

---

---

# A Declarative Approach to C Program Specialization

Anne-Françoise Le Meur

Charles Consel

Compose group

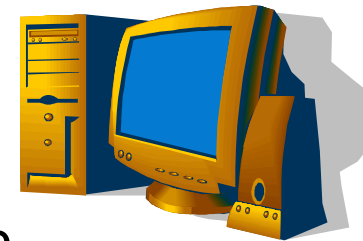
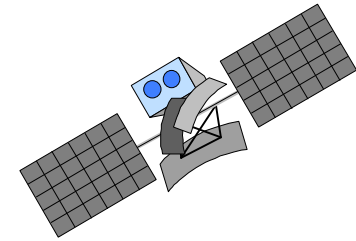
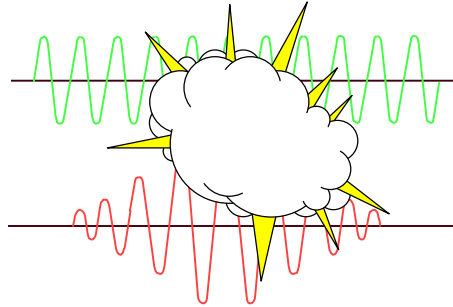
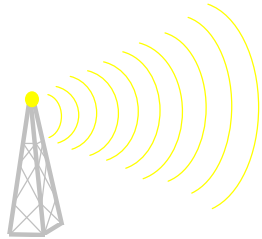
INRIA/ LaBRI, Bordeaux, France

# Context

---

- Software development
  - increasingly complex
  - maintenance is difficult and expensive
  - need for rapid development
- Goals
  - development of generic components
    - » basic blocks
    - » component corresponds to a given problem family
- Constraints
  - code size
  - execution time
- Solution: configuration

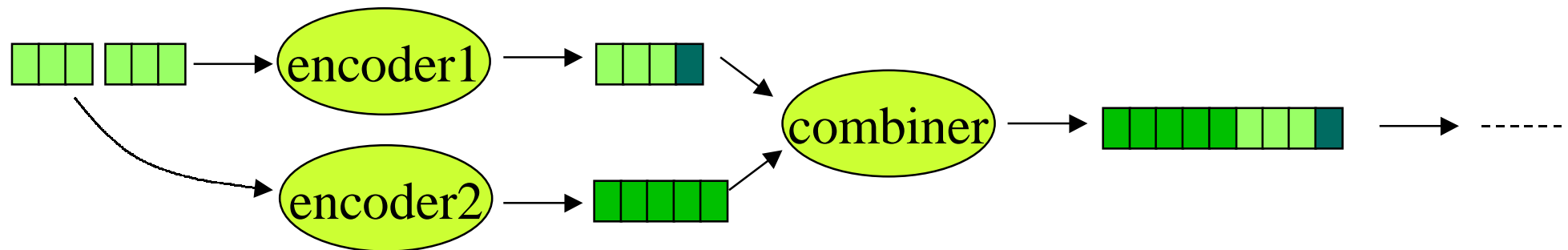
# An example problem family : forward error correction encoders



- Prevent losses and errors
- Transmit redundant information

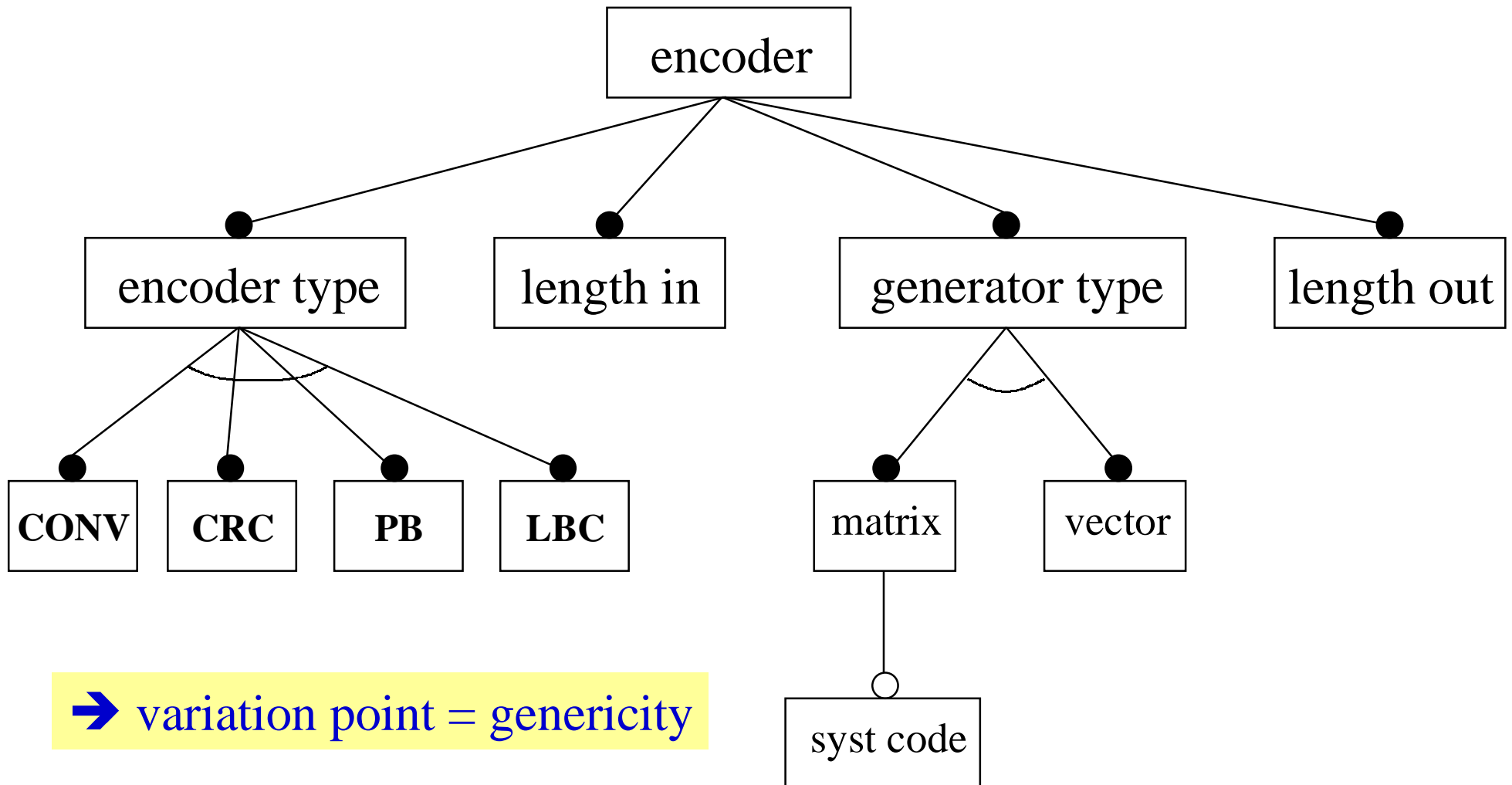
→ family of problems

# Forward Error Correction (FEC)



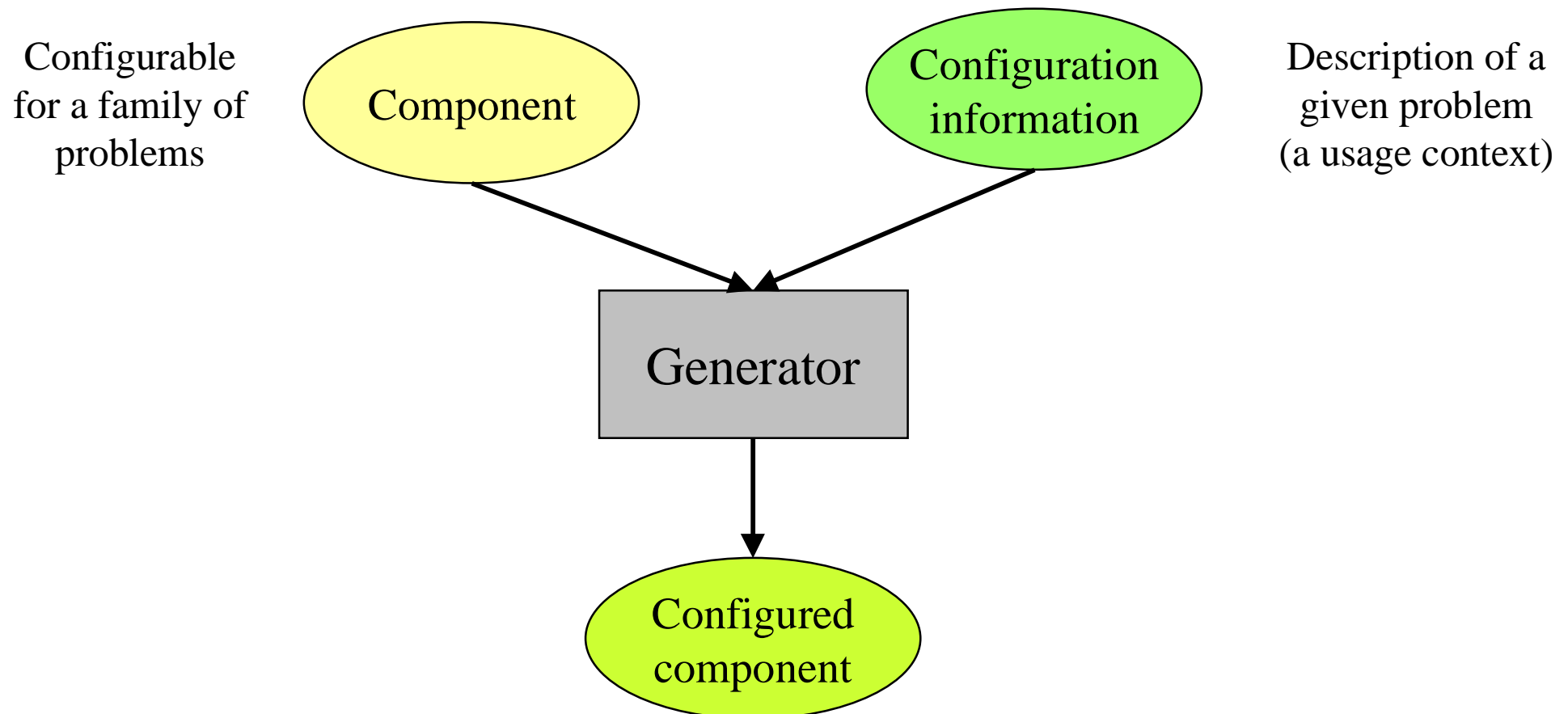
- Family of problems
  - Different encoders
    - » Types (properties)
    - » Parameters (e.g., amount of data redundancy)
  - Different degrees of criticality for transferred data
  - Different compositions of encoders

# Variation points



# Configuration of generic components

- Configurable component → family of problems
- Time and space constraints → configuration



# Software component configuration

---

- Functional configuration
  - restrict behavior
    - » ex: JavaBeans
- Code configuration
  - restrict behavior
  - reduce genericity
- 2 alternatives:
  - implement transformations
  - use a transformation generator

# Implement transformations

---

---

## – How :

- `ifdef` compiler directives
  - » code hard to read, hard to maintain, complex
- C++ Templates
  - » cumbersome, error-prone, difficult, no debugger

## – Limitation :

- the willingness of the programmer to encode the transformations

# Example: power

---

```
int power(int base, int expon) {  
    int accum = 1;  
    while (expon > 0) {  
        accum *= base;  
        expon - - ; }  
    return accum; }
```

# Example: power

---

```
int power(int base, int expon) {  
    int accum = 1;  
    while (expon > 0) {  
        accum *= base;  
        expon - - ; }  
    return accum; }
```

Code configuration

# Example: power

---

---

```
int power(int base, int expon) {  
    int accum = 1;  
    while (expon > 0) {  
        accum *= base;  
        expon - - ; }  
    return accum; }  

