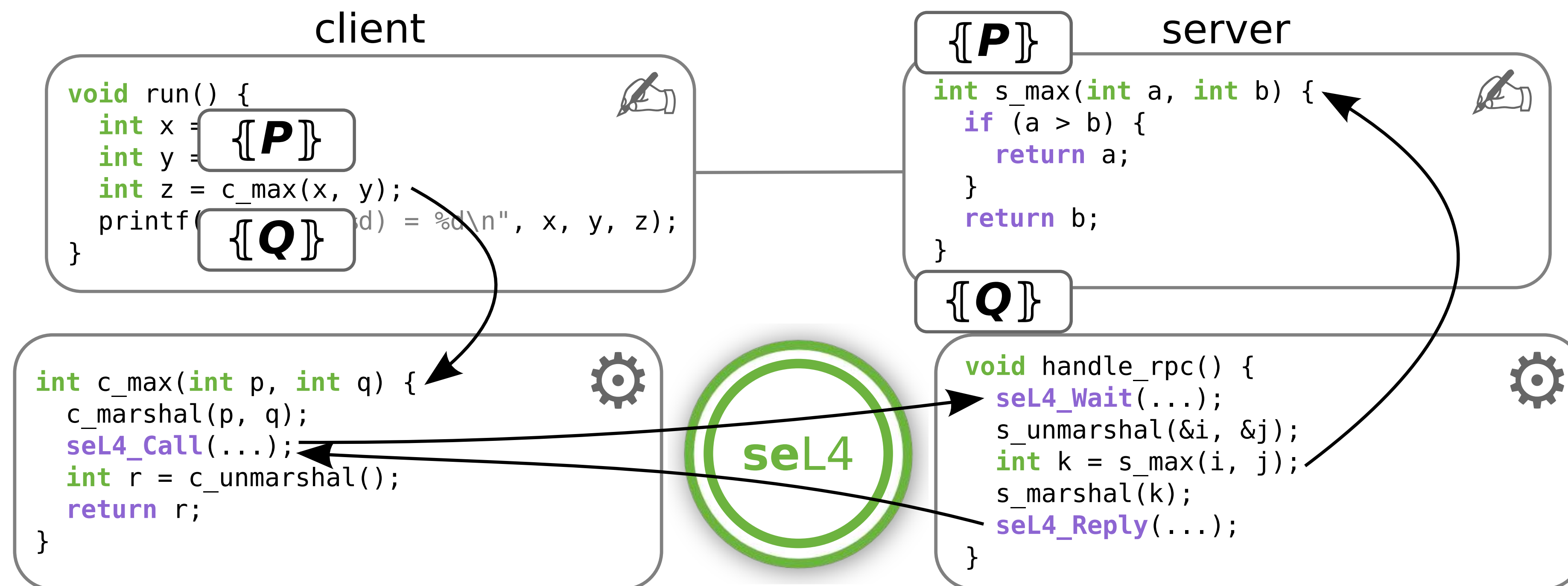
$$\{P\} s_max\ a\ b\ \{Q\}! \Rightarrow \{P\} c_max\ a\ b\ \{Q\}!$$



RPC invocation \cong local invocation

Verifying Generated Code

$\{P\} s_max\ a\ b\ \{Q\}! \Rightarrow \{P\}$
 $\begin{array}{l} \text{do } c_marsh\ a\ b; \\ \text{seL4_Call } \dots; \\ \text{c_unmarshal } r\ \{Q\}! \\ \text{od} \end{array}$

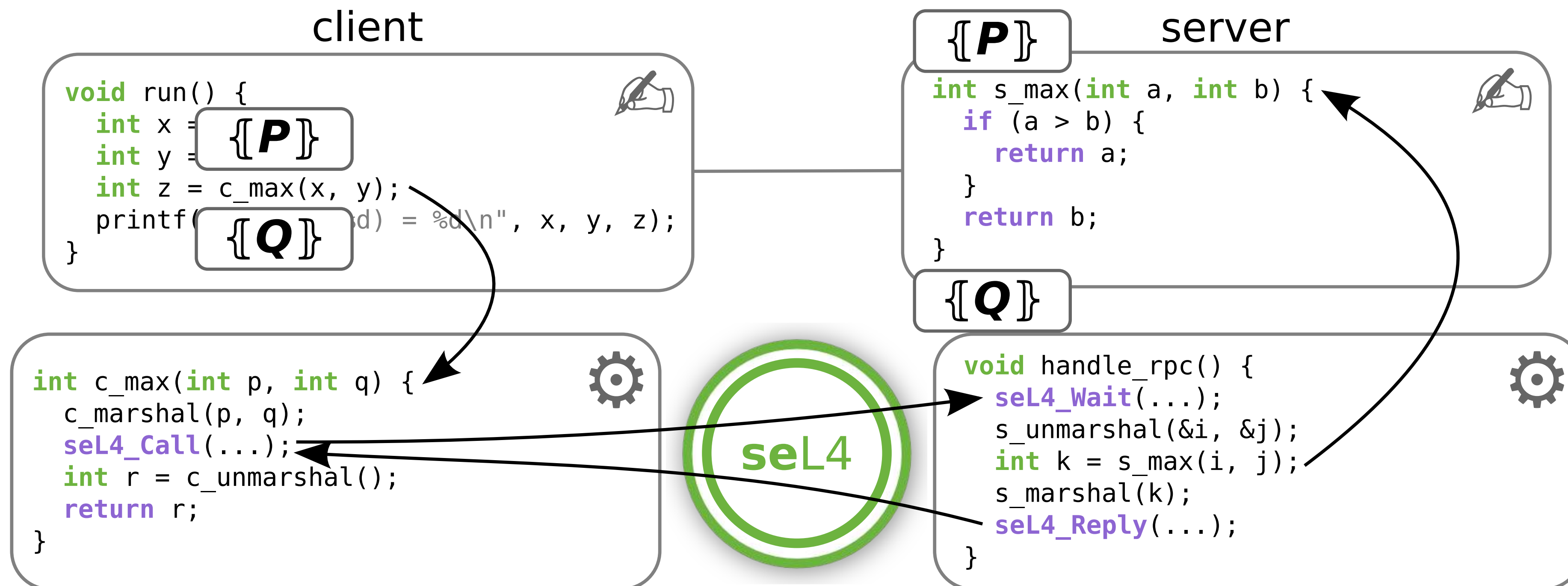


RPC invocation \cong local invocation

Verifying Generated Code

$\{P\} s_max\ a\ b\ \{Q\}! \Rightarrow \{P\}$

```
do c_marshall a b;
   s_unmarshal a b;
   r ← s_max a b;   {Q}!
   s_marshall r;
   c_unmarshal r
od
```

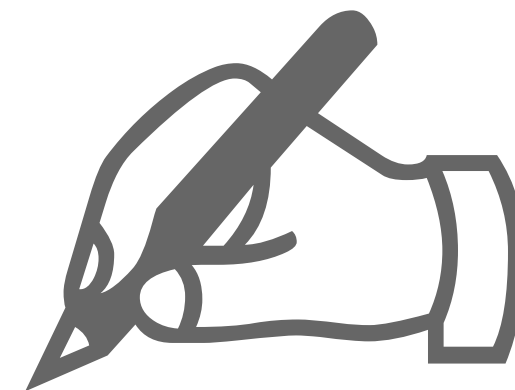


RPC invocation \cong local invocation

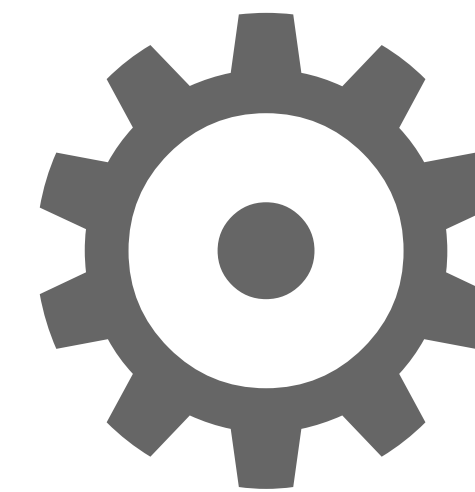
***What is
correctness?***



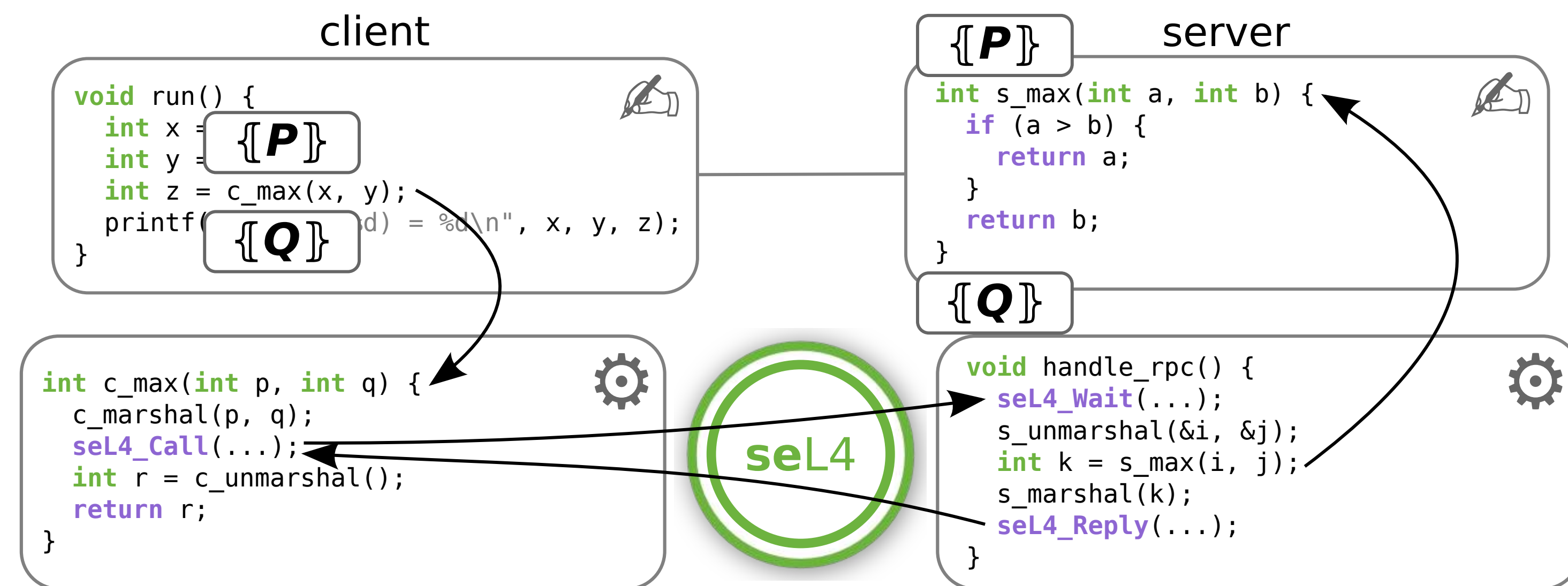
***How do we
prove it?***



***How do we
automate it?***

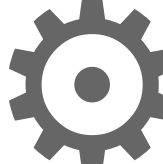


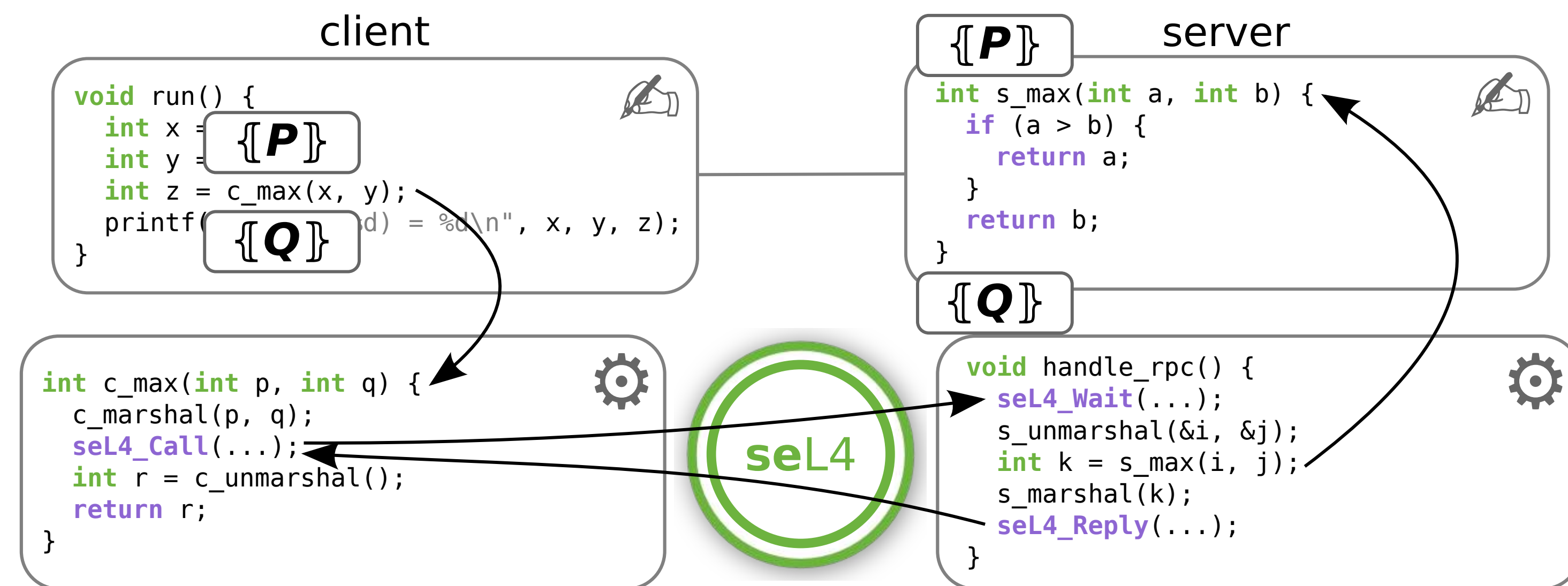
Generated Proofs



Generated Proofs



 **locale** rpcstubs_{max} =



 **locale** rpcstubs_{max} =

fixes $P_{max} :: lifted_globals \Rightarrow int \Rightarrow int \Rightarrow bool$

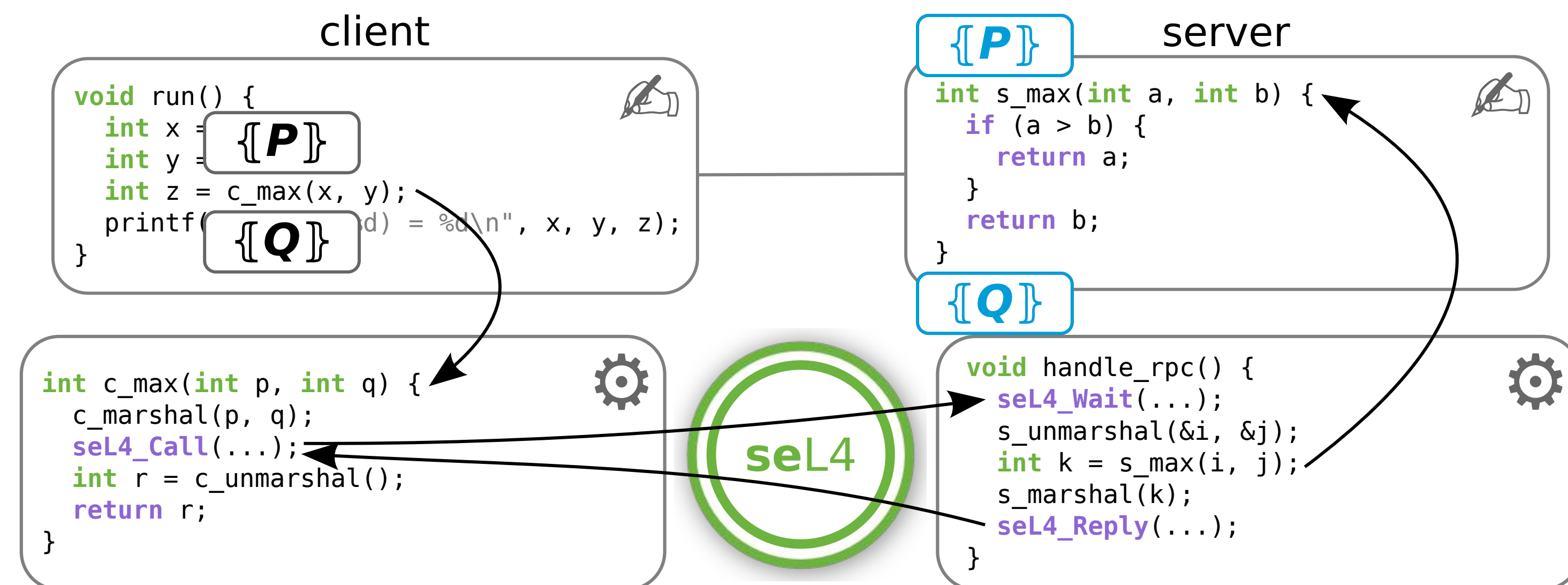
fixes $Q_{max} :: lifted_globals \Rightarrow lifted_globals \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow bool$