```

---

expon = 3

```
int power(int base){  
    return base*base*base;  
}
```

Code configuration

# Example: power

```
int power(int base, int expon) {  
    int accum = 1;  
    while (expon > 0) {  
        accum *= base;  
        expon - - ; }  
    return accum; }
```

**expon = 3**

```
int power(int base){  
    return base*base*base;  
}
```

```
template<int expon>  
inline int power(const int& base)  
{ return power<expon-1>(base) * base;}
```

```
template<>  
inline int power<1>(const int& base)  
{ return base;}
```

```
template<>  
inline int power<0>(const int& base)  
{ return 1;}
```

# Example: power

```
int power(int base, int expon) {  
    int accum = 1;  
    while (expon > 0) {  
        accum *= base;  
        expon - - ; }  
    return accum; }
```

**expon = 3**

```
int power(int base){  
    return base*base*base;  
}
```

```
template<int expon>  
inline int power(const int& base)  
{ return power<expon-1>(base) * base;}
```

```
template<>  
inline int power<1>(const int& base)  
{ return base;}
```

```
template<>  
inline int power<0>(const int& base)  
{ return 1;}
```

Poor readability

# Example: power

```
int power(int base, int expon) {  
    int accum = 1;  
    while (expon > 0) {  
        accum *= base;  
        expon - - ; }  
    return accum; }
```

**expon = 3**

```
int power(int base){  
    return base*base*base;  
}
```

```
template<int expon>  
inline int power(const int& base)  
{ return power<expon-1>(base) * base;}
```

```
template<>  
inline int power<1>(const int& base)  
{ return base;}
```

```
template<>  
inline int power<0>(const int& base)  
{ return 1;}
```

Transformation semantics

# Example: power

```
int power(int base, int expon) {  
    int accum = 1;  
    while (expon > 0) {  
        accum *= base;  
        expon - - ; }  
    return accum; }
```

expon = 3

```
int power(int base){  
    return base*base*base;  
}
```

```
template<int expon>  
inline int power(const int& base)  
{ return power<expon-1>(base) * base;}
```

```
template<>  
inline int power<1>(const int& base)  
{ return base;}
```

```
template<>  
inline int power<0>(const int& base)  
{ return 1;}
```

power<3>

# Example: power

```
int power(int base, int expon) {  
    int accum = 1;  
    while (expon > 0) {  
        accum *= base;  
        expon - - ; }  
    return accum; }
```

**expon = 3**

```
int power(int base){  
    return base*base*base;  
}
```

```
template<int expon>  
inline int power(const int& base)  
{ return power<expon-1>(base) * base;}
```

```
template<>  
inline int power<1>(const int& base)  
{ return base;}
```

```
template<>  
inline int power<0>(const int& base)  
{ return 1;}
```

**power<2>(base) \* base**

# Example: power

```
int power(int base, int expon) {
    int accum = 1;
    while (expon > 0) {
        accum *= base;
        expon - - ; }
    return accum; }
```

**expon = 3**

```
int power(int base){
    return base*base*base;
}
```

```
template<int expon>
inline int power(const int& base)
{ return power<expon-1>(base) * base;}
```

```
template<>
inline int power<1>(const int& base)
{ return base;}
```

```
template<>
inline int power<0>(const int& base)
{ return 1;}
```

`power<1>(base) * base * base`

# Example: power

```
int power(int base, int expon) {  
    int accum = 1;  
    while (expon > 0) {  
        accum *= base;  
        expon - - ; }  
    return accum; }
```

expon = 3

```
int power(int base){  
    return base*base*base;  
}
```

```
template<int expon>  
inline int power(const int& base)  
{ return power<expon-1>(base) * base;}
```

```
template<>  
inline int power<1>(const int& base)  
{ return base;}
```

```
template<>  
inline int power<0>(const int& base)  
{ return 1;}
```

base \* base \* base

# Example: power

```
int power(int base, int expon) {  
    int accum = 1;  
    while (expon > 0) {  
        accum *= base;  
        expon - - ; }  
    return accum; }  
  
expon = 3  
  
int power(int base){  
    return base*base*base;  
}
```

```
template<int expon>  
inline int power(const int& base)  
{ return power<expon-1>(base) * base;}
```

```
template<>  
inline int power<1>(const int& base)  
{ return base;}
```

```
template<>  
inline int power<0>(const int& base)  
{ return 1;}
```

Do we really get what we wanted ??