assumes

$\{\lambda s. \quad P_{max} \ s \ a \ b\}$

$s_max \ a \ b$

$\{\lambda r \ s. \quad Q_{max} \ s0 \ s \ r \ a \ b\}!$



 **locale** rpcstubs_{max} =

fixes $P_{max} :: lifted_globals \Rightarrow int \Rightarrow int \Rightarrow bool$

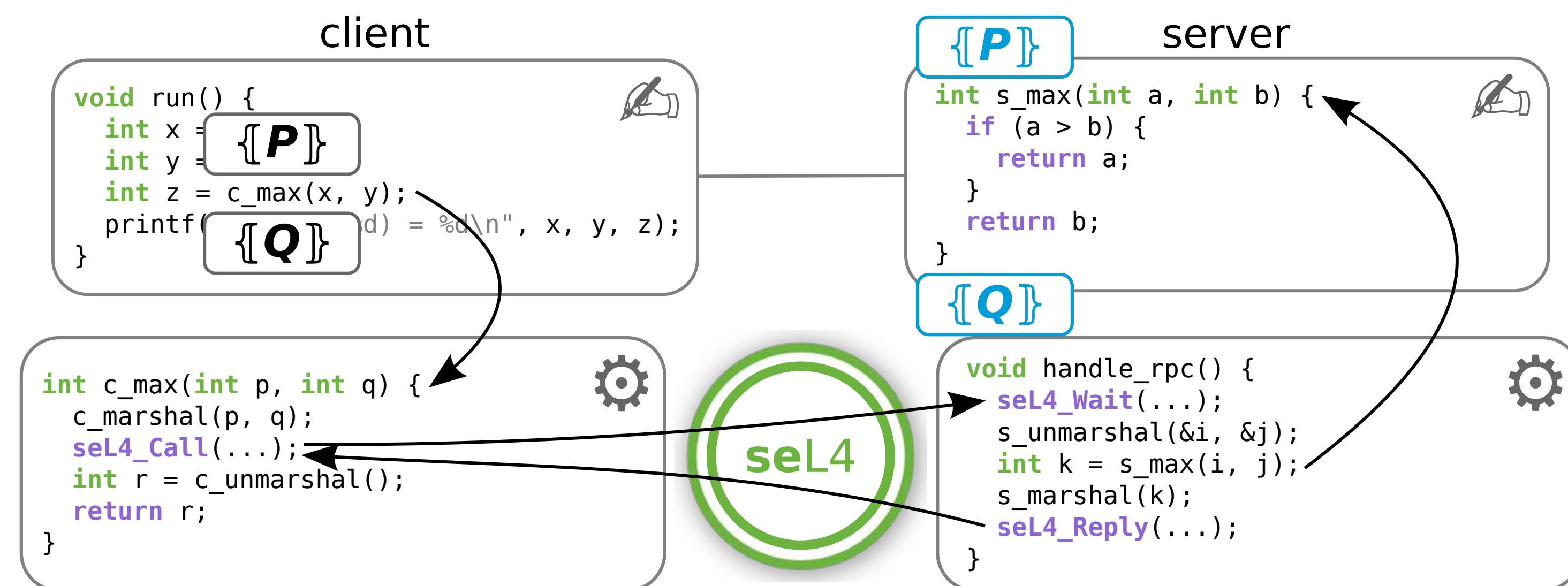
fixes $Q_{max} :: lifted_globals \Rightarrow lifted_globals \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow bool$

assumes

$\{\lambda s. s = s0 \wedge inv\ s \wedge P_{max}\ s\ a\ b\}$

$s_max\ a\ b$

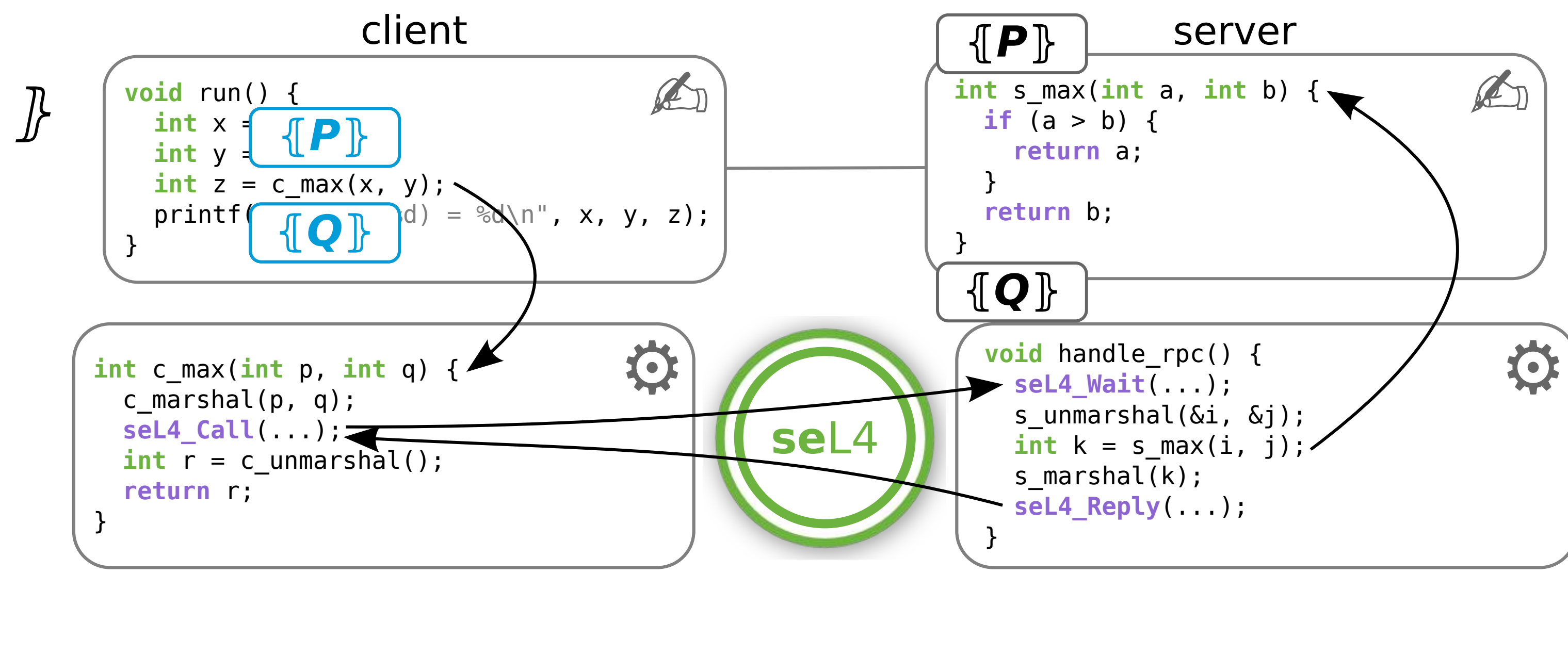
$\{\lambda r\ s. inv\ s \wedge Q_{max}\ s0\ s\ r\ a\ b\}!$