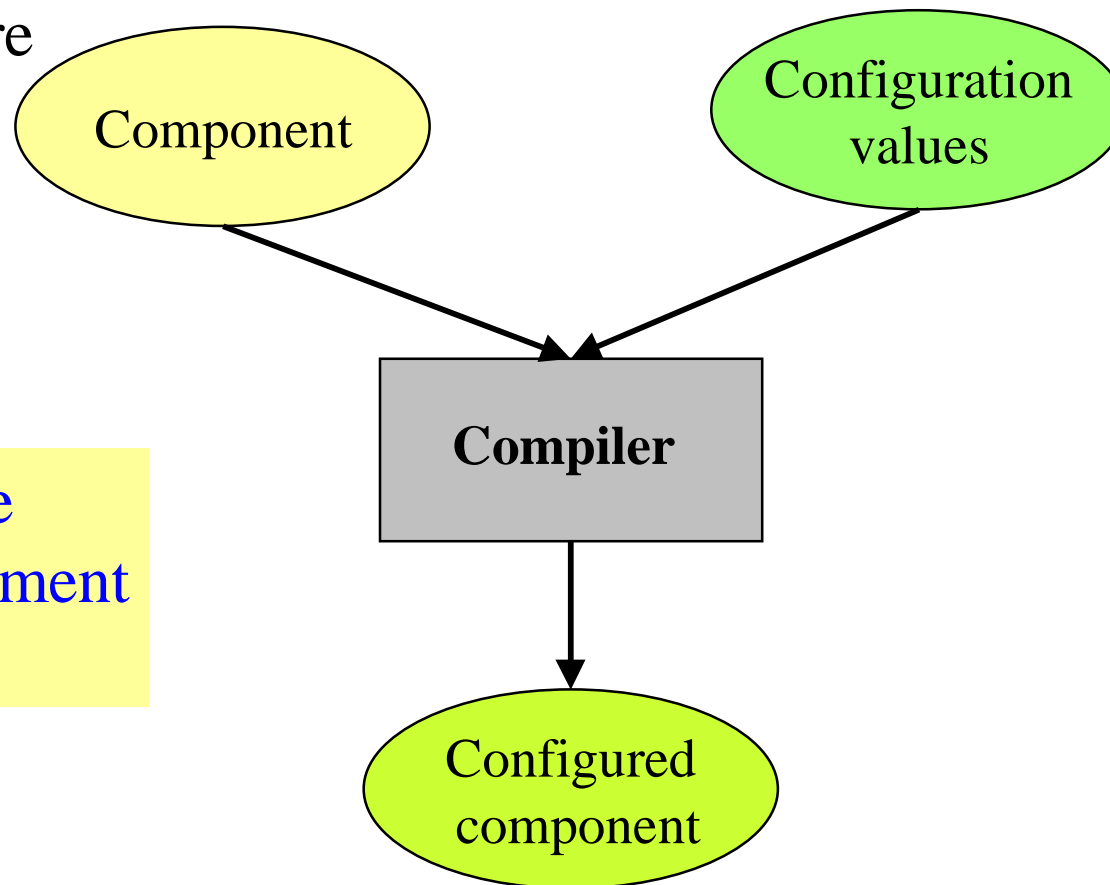
base \* base \* base

# Implement transformations: summary

---

---

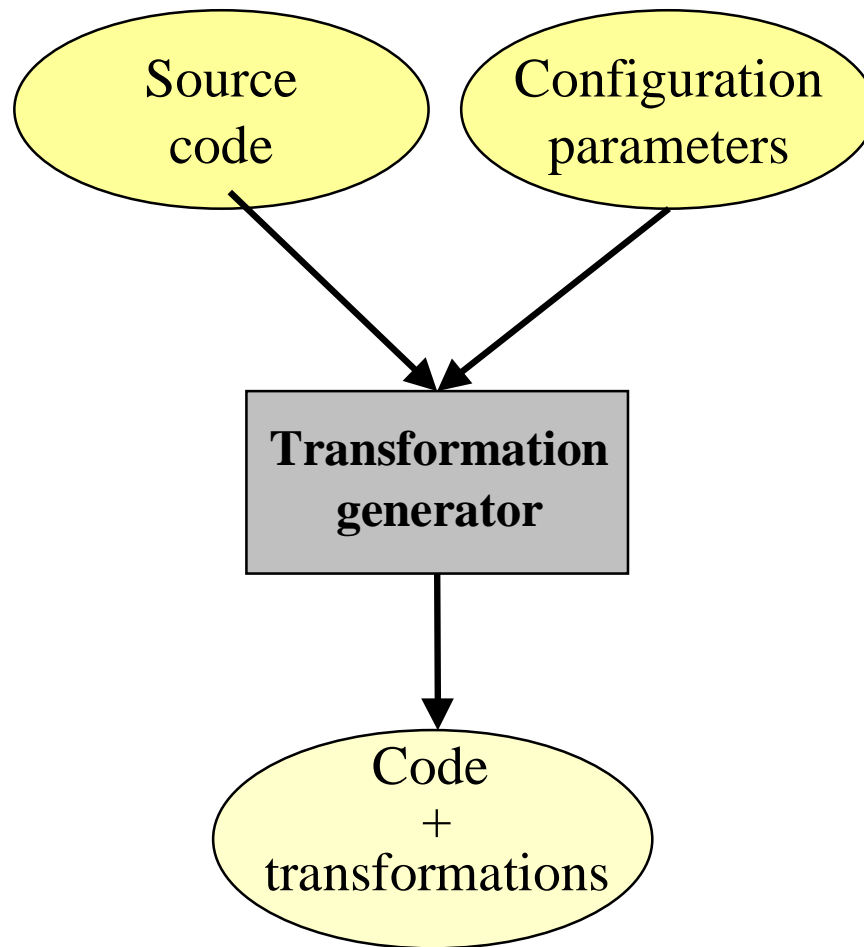
What to configure  
and how



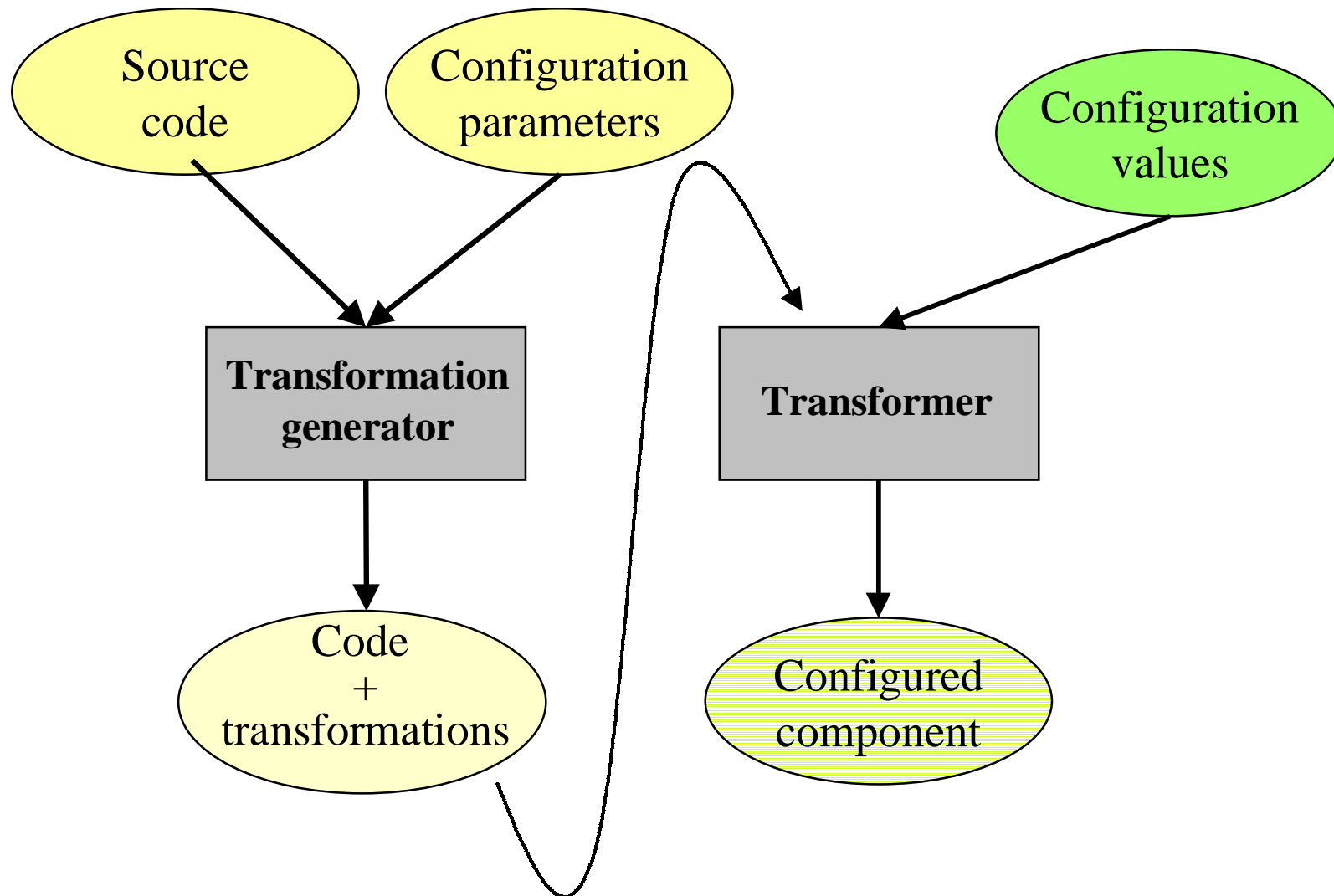
- Complex code
- Hard to implement
- No guarantee

# Transformation generator

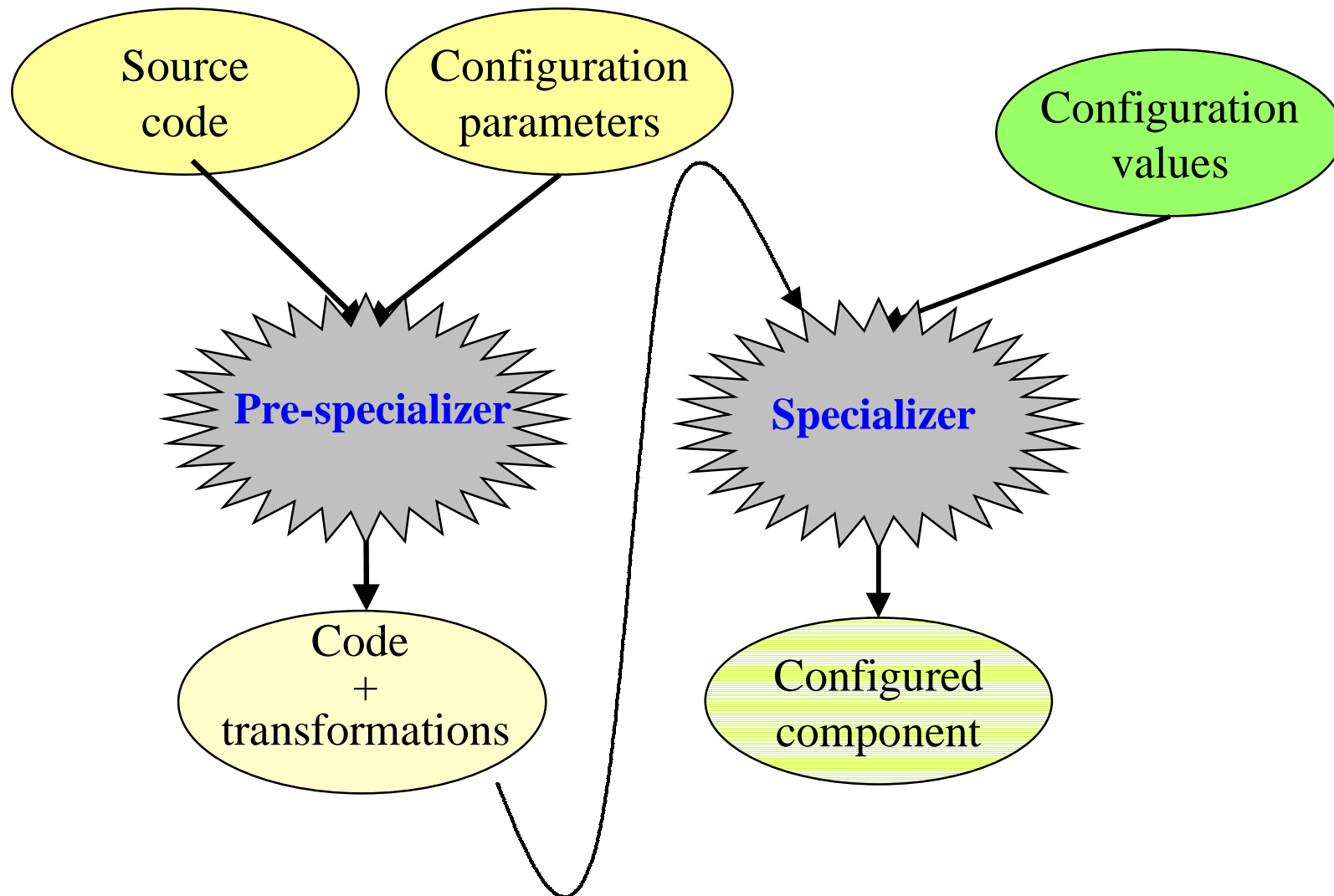
---



# Transformation generator



# Transformation generator



```

#include "encode.h"
#include "parity_bit.h"
#include "lin_b_code.h"
#include "lin_b_code_sys.h"
#include "code_conv.h"
#include "encode_crc.h"
#include "libstring.h"

void
encode(struct data *donnees, struct code *codage, struct data *result){
  if (donnees->longueur != codage->k)
    { perror("donnees->longueur != codage->k");
      exit(1); }
  else if (strcmp(codage->type, "parity_bit") == 0)
    if (result->longueur != codage->k + 1)
      { perror("result->longueur != codage->k + 1");
        exit(1); }
    else parity_bit(codage->k, donnees->v, result->v);
  else if (strcmp(codage->type, "lin_b_code") == 0)
    if ((result->longueur != (*codage).n)
        { perror("result->longueur != codage->n");
          exit(1); }
      else if (strcmp(codage->opt, "syst") == 0)
        lin_b_code_sys(donnees->v, codage->k, codage->n, codage->matrix, result->v);
      else
        lin_b_code(donnees->v, codage->k, codage->n, codage->matrix, result->v);
  else if (strcmp(codage->type, "code_conv") == 0)
    if ((result->longueur != (2 * (*codage).k))
        { perror("result->longueur != (2 * codage->k)");
          exit(1); }
      else code_conv(donnees->v, codage->k, result->v);
  else if (strcmp(codage->type, "crc") == 0)
    if ((result->longueur != (*codage).n)
        { perror("result->longueur != codage->n");
          exit(1); }
      else encode_crc(donnees->v, codage->k, codage->n, codage->matrix[0], result->v);
  else
    { perror("unknown code type");
      exit(1); }
}

```

```

struct data
{
  int longueur;
  int *v;
};

struct code
{
  int k;
  int n;
  int **matrix;
  char *type;
  char *opt;
};

extern void encode(struct data *donnees, struct code *codage, struct data *result);

```

```
extern int strcmp(char *s1, char *s2);
```

```
extern void encode_crc(int *donnees, int k, int n, int *poly, int *result);
```

```

#include "encode_crc.h"
void
encode_crc(int *donnees, int k, int n, int *poly, int *result)
{
  int l_poly;
  int i, j;

  l_poly = n - k + 1;
  for (i = 0; i < k; i++)
    result[i] = donnees[i];
  for (i = 0; i < k; i++)
    if (result[i] == 1)
      for (j = 0; j < l_poly; j++)
        result[i + j] ^= poly[j];
  for (i = 0; i < k; i++)
    result[i] = donnees[i];
}

```

```
extern void lin_b_code(int *vector, int k, int n, int **matrix, int *result);
```

```

#include "lin_b_code.h"
#include "mult_mat.h"
void
lin_b_code(int *vector, int k, int n, int **matrix, int *result)
{
  mult_mat(vector, k, n, matrix, result, 0);
}

```

```
extern void code_conv(int *donnees, int l, int *result);
```

```

#include "code_conv.h"
void
code_conv(int *donnees, int k, int *result)
{
  int temp;
  int i;
  for (i = 0; i < k - 2; i++)
    {
      temp = donnees[i] ^ donnees[i + 2];
      result[2 * i + 1] = temp;
      temp ^= donnees[i + 1];
      result[2 * i] = temp;
    }
  temp = donnees[k - 2] ^ donnees[k - 1];
  result[2 * k - 4] = temp;
  result[2 * k - 3] = donnees[k - 2];
  result[2 * k - 2] = donnees[k - 1];
  result[2 * k - 1] = donnees[k - 1];
}

```

```
extern void lin_b_code_sys(int *vector, int k, int n, int **matrix, int *result);
```

```

#include "lin_b_code_sys.h"
#include "mult_mat.h"
void
lin_b_code_sys(int *vector, int k, int n, int **matrix, int *result)
{
  int sub_matrix_size;
  int i;

  sub_matrix_size = n - k;
  mult_mat(vector, k, sub_matrix_size, matrix, result, k);
  for (i = 0; i < k; i++)
    result[i] = vector[i];
}

```

```
extern void mult_mat(int *vector, int k, int n, int **matrix, int *result, int indice);
```

```

#include "mult_mat.h"
void
mult_mat(int *vector, int k, int n, int **matrix, int *result, int indice)
{
  int temp;
  int i, j;

  for (i = 0; i < n; i++)
    {
      temp = 0;

      for (j = 0; j < k; j++)
        temp ^= vector[j] & matrix[j][i];

      result[i + indice] = temp;
    }
}

```

```
extern void parity_bit(int l, int *vector, int *result);
```

```

#include "parity_bit.h"
void
parity_bit(int l, int *vector, int *result)
{
  int res;
  int i;
  res = 0;

  for (i = 0; i < l; i++)
    {
      res ^= vector[i];
      result[i] = vector[i];
    }
  result[l] = res;
}