locale `rpcstubsmax` =
fixes $P_{max} :: lifted_globals \Rightarrow int \Rightarrow int \Rightarrow bool$
fixes $Q_{max} :: lifted_globals \Rightarrow lifted_globals \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow bool$
assumes
 $\{\lambda s. s = s0 \wedge inv\ s \wedge P_{max}\ s\ a\ b\}$
 $s_max\ a\ b$
 $\{\lambda r\ s. inv\ s \wedge Q_{max}\ s0\ s\ r\ a\ b\}!$

theorem

{

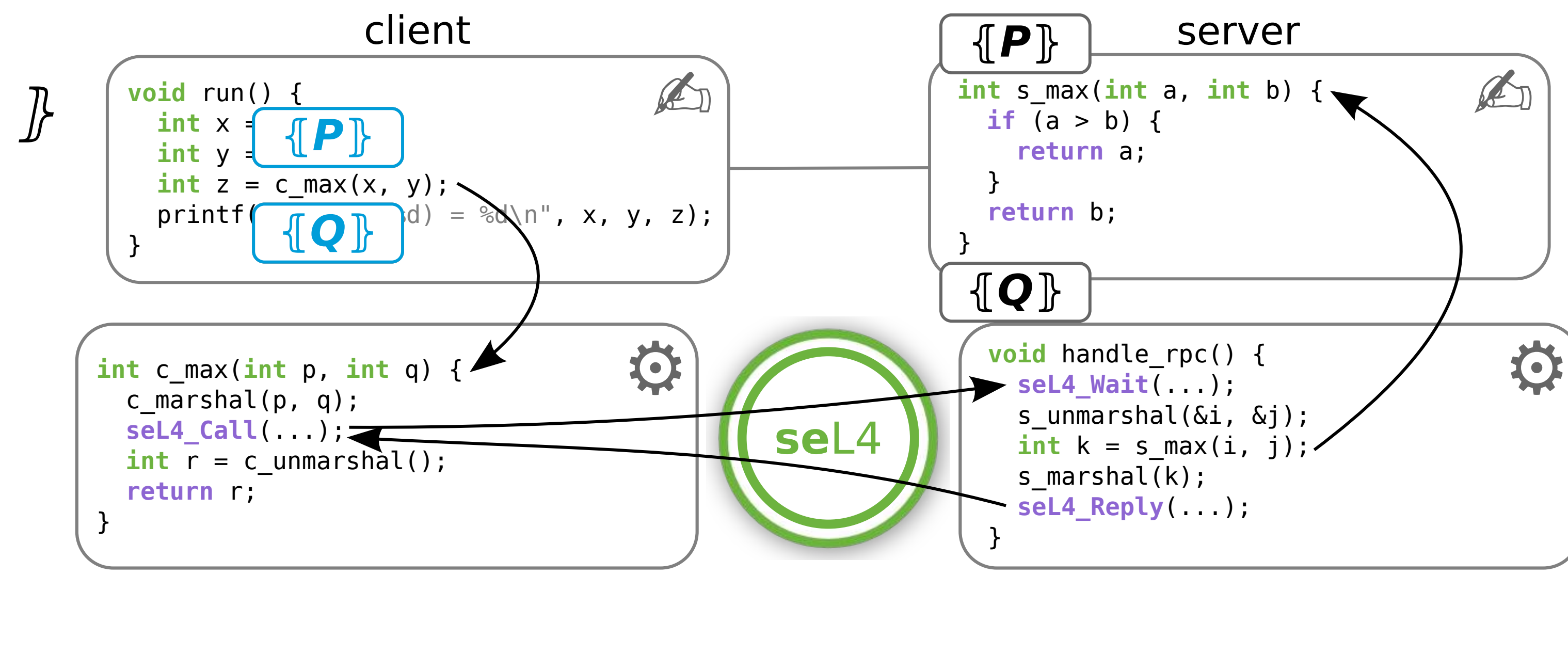


{

}!

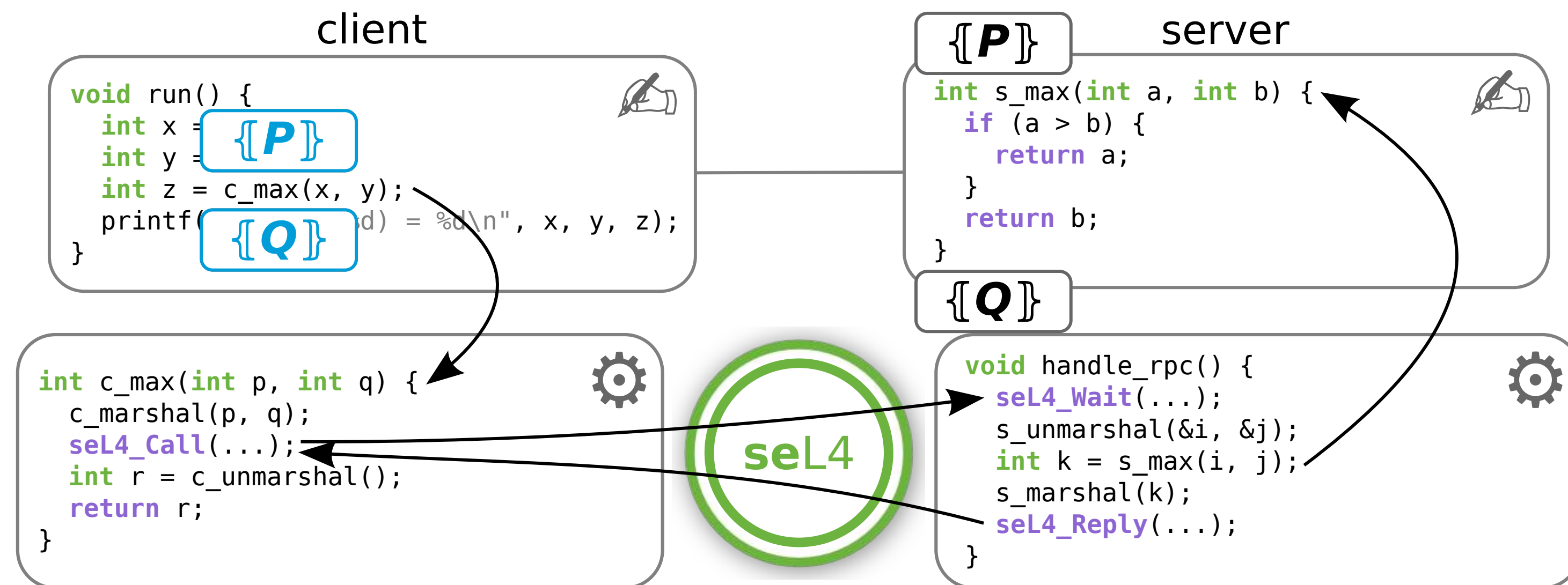
locale `rpcstubsmax` =
fixes $P_{max} :: lifted_globals \Rightarrow int \Rightarrow int \Rightarrow bool$
fixes $Q_{max} :: lifted_globals \Rightarrow lifted_globals \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow bool$
assumes
 $\{\lambda s. s = s0 \wedge inv\ s \wedge P_{max}\ s\ a\ b\}$
 $s_max\ a\ b$
 $\{\lambda r\ s. inv\ s \wedge Q_{max}\ s0\ s\ r\ a\ b\}!$

theorem
 $\{$
do $c_marshal\ a\ b;$
 $s_unmarshal\ a\ b;$
 $r \leftarrow s_max\ a\ b;$
 $s_marshal\ r;$
 $c_unmarshal\ r$
od
 $\{$