```

# Configuration parameters

length in  
length out  
type  
generator  
opt

```
#include "encode.h"
#include "parity_bit.h"
#include "lin_b_code.h"
#include "lin_b_code_sys.h"
#include "code_conv.h"
#include "encode_crc.h"
#include "libstring.h"

void
encode(struct data *donnees, struct code *codage, struct data *result){
  if (donnees->longueur != codage->k)
    { perror("donnees->longueur != codage->k");
      exit(1); }
  else if (strcmp(codage->type, "parity_bit") == 0)
    if (result->longueur != codage->k + 1)
      { perror("result->longueur != codage->k + 1");
        exit(1); }
    else parity_bit(codage->k, donnees->v, result->v);
  else if (strcmp(codage->type, "lin_b_code") == 0)
    if ((result->longueur != (*codage).n)
        { perror("result->longueur != codage->n");
          exit(1); }
    else if (strcmp(codage->opt, "sys") == 0)
      lin_b_code_sys(donnees->v, codage->k, codage->n, codage->matrix, result->v);
    else
      lin_b_code(donnees->v, codage->k, codage->n, codage->matrix, result->v);
  else if (strcmp(codage->type, "code_conv") == 0)
    if ((result->longueur != (2 * (*codage).k))
        { perror("result->longueur != (2 * codage->k)");
          exit(1); }
    else code_conv(donnees->v, codage->k, result->v);
  else if (strcmp(codage->type, "crc") == 0)
    if ((result->longueur != (*codage).n)
        { perror("result->longueur != codage->n");
          exit(1); }
    else encode_crc(donnees->v, codage->k, codage->n, codage->matrix[0], result->v);
  else
    { perror("unknown code type");
      exit(1); }
}
```

```
struct data
{
  int longueur;
  int *v;
};

struct code
{
  int k;
  int n;
  int **matrix;
  char *type;
  char *opt;
};

extern void encode(struct data *donnees, struct code *codage, struct data *result);
```

```
extern int strcmp(char *s1, char *s2);
```

```
extern void encode_crc(int *donnees, int k, int n, int *poly, int *result);
```

```
#include "encode_crc.h"
void
encode_crc(int *donnees, int k, int n, int *poly, int *result)
{
  int l_poly;
  int i, j;

  l_poly = n - k + 1;
  for (i = 0; i < k; i++)
    result[i] = donnees[i];
  for (i = 0; i < k; i++)
    if (result[i] == 1)
      for (j = 0; j < l_poly; j++)
        result[i + j] ^= poly[j];
  for (i = 0; i < k; i++)
    result[i] = donnees[i];
}
```

```
extern void lin_b_code(int *vector, int k, int n, int **matrix, int *result);
```

```
#include "lin_b_code.h"
#include "mult_mat.h"
void
lin_b_code(int *vector, int k, int n, int **matrix, int *result)
{
  mult_mat(vector, k, n, matrix, result, 0);
}
```

```
extern void code_conv(int *donnees, int l, int *result);
```

```
#include "code_conv.h"
void
code_conv(int *donnees, int k, int *result)
{
  int temp;
  int i;
  for (i = 0; i < k - 2; i++)
    {
      temp = donnees[i] ^ donnees[i + 2];
      result[2 * i + 1] = temp;
      temp ^= donnees[i + 1];
      result[2 * i] = temp;
    }
  temp = donnees[k - 2] ^ donnees[k - 1];
  result[2 * k - 4] = temp;
  result[2 * k - 3] = donnees[k - 2];
  result[2 * k - 2] = donnees[k - 1];
  result[2 * k - 1] = donnees[k - 1];
}
```

```
extern void lin_b_code_sys(int *vector, int k, int n, int **matrix, int *result);
```

```
#include "lin_b_code_sys.h"
#include "mult_mat.h"
void
lin_b_code_sys(int *vector, int k, int n, int **matrix, int *result)
{
  int sub_matrix_size;
  int i;

  sub_matrix_size = n - k;
  mult_mat(vector, k, sub_matrix_size, matrix, result, k);
  for (i = 0; i < k; i++)
    result[i] = vector[i];
}
```

```
extern void mult_mat(int *vector, int k, int n, int **matrix, int *result, int indice);
```

```
#include "mult_mat.h"
void
mult_mat(int *vector, int k, int n, int **matrix, int *result, int indice)
{
  int temp;
  int i, j;

  for (i = 0; i < n; i++)
    {
      temp = 0;

      for (j = 0; j < k; j++)
        temp ^= vector[j] & matrix[j][i];

      result[i + indice] = temp;
    }
}
```

```
extern void parity_bit(int l, int *vector, int *result);
```

```
#include "parity_bit.h"
void
parity_bit(int l, int *vector, int *result)
{
  int res;
  int i;
  res = 0;

  for (i = 0; i < l; i++)
    {
      res ^= vector[i];
      result[i] = vector[i];
    }
  result[l] = res;
}
```

# Configuration parameters

length in  
length out  
type  
generator  
opt

```
#include "encode.h"
#include "parity_bit.h"
#include "lin_b_code.h"
#include "lin_b_code_sys.h"
#include "code_conv.h"
#include "encode_crc.h"
#include "libstring.h"

void
encode(struct data *donnees, struct code *codage, struct data *result){
  if (donnees->longueur != codage->k)
    { perror("donnees->longueur != codage->k");
      exit(1); }
  else if (strcmp(codage->type, "parity_bit") == 0)
    if (result->longueur != codage->k + 1)
      { perror("result->longueur != codage->k + 1");
        exit(1); }
    else parity_bit(codage->k, donnees->v, result->v);
  else if (strcmp(codage->type, "lin_b_code") == 0)
    if ((result->longueur != (*codage).n)
        { perror("result->longueur != codage->n");
          exit(1); }
    else if (strcmp(codage->opt, "sys") == 0)
      lin_b_code_sys(donnees->v, codage->k, codage->n, codage->matrix, result->v);
    else
      lin_b_code(donnees->v, codage->k, codage->n, codage->matrix, result->v);
  else if (strcmp(codage->type, "code_conv") == 0)
    if ((result->longueur != (2 * (*codage).k))
        { perror("result->longueur != (2 * codage->k)");
          exit(1); }
    else code_conv(donnees->v, codage->k, result->v);
  else if (strcmp(codage->type, "crc") == 0)
    if ((result->longueur != (*codage).n)
        { perror("result->longueur != codage->n");
          exit(1); }
    else encode_crc(donnees->v, codage->k, codage->n, codage->matrix[0], result->v);
  else
    { perror("unknown code type");
      exit(1); }
}
```

```
struct data
{
  int longueur;
  int *v;
};

struct code
{
  int k;
  int n;
  int **matrix;
  char *type;
  char *opt;
};

extern void encode(struct data *donnees, struct code *codage, struct data *result);
```

```
extern int strcmp(char *s1, char *s2);
```

```
extern void encode_crc(int *donnees, int k, int n, int *poly, int *result);
```

```
#include "encode_crc.h"
void
encode_crc(int *donnees, int k, int n, int *poly, int *result)
{
  int l_poly;
  int i, j;

  l_poly = n - k + 1;
  for (i = 0; i < k; i++)
    result[i] = donnees[i];
  for (i = 0; i < k; i++)
    if (result[i] == 1)
      for (j = 0; j < l_poly; j++)
        result[i + j] ^= poly[j];
  for (i = 0; i < k; i++)
    result[i] = donnees[i];
}
```

```
extern void lin_b_code(int *vector, int k, int n, int **matrix, int *result);
```

```
#include "lin_b_code.h"
#include "mult_mat.h"
void
lin_b_code(int *vector, int k, int n, int **matrix, int *result)
{
  mult_mat(vector, k, n, matrix, result, 0);
}
```

```
extern void code_conv(int *donnees, int l, int *result);
```

```
#include "code_conv.h"
void
code_conv(int *donnees, int k, int *result)
{
  int temp;
  int i;
  for (i = 0; i < k - 2; i++)
    {
      temp = donnees[i] ^ donnees[i + 2];
      result[2 * i + 1] = temp;
      temp ^= donnees[i + 1];
      result[2 * i] = temp;
    }
  temp = donnees[k - 2] ^ donnees[k - 1];
  result[2 * k - 4] = temp;
  result[2 * k - 3] = donnees[k - 2];
  result[2 * k - 2] = donnees[k - 1];
  result[2 * k - 1] = donnees[k - 1];
}
```

```
extern void lin_b_code_sys(int *vector, int k, int n, int **matrix, int *result);
```

```
#include "lin_b_code_sys.h"
#include "mult_mat.h"
void
lin_b_code_sys(int *vector, int k, int n, int **matrix, int *result)
{
  int sub_matrix_size;
  int i;

  sub_matrix_size = n - k;
  mult_mat(vector, k, sub_matrix_size, matrix, result, k);
  for (i = 0; i < k; i++)
    result[i] = vector[i];
}
```

```
extern void mult_mat(int *vector, int k, int n, int **matrix, int *result, int indice);
```

```
#include "mult_mat.h"
void
mult_mat(int *vector, int k, int n, int **matrix, int *result, int indice)
{
  int temp;
  int i, j;

  for (i = 0; i < n; i++)
    {
      temp = 0;

      for (j = 0; j < k; j++)
        temp ^= vector[j] & matrix[j][i];

      result[i + indice] = temp;
    }
}
```

```
extern void parity_bit(int l, int *vector, int *result);
```

```
#include "parity_bit.h"
void
parity_bit(int l, int *vector, int *result)
{
  int res;
  int i;
  res = 0;
  for (i = 0; i < l; i++)
    {
      res ^= vector[i];
      result[i] = vector[i];
    }
  result[l] = res;
}
```



# Configuration values

length in = 4

length out = 7

type = "lin\_b\_code"

matrix =  $\begin{Bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{Bmatrix}$

opt = "syst"

```
#include "encode.h"
#include "parity_bit.h"
#include "lin_b_code.h"
#include "lin_b_code_sys.h"
#include "code_conv.h"
#include "encode_crc.h"
#include "libstring.h"

void
encode(struct data *donnees, struct code *codage, struct data *result){
  if (donnees->longueur != codage->k)
    { perror("donnees->longueur != codage->k");
      exit(1); }
  else if (strcmp(codage->type, "parity_bit") == 0)
    if (result->longueur != codage->k + 1)
      { perror("result->longueur != codage->k + 1");
        exit(1); }
    else parity_bit(codage->k, donnees->v, result->v);
  else if (strcmp(codage->type, "lin_b_code") == 0)
    if ((result->longueur != (*codage).n)
        { perror("result->longueur != codage->n");
          exit(1); }
        else if (strcmp(codage->opt, "syst") == 0)
          lin_b_code_sys(donnees->v, codage->k, codage->n, codage->matrix, result->v);
        else
          lin_b_code(donnees->v, codage->k, codage->n, codage->matrix, result->v);
  else if (strcmp(codage->type, "code_conv") == 0)
    if ((result->longueur != (2 * (*codage).k))
        { perror("result->longueur != (2 * codage->k)");
          exit(1); }
        else code_conv(donnees->v, codage->k, result->v);
  else if (strcmp(codage->type, "crc") == 0)
    if ((result->longueur != (*codage).n)
        { perror("result->longueur != codage->n");
          exit(1); }
        else encode_crc(donnees->v, codage->k, codage->n, codage->matrix[0], result->v);
  else
    { perror("unknown code type");
      exit(1); }
}
```

```
struct data
{
  int longueur;
  int *v;
};

struct code
{
  int k;
  int n;
  int **matrix;
  char *type;
  char *opt;
};

extern void encode(struct data *donnees, struct code *codage, struct data *result);
```

```
extern int strcmp(char *s1, char *s2);
```

```
extern void parity_bit(int l, int *vector, int *result);
```

```
#include "parity_bit.h"
void
parity_bit(int l, int *vector, int *result)
{
  int res;
  int i;
  res = 0;
  for(i = 0; i < l; i++)
    {
      res ^= vector[i];
      result[i] = vector[i];
    }
  result[l] = res;
}
```

```
extern void lin_b_code_sys(int *vector, int k, int n, int **matrix, int *result);
```

```
#include "lin_b_code_sys.h"
#include "mult_mat.h"
void
lin_b_code_sys(int *vector, int k, int n, int **matrix, int *result)
{
  int sub_matrix_size;
  int i;
  sub_matrix_size = n - k;
  mult_mat(vector, k, sub_matrix_size, matrix, result, k);
  for(i = 0; i < k; i++)
    result[i] = vector[i];
}
```

```
extern void code_conv(int *donnees, int l, int *result);
```

```
#include "code_conv.h"
void
code_conv(int *donnees, int k, int *result)
{
  int temp;
  int i;
  for (i = 0; i < k - 2; i++)
    {
      temp = donnees[i] ^ donnees[i + 2];
      result[2 * i + 1] = temp;
      temp ^= donnees[i + 1];
      result[2 * i] = temp;
    }
  temp = donnees[k - 2] ^ donnees[k - 1];
  result[2 * k - 4] = temp;
  result[2 * k - 3] = donnees[k - 2];
  result[2 * k - 2] = donnees[k - 1];
  result[2 * k - 1] = donnees[k - 1];
}
```

```
extern void encode_crc(int *donnees, int k, int n, int *poly, int *result);
```

```
#include "encode_crc.h"
void
encode_crc(int *donnees, int k, int n, int *poly, int *result)
{
  int l_poly;
  int i, j;
  l_poly = n - k + 1;
  for (i = 0; i < k; i++)
    result[i] = donnees[i];
  for (i = 0; i < k; i++)
    if (result[i] == 1)
      for (j = 0; j < l_poly; j++)
        result[i + j] ^= poly[j];
  for (i = 0; i < k; i++)
    result[i] = donnees[i];
}
```

```
extern void lin_b_code(int *vector, int k, int n, int **matrix, int *result);
```

```
#include "lin_b_code.h"
#include "mult_mat.h"
void
lin_b_code(int *vector, int k, int n, int **matrix, int *result)
{
  mult_mat(vector, k, n, matrix, result, 0);
}
```

```
extern void mult_mat(int *vector, int k, int n, int **matrix, int *result, int indice);
```

```
#include "mult_mat.h"
void
mult_mat(int *vector, int k, int n, int **matrix, int *result, int indice)
{
  int temp;
  int i, j;
  for(i = 0; i < n; i++)
    {
      temp = 0;
      for(j = 0; j < k; j++)
        temp ^= vector[j] & matrix[j][i];
      result[i + indice] = temp;
    }
}
```



# Configured component

```
struct data { int longueur;
              int *v; };
struct code { int k;
              int n;
              int **matrix;
              char *type;
              char *opt; };
extern int perror();
extern int exit();
extern void encode_spe(struct data *, struct data *);

extern void encode_spe/*0*/(struct data *donnees, struct data *result)
{
  {
    int *lin_b_code_sys_0_result;
    int *lin_b_code_sys_0_vector;

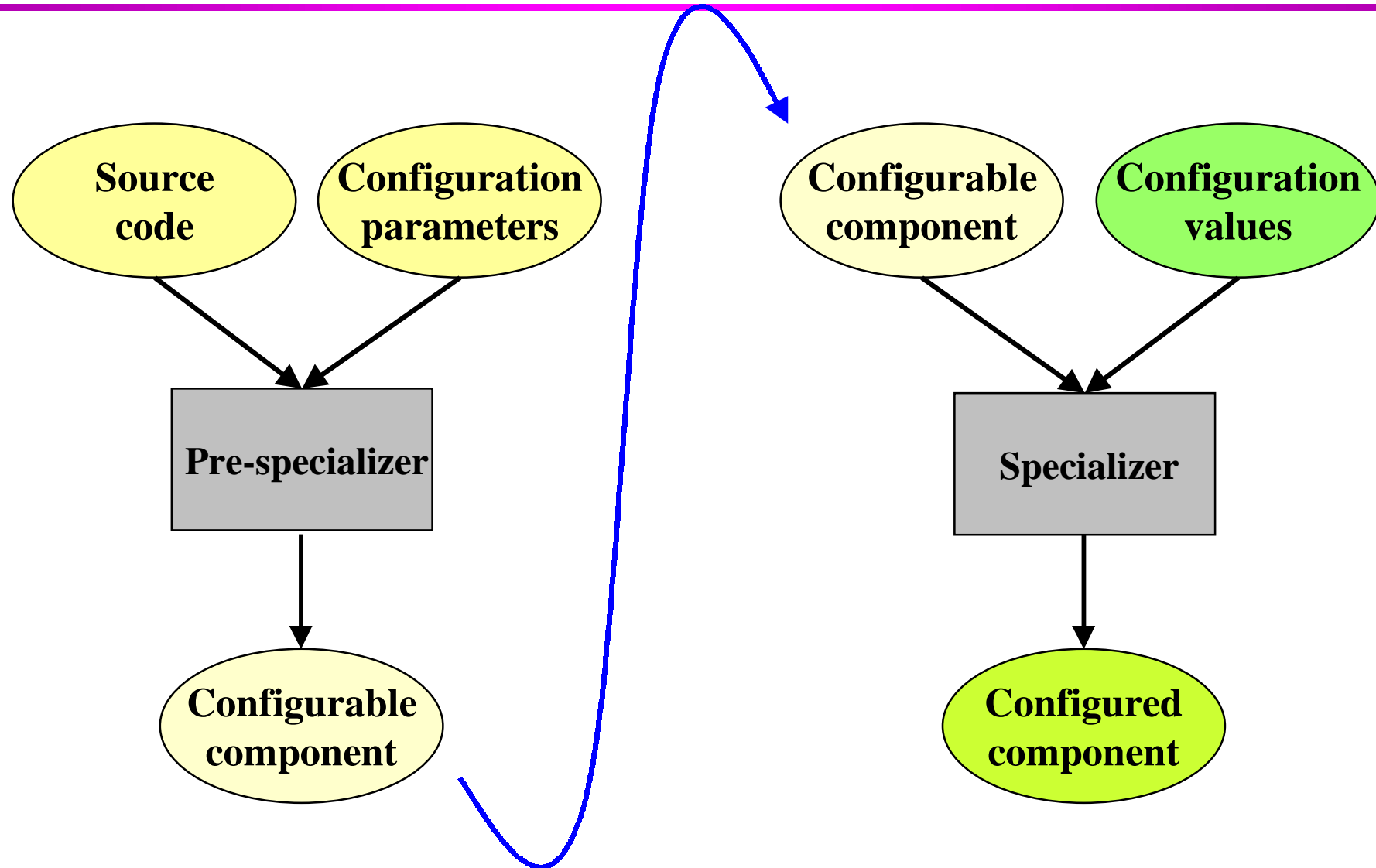
    lin_b_code_sys_0_vector = (*donnees).v;
    lin_b_code_sys_0_result = (*result).v;
    {
      int *mult_mat_1_result;
      int *mult_mat_1_vector;
      int mult_mat_1_temp;

      mult_mat_1_vector = lin_b_code_sys_0_vector;
      mult_mat_1_result = lin_b_code_sys_0_result;
      mult_mat_1_temp = (((0 ^ (unsigned int)((int)((unsigned int)mult_mat_1_vector[0] & 1))) ^
                          (unsigned int)((int)((unsigned int)mult_mat_1_vector[1] & 1))) ^
                          (unsigned int)((int)((unsigned int)mult_mat_1_vector[2] & 1))) ^
                          (unsigned int)((int)((unsigned int)mult_mat_1_vector[3] & 0)));
      mult_mat_1_result[4] = mult_mat_1_temp;
      mult_mat_1_temp = (((0 ^ (unsigned int)((int)((unsigned int)mult_mat_1_vector[0] & 1))) ^
                          (unsigned int)((int)((unsigned int)mult_mat_1_vector[1] & 1))) ^
                          (unsigned int)((int)((unsigned int)mult_mat_1_vector[2] & 0))) ^
                          (unsigned int)((int)((unsigned int)mult_mat_1_vector[3] & 1)));
      mult_mat_1_result[5] = mult_mat_1_temp;
      mult_mat_1_temp = (((0 ^ (unsigned int)((int)((unsigned int)mult_mat_1_vector[0] & 1))) ^
                          (unsigned int)((int)((unsigned int)mult_mat_1_vector[1] & 0))) ^
                          (unsigned int)((int)((unsigned int)mult_mat_1_vector[2] & 0))) ^
                          (unsigned int)((int)((unsigned int)mult_mat_1_vector[3] & 1)));
      mult_mat_1_result[6] = mult_mat_1_temp;
    }
    lin_b_code_sys_0_result[0] = lin_b_code_sys_0_vector[0];
    lin_b_code_sys_0_result[1] = lin_b_code_sys_0_vector[1];
    lin_b_code_sys_0_result[2] = lin_b_code_sys_0_vector[2];
    lin_b_code_sys_0_result[3] = lin_b_code_sys_0_vector[3];
  }
  return;
}
```