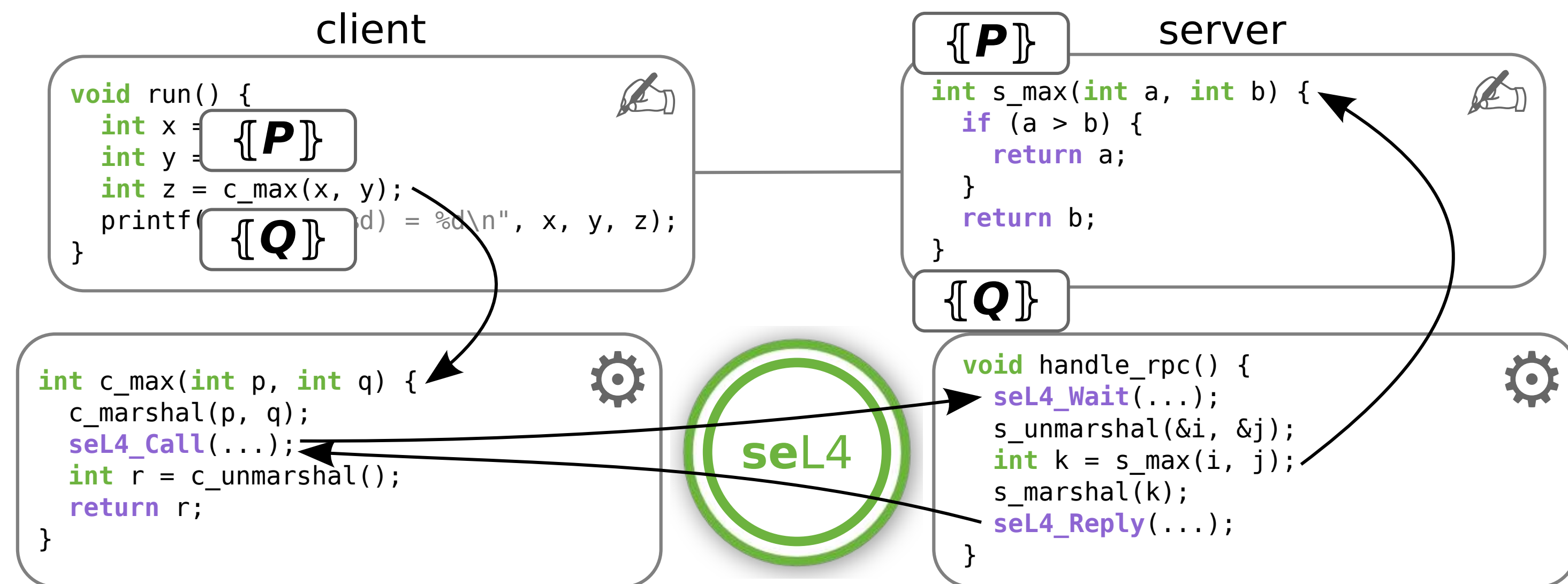
locale `rpcstubsmax` =
fixes $P_{max} :: \text{lifted_globals} \Rightarrow \text{int} \Rightarrow \text{int} \Rightarrow \text{bool}$
fixes $Q_{max} :: \text{lifted_globals} \Rightarrow \text{lifted_globals} \Rightarrow \text{int} \Rightarrow \text{int} \Rightarrow \text{int} \Rightarrow \text{bool}$
assumes
 $\{\lambda s. s = s0 \wedge \text{inv } s \wedge P_{max} \ s \ a \ b\}$
 $s_max \ a \ b$
 $\{\lambda r \ s. \text{inv } s \wedge Q_{max} \ s0 \ s \ r \ a \ b\}!$

theorem
 $\{\lambda s. s = s0 \wedge \text{inv } s \wedge P_{max} \ s \ a \ b\}$
do $c_marshal \ a \ b;$
 $s_unmarshal \ a \ b;$
 $r \leftarrow s_max \ a \ b;$
 $s_marshal \ r;$
 $c_unmarshal \ r$
od
 $\{\lambda r \ s. \text{inv } s \wedge Q_{max} \ s0 \ s \ r \ a \ b\}!$

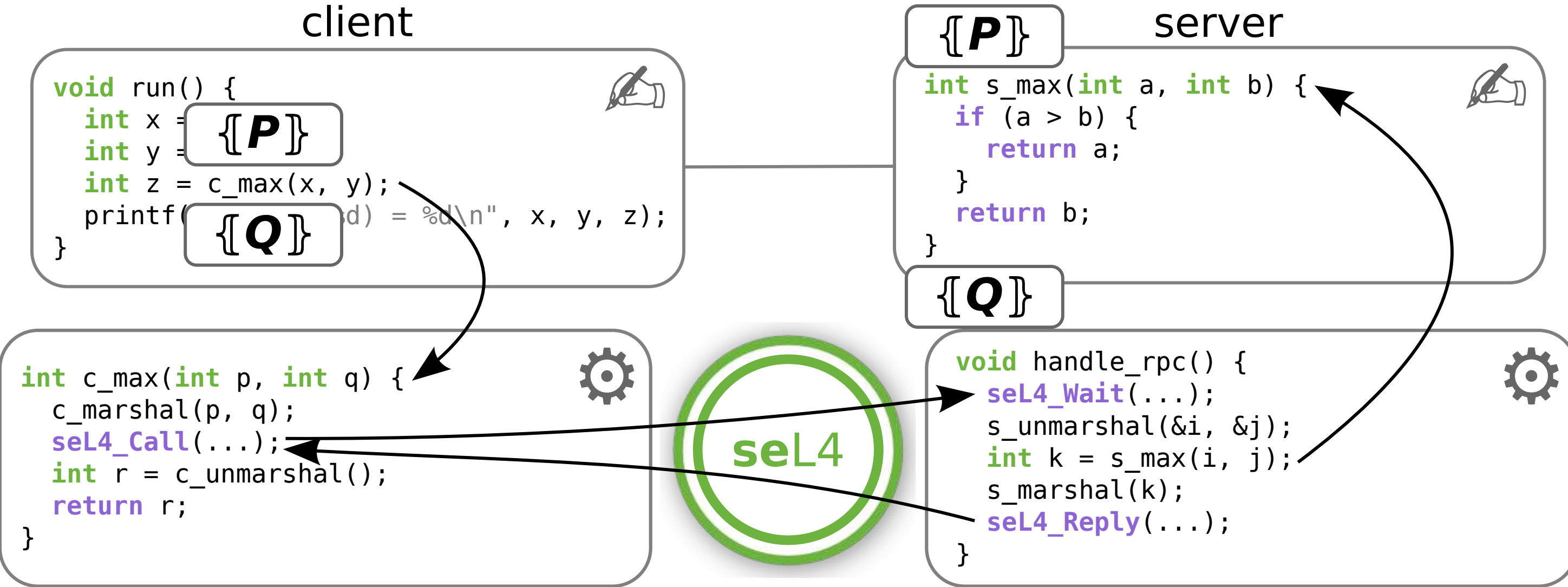


locale `rpcstubsmax` =
fixes $P_{max} :: lifted_globals \Rightarrow int \Rightarrow int \Rightarrow bool$
fixes $Q_{max} :: lifted_globals \Rightarrow lifted_globals \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow bool$
assumes
 $\{\lambda s. s = s0 \wedge inv\ s \wedge P_{max}\ s\ a\ b\}$
 $s_max\ a\ b$
 $\{\lambda r\ s. inv\ s \wedge Q_{max}\ s0\ s\ r\ a\ b\}!$

theorem
 $\{\lambda s. s = s0 \wedge inv\ s \wedge P_{max}\ s\ a\ b\}$
do $c_marshal\ a\ b;$
 $s_unmarshal\ a\ b;$
 $r \leftarrow s_max\ a\ b;$
 $s_marshal\ r;$
 $c_unmarshal\ r$
od
 $\{\lambda r\ s. inv\ s \wedge Q_{max}\ s0\ s\ r\ a\ b\}!$
generated proof



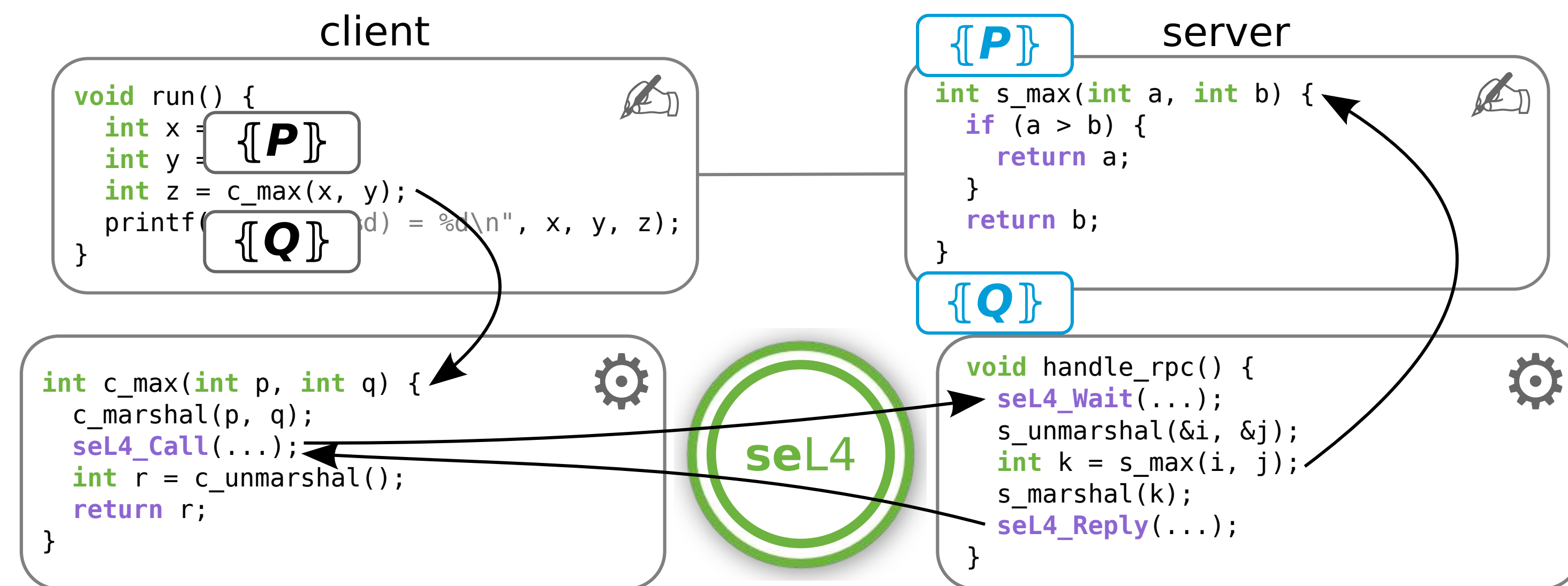
Composing Proofs



Composing Proofs

theorem

$\{\lambda_. True\}$
 $s_max\ a\ b$
 $\{\lambda r_. r = \text{if } a > b \text{ then } a \text{ else } b\}!$

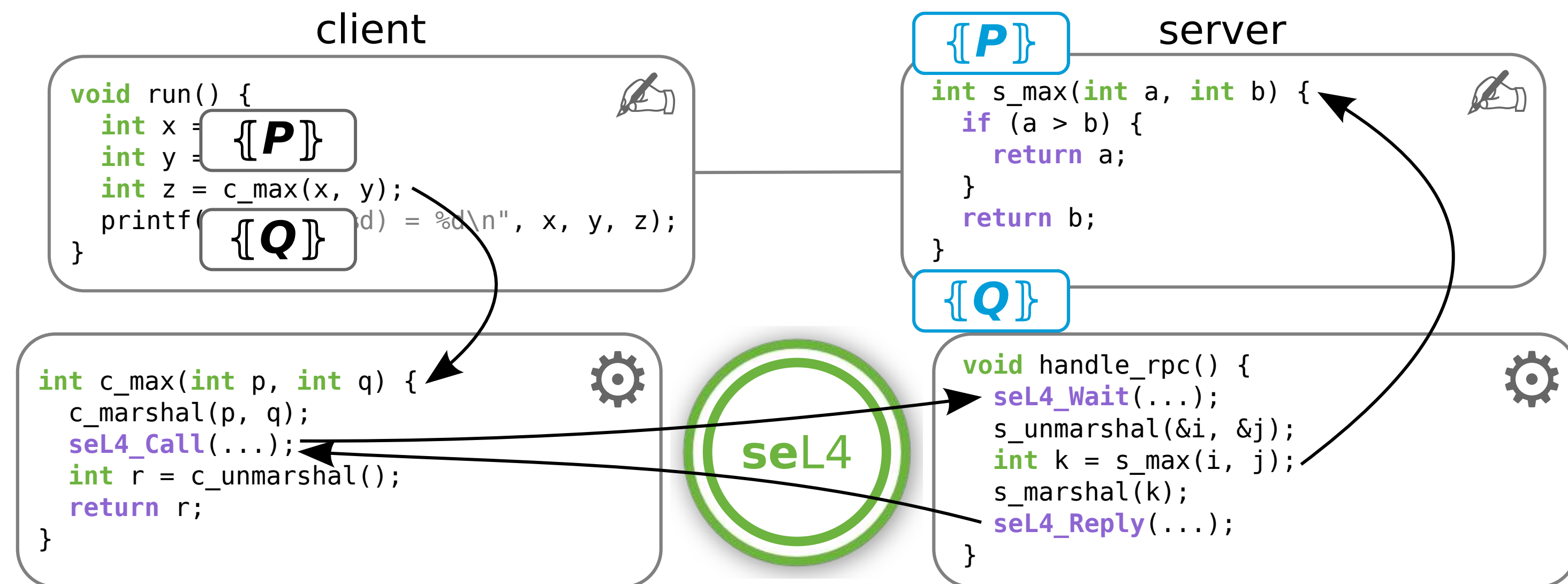


theorem

$\{\lambda_. True\}$
 $s_max\ a\ b$
 $\{\lambda r_. r = \text{if } a > b \text{ then } a \text{ else } b\}!$

interpretation $rpcstubs_{max}$

$\lambda_._._ . True$ -- P_{max}
 $\lambda_._ r\ a\ b. r = \text{if } a > b \text{ then } a \text{ else } b$ -- Q_{max}



 **theorem**

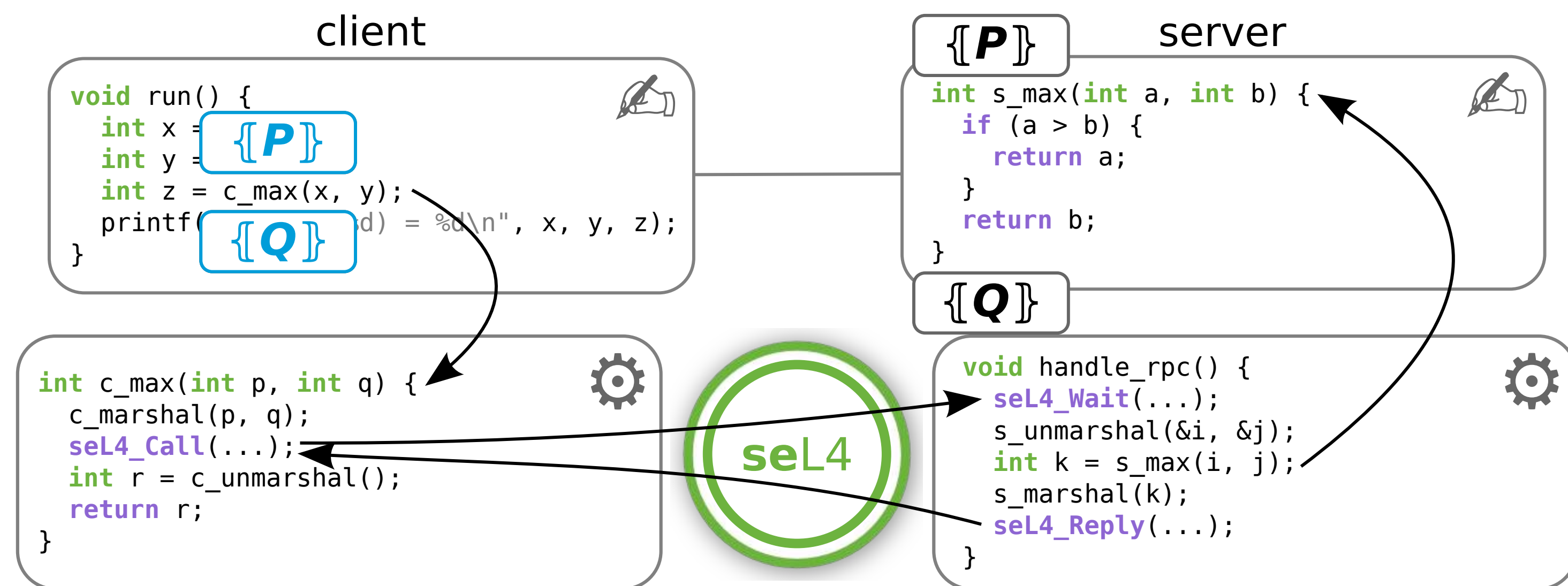
$\{\lambda_. True\}$
 $s_max\ a\ b$
 $\{\lambda r_. r = \text{if } a > b \text{ then } a \text{ else } b\}!$