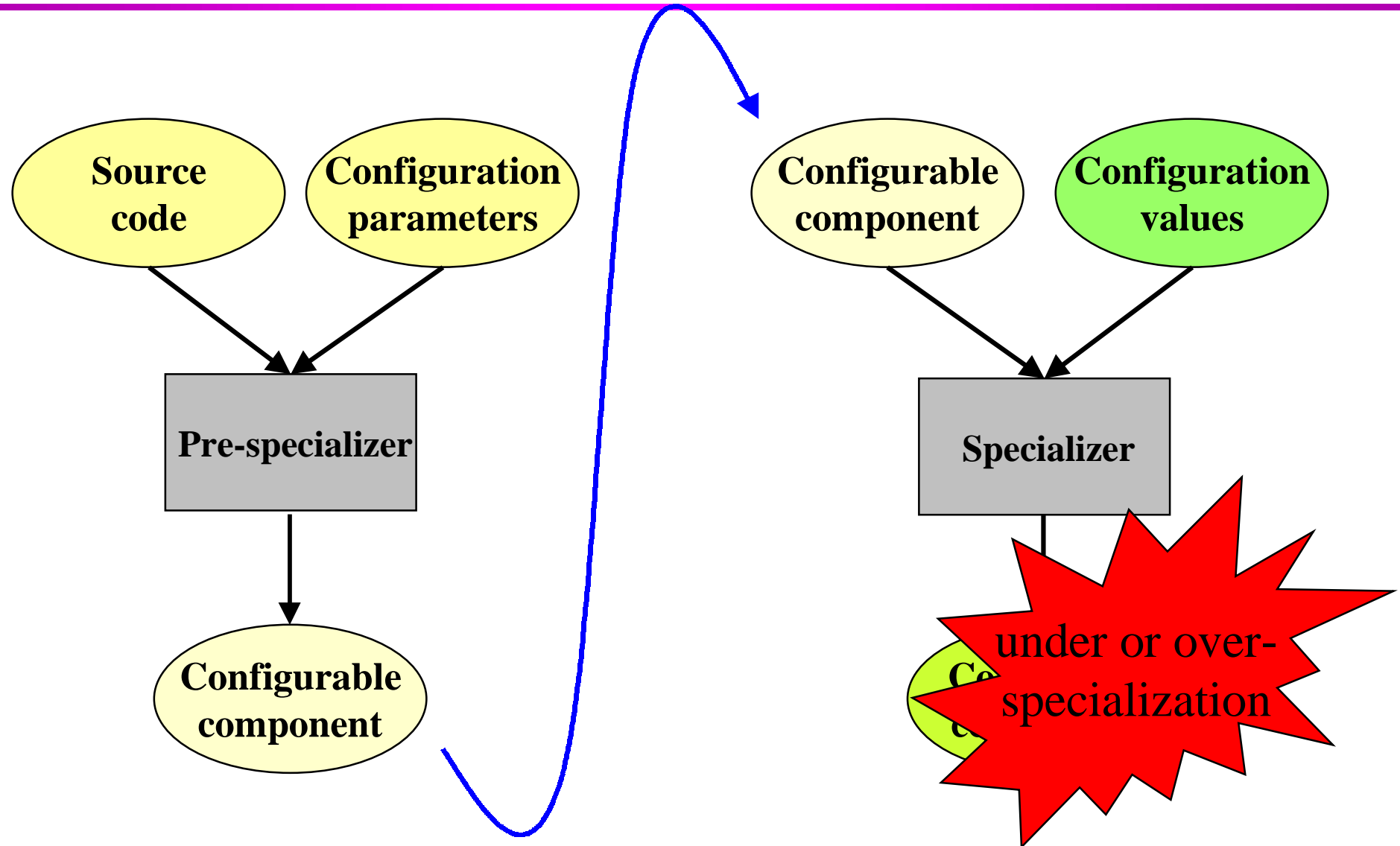
# Configuration using specialization

---

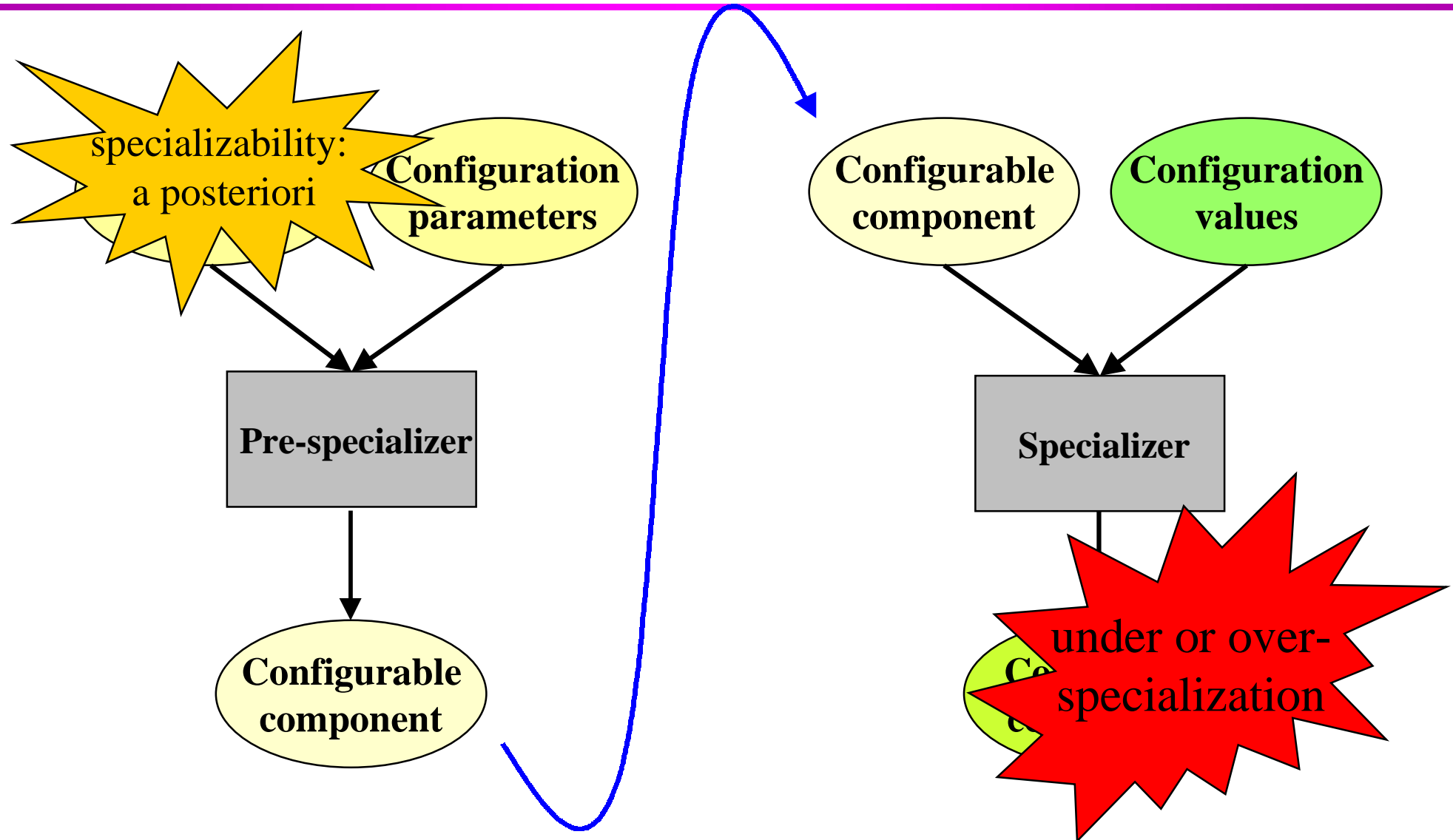
---



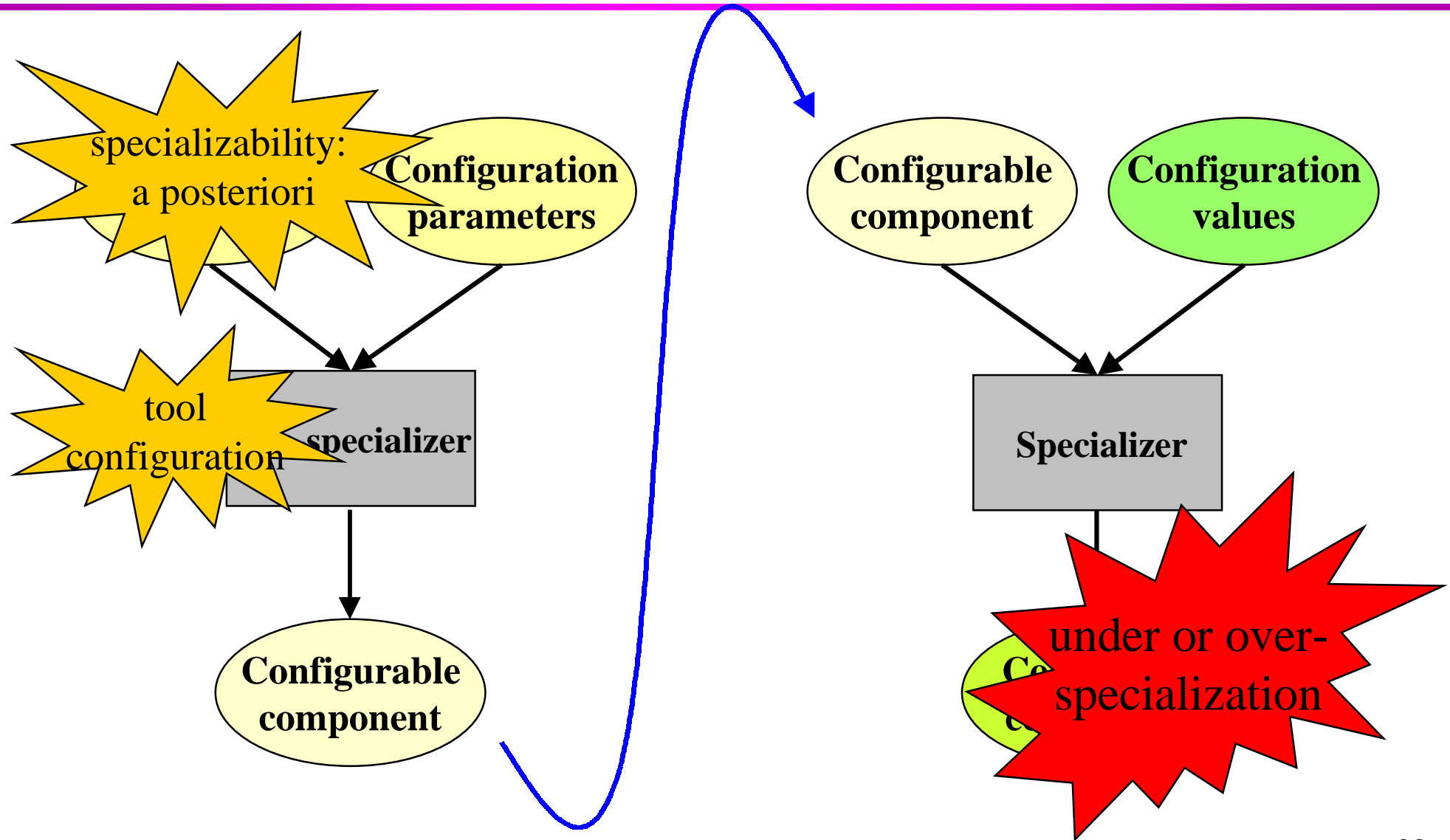
# Configuration using specialization



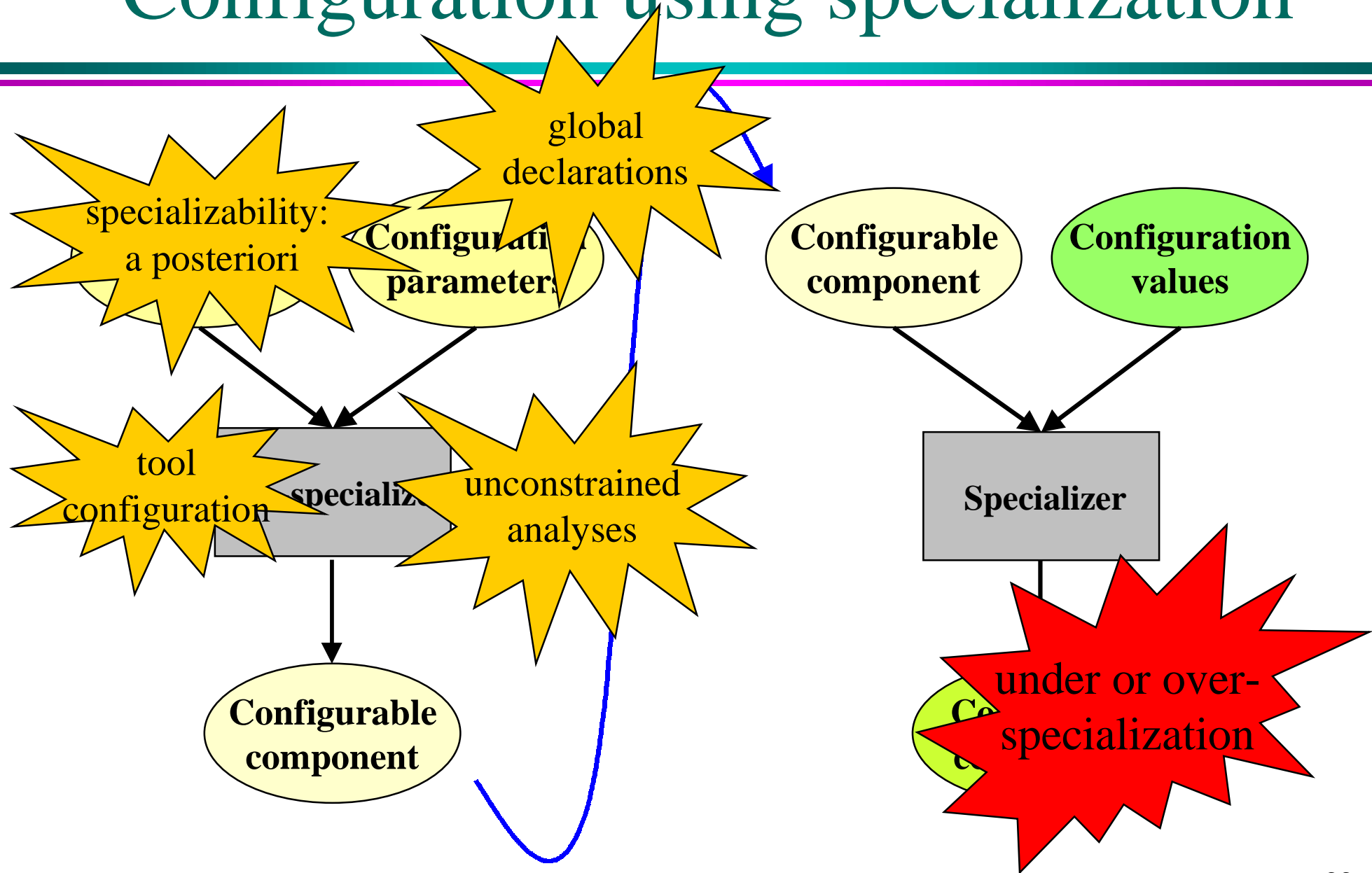
# Configuration using specialization



# Configuration using specialization



# Configuration using specialization



# Thesis

---

---

- Make specialization predictable and easier to use
- Declarative approach
  - specialization declarations
  - developer's intention
  - specialization verifications
    - » feasibility of the desired specialization