 **interpretation** $rpcstubs_{max}$

$\lambda_.. True$ -- P_{max}
 $\lambda_r\ a\ b. r = \text{if } a > b \text{ then } a \text{ else } b$ -- Q_{max}

 **theorem**

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



 **theorem**

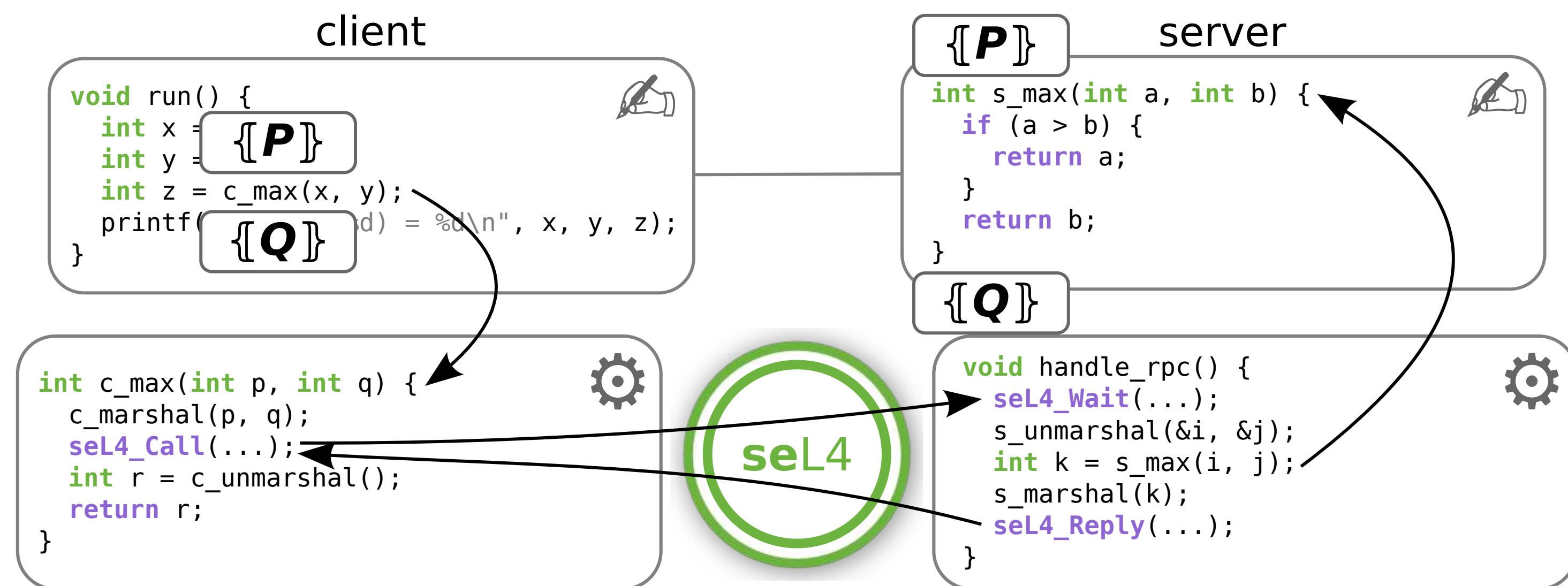
$\{\lambda_. True\}$
 $s_max\ a\ b$
 $\{\lambda r_. r = \text{if } a > b \text{ then } a \text{ else } b\}!$

 **interpretation** $rpcstubs_{max}$

$\lambda_.. True$ -- P_{max}
 $\lambda_r\ a\ b. r = \text{if } a > b \text{ then } a \text{ else } b$ -- Q_{max}

 **theorem**

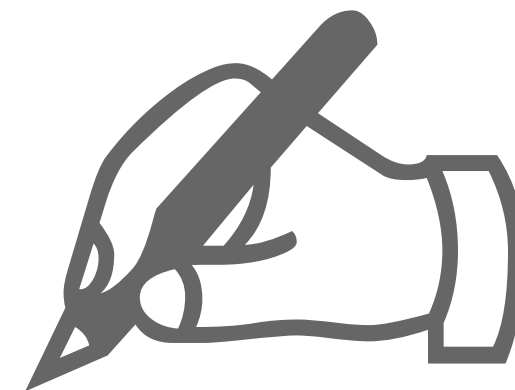
$\{\lambda s. inv\ s\}$
do $c_marshal\ a\ b;$
 $s_unmarshal\ a\ b;$
 $r \leftarrow s_max\ a\ b;$
 $s_marshal\ r;$
 $c_unmarshal\ r$
od
 $\{\lambda r\ s. inv\ s \wedge r = \text{if } a > b \text{ then } a \text{ else } b\}!$



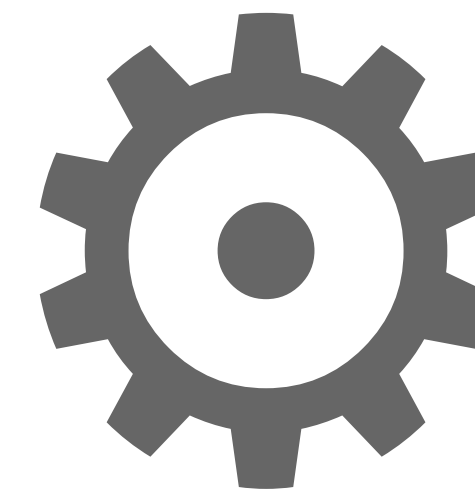
***What is
correctness?***



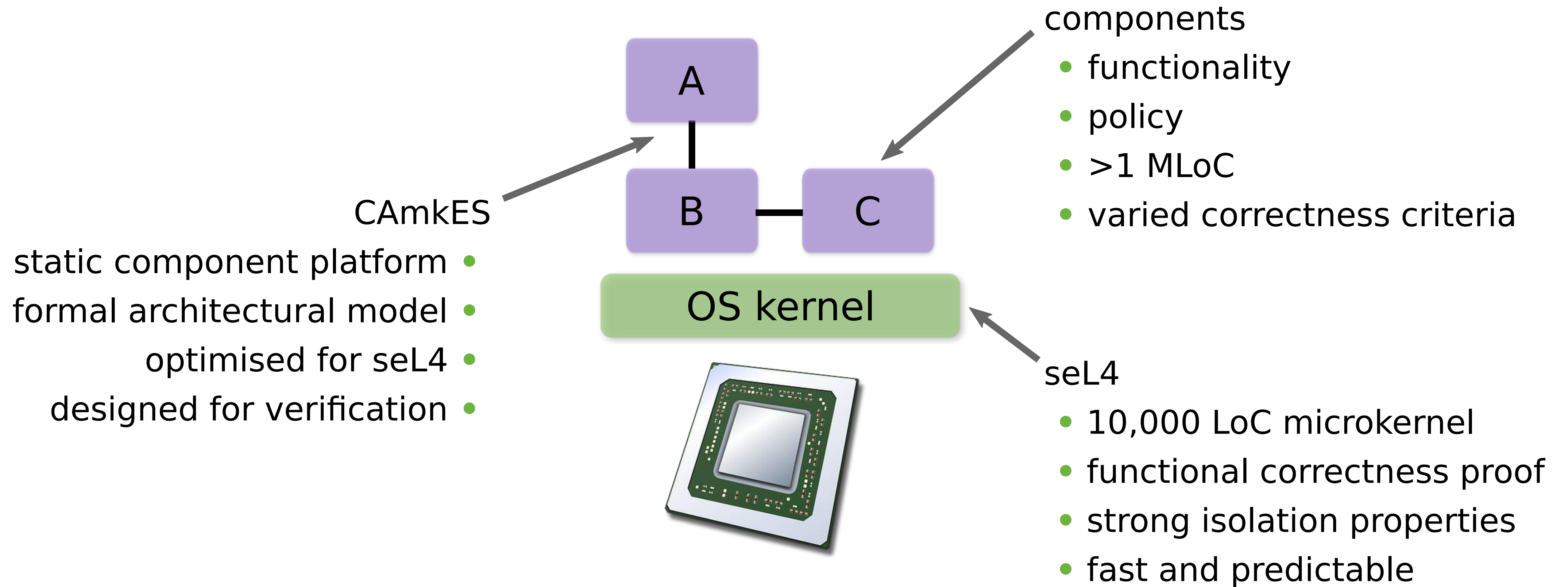
***How do we
prove it?***



***How do we
automate it?***

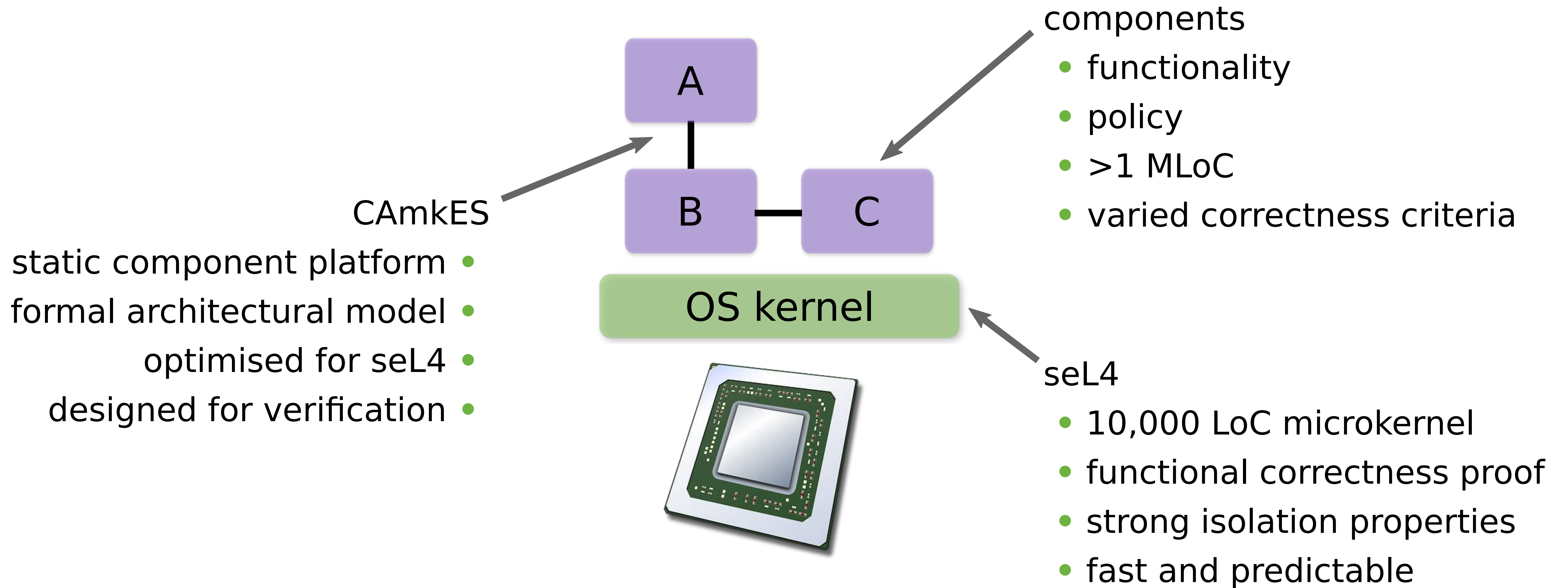


Generated Proofs for Generated Code



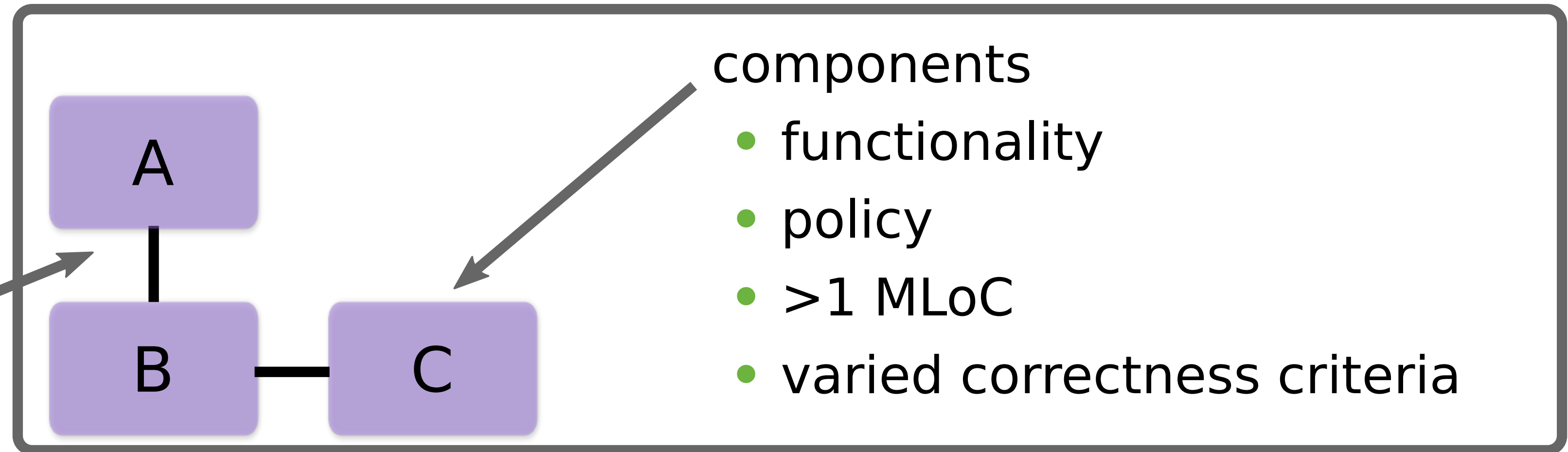
Generated Proofs for Generated Code

proof obligations



Generated Proofs for Generated Code

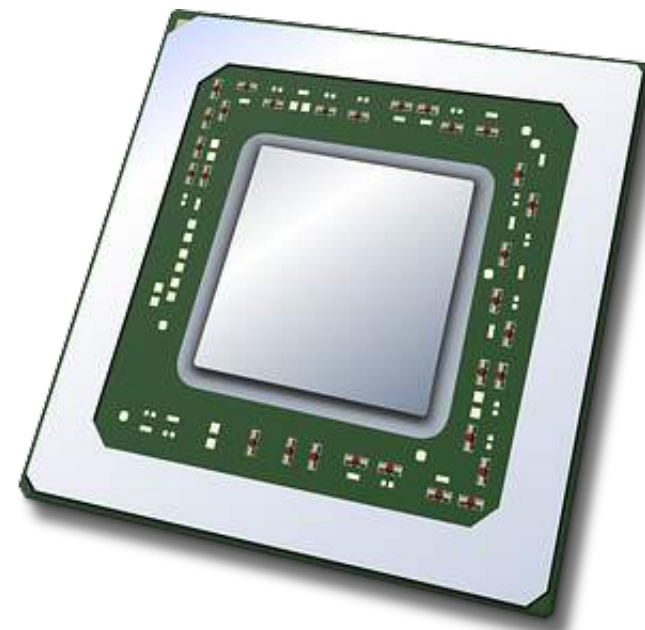
proof obligations



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