# Problems to solve

---

---

- Specializability of a program
  - a posteriori
- Complex mechanisms
  - tool configuration difficult
- Difference between the result and the developer's expectation
  - global specialization information
  - unconstrained analyses

# Proposed solutions (1/3)

---

- Specializability of a program
  - a posteriori



- Take specialization into account during program development
  - what code fragments to specialize
  - in what context
    - Declarations: *specialization scenarios*

# Proposed solutions (2/3)

---

- Complex mechanisms
  - tool configuration difficult



- Translate declarations into configuration parameters accepted by the specializer

# Proposed solutions (3/3)

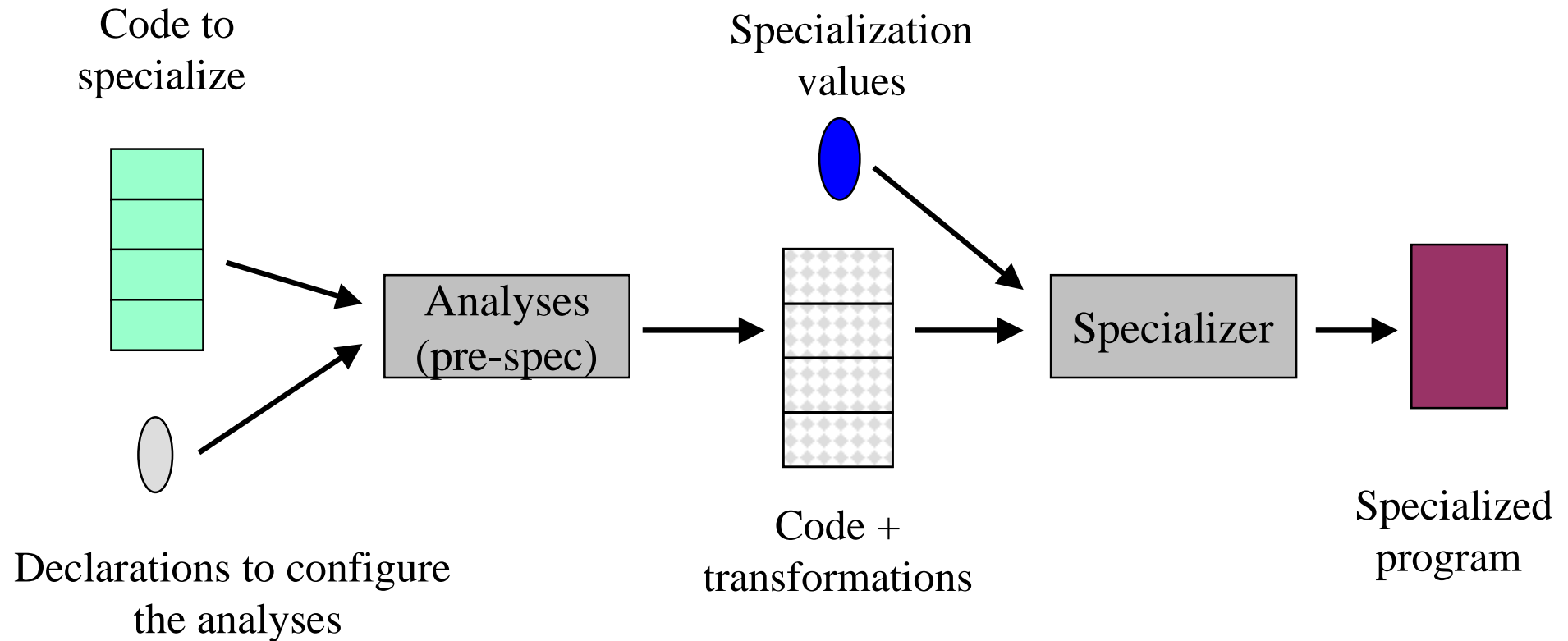
---

- Difference between the result and the developer's expectations
  - global specialization information
  - unconstrained analyses



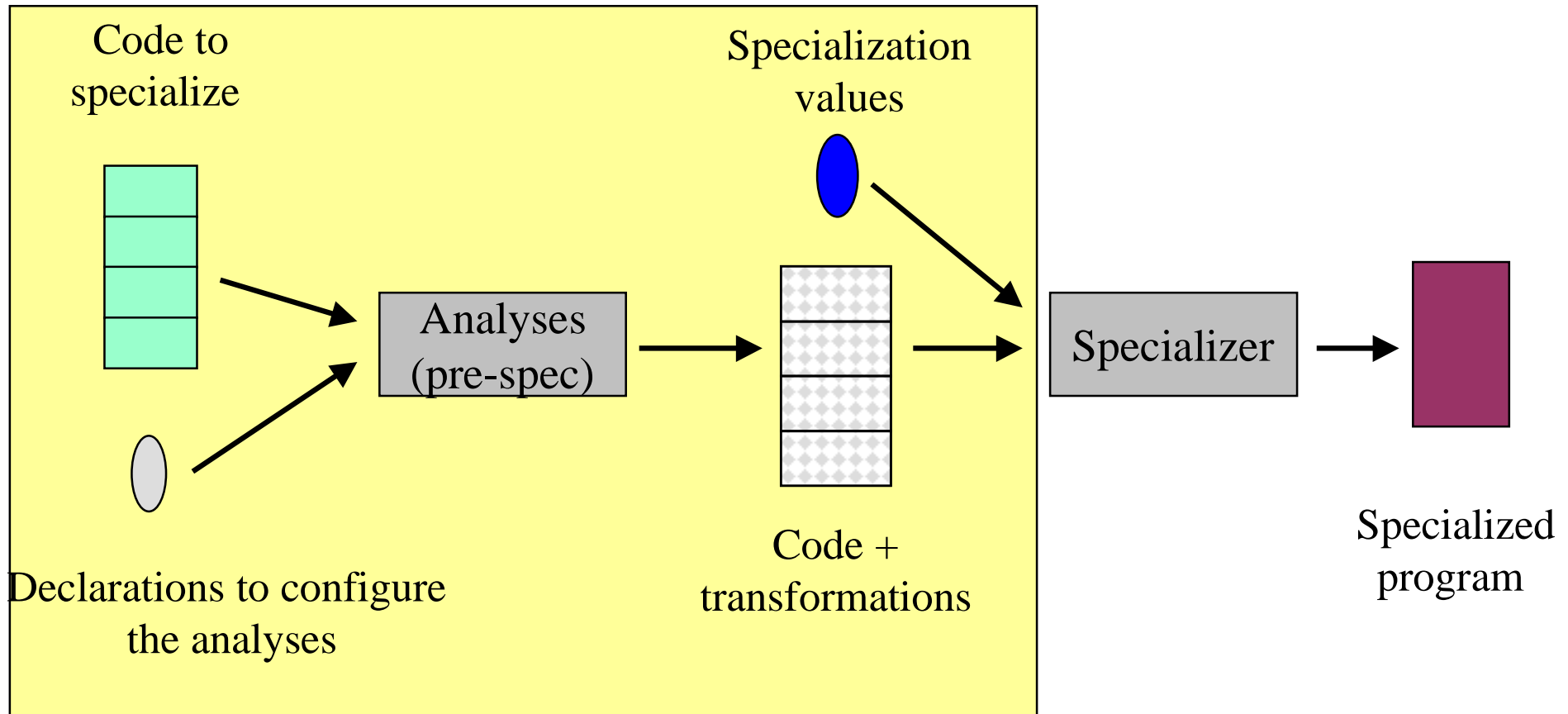
- Declarations express intentions: constraints
- Verification during analyses that constraints are respected

# Integrating our approach into an existing specializer



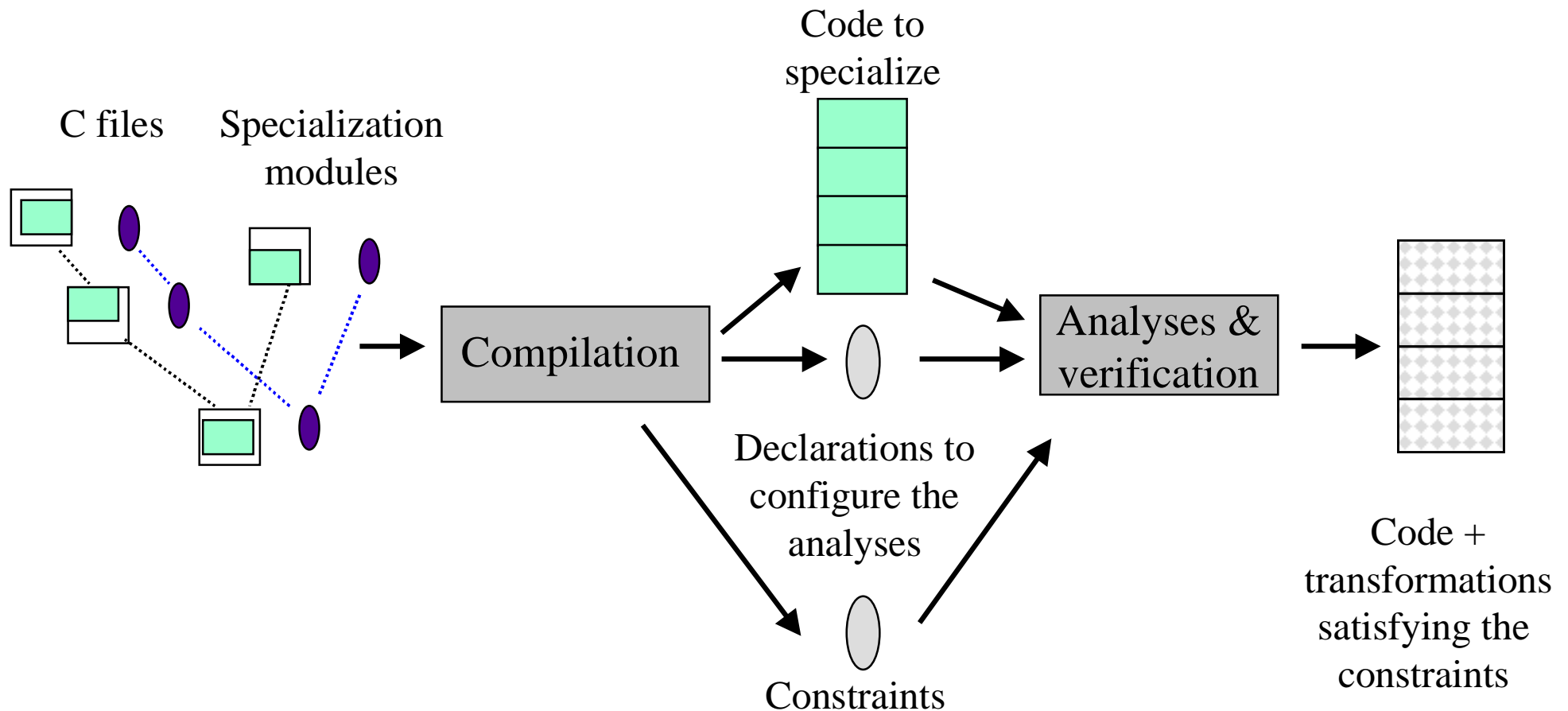
Compile-time specialization process

# Integrating our approach into an existing specializer

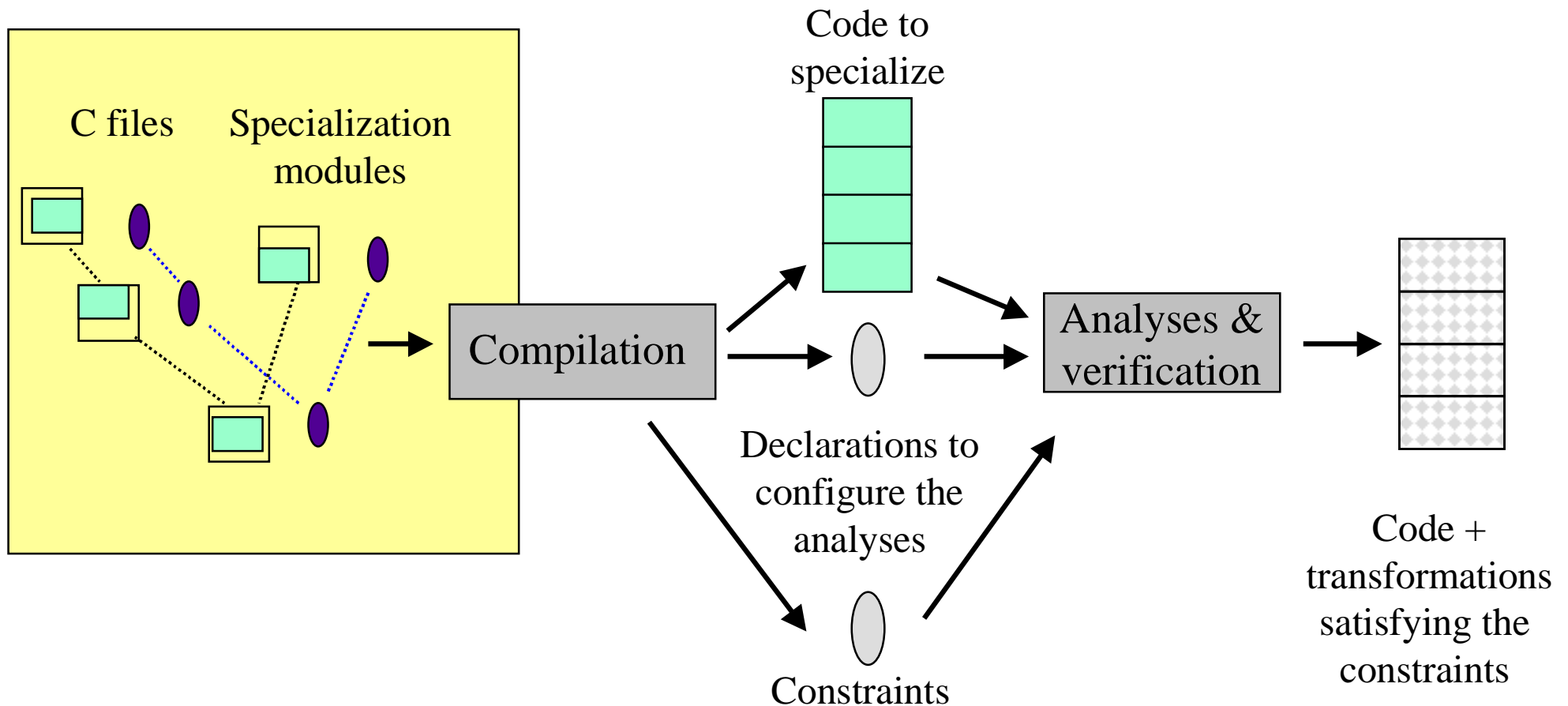


Compile-time specialization process

# Global view of our approach



# Global view of our approach



```

#include "encode.h"
#include "parity_bit.h"
#include "lin_b_code.h"
#include "lin_b_code_ysp.h"
#include "code_conv.h"
#include "encode_crc.h"
#include "libstring.h"

void
encode(struct data *donnees, struct code *codage, struct data *result){
  if (donnees->longueur != codage->K)
    { perror("donnees->longueur != codage->K");
      exit(1); }
  else if (strcmp(codage->type, "parity_bit") == 0)
    if (result->longueur != codage->K + 1)
      { perror("result->longueur != codage->K + 1");
        exit(1); }
    else parity_bit(codage->K, donnees->v, result->v);
  else if (strcmp(codage->type, "lin_b_code") == 0)
    if (!(*result).longueur)
      { perror ("result->longueur != codage->K");
        exit (1); }
    else if (strcmp(codage->opt, "ysp") == 0)
      lin_b_code_ysp(donnees->v, codage->K, codage->n, codage->matrix, result->v);
    else
      lin_b_code(donnees->v, codage->K, codage->n, codage->matrix, result->v);
  else if (strcmp(codage->type, "code_conv") == 0)
    if (!(*result).longueur)
      { perror ("result->longueur != (2 * codage->K)");
        exit (1); }
    else code_conv(donnees->v, codage->K, result->v);
  else if (strcmp(codage->type, "crc") == 0)
    if (!(*result).longueur)
      { perror ("result->longueur != codage->n");
        exit (1); }
    else encode_crc(donnees->v, codage->K, codage->n, codage->matrix[0], result->v);
  else
    { perror("unknown code type");
      exit(1); }
}

```

```

struct data
{
  int longueur;
  int *v;
};

struct code
{
  int k;
  int n;
  int **matrix;
  char *type;
  char *opt;
};

extern void encode(struct data *donnees, struct code *codage, struct data *result);

```

```
extern int strlen(char *s1, char *s2);
```

```
extern void encode_crc(int *donnees, int k, int n, int *poly, int *result);
```

```

#include "encode_crc.h"
void
encode_crc(int *donnees, int k, int n, int *poly, int *result)
{
  int i, j;
  i_poly = n - k + 1;
  for (i = 0; i < n; i++)
    result[i] = donnees[i];
  for (i = 0; i < k; i++)
    for (j = 0; j < i_poly; j++)
      result[i + j] ^= poly[j];
  for (i = 0; i < k; i++)
    result[i] = donnees[i];
}

```

```
extern void lin_b_code(int *vector, int k, int n, int **matrix, int *result);
```

```

#include "lin_b_code.h"
#include "mult_mat.h"
void
lin_b_code(int *vector, int k, int n, int **matrix, int *result)
{
  mult_mat(vector, k, n, matrix, result, 0);
}

```

```
extern void code_conv(int *donnees, int l, int *result);
```

```

#include "code_conv.h"
void
code_conv(int *donnees, int k, int *result)
{
  int temp;
  int i;
  for (i = 0; i < k - 2; i++)
    { temp = donnees[i] * donnees[i + 2];
      result[i + 1] = temp;
      temp *= donnees[i + 2];
      result[i + 2] = temp;
    }
  temp = donnees[k - 2] ^ donnees[k - 1];
  result[k - k] = temp;
  result[k - k + 1] = donnees[k - 2];
  result[k - k + 2] = donnees[k - 1];
  result[k - k + 3] = temp;
}

```

```
extern void mult_mat(int *vector, int k, int n, int **matrix, int *result, int indice);
```

```

#include "mult_mat.h"
void
mult_mat(int *vector, int k, int n, int **matrix, int *result, int indice)
{
  int temp;
  int i, j;
  for(i = 0; i < n; i++)
    { temp = 0;
      for(j = 0; j < k; j++)
        temp ^= vector[j] & matrix[j][i];
      result[i + indice] = temp;
    }
}

```

```
extern void lin_b_code_ysp(int *vector, int k, int n, int **matrix, int *result);
```

```

#include "lin_b_code_ysp.h"
#include "mult_mat.h"
void
lin_b_code_ysp(int *vector, int k, int n, int **matrix, int *result)
{
  int sub_matrix_size;
  int i;
  sub_matrix_size = n - k;
  mult_mat(vector, k, sub_matrix_size, matrix, result, k);
  for(i = 0; i < k; i++)
    result[i] = vector[i];
}

```

```
extern void parity_bit(int l, int *vector, int *result);
```

```

#include "parity_bit.h"
void
parity_bit(int l, int *vector, int *result)
{
  int i;
  int s;
  int l;
  int u;
  int v;
  for(i = 0; i < l; i++)
    { s = vector[i];
      result[i] = vector[i];
    }
  result[l] = s;
}

```

# Example: multmat

Think of an algorithm **and** of the desired specialization

**multmat.c**

**multmat.mdl**

# Example: multmat

```
int multmat (int *data, int k, int n, int **matrix, int * result, int ind)
```

**multmat.c**

**multmat.mdl**

# Example: multmat

```
int multmat (int *data, int k, int n, int **matrix, int * result, int ind)
```

**multmat.c**

**multmat.mdl**

# Example: multmat

```
int multmat (int *data, int k, int n, int **matrix, int * result, int ind)
```

**multmat.c**

```
multmat (int* data, int k, int n, int ** matrix, int * result, int ind);
```

**multmat.mdl**

# Example: multmat

```
int multmat (int *data, int k, int n, int **matrix, int * result, int ind)
```

S : known at specialization time

D : unknown at specialization time

multmat.c

```
multmat (D(int *) data, S(int) k, S(int) n,  
         S(int **) matrix, D(int *) result, D(int) ind);
```

multmat.mdl

# Example: multmat

```
int multmat (int *data, int k, int n, int **matrix, int * result, int ind) {  
    int tmp, i, j;  
    for(i = 0; i < n; i++)  
        tmp = 0;  
        for(j = 0; j < k; j++)  
            tmp = tmp ^ (data[j] & matrix[j][i] );  
        result[ind + i] = tmp;  
}
```

**multmat.c**

```
multmat (D(int *) data, S(int) k, S(int) n,  
        S(int **) matrix, D(int *) result, D(int) ind);
```

**multmat.mdl**

# Example: multmat

```
int multmat (int *data, int k, int n, int **matrix, int * result, int ind) {  
    int tmp, i, j;  
    for(i = 0; i < n; i++)  
        tmp = 0;  
        for(j = 0; j < k; j++)  
            tmp = tmp ^ (data[j] & matrix[j][i] );  
        result[ind + i] = tmp;  
}
```

**multmat.c**

```
Btmultmat :: multmat (D(int *) data, S(int) k, S(int) n,  
                    S(int **) matrix, D(int *) result, D(int) ind);
```

**multmat.mdl**

# Example: multmat

```
int multmat (int *data, int k, int n, int **matrix, int * result, int ind) {  
    int tmp, i, j;  
    for(i = 0; i < n; i++)  
        tmp = 0;  
        for(j = 0; j < k; j++)  
            tmp = tmp ^ (data[j] & matrix[j][i] );  
        result[ind + i] = tmp;  
}
```

multmat.c

From multmat.c {

```
Btmultmat :: intern multmat (D(int *) data, S(int) k, S(int) n,  
                             S(int **) matrix, D(int *) result, D(int) ind);  
}
```

multmat.mdl

# Example: multmat

```
int multmat (int *data, int k, int n, int **matrix, int * result, int ind) {
    int tmp, i, j;
    for(i = 0; i < n; i++)
        tmp = 0;
        for(j = 0; j < k; j++)
            tmp = tmp ^ (data[j] & matrix[j][i] );
        result[ind + i] = tmp;
}
```

multmat.c

```
Module mult_mat {
    Defines {
        From multmat.c {
            Btmultmat :: intern multmat (D(int *) data, S(int) k, S(int) n,
                S(int **) matrix, D(int *) result, D(int) ind);
        }}
    Exports { Btmultmat; }
}
```

multmat.mdl

```

#include <unistd.h>
#include <parity_bit.h>
#include <lin_b_code.h>
#include <lin_b_code_sys.h>
#include <code_conv.h>
#include <encode_crc.h>
#include <libstring.h>

void
encode(struct data *donnees, struct code *codage, struct data *result){
  if (donnees->longueur != codage->K)
    { perror("donnees->longueur != codage->K");
      exit(1); }
  else if (sizeof(codage->type, "parity_bit") == 0)
    { if (result->longueur != codage->k + 1)
        { perror("result->longueur != codage->k + 1");
          exit(1); }
      else parity_bit(codage->k, donnees->v, result->v);
    }
  else if (sizeof(codage->type, "lin_b_code") == 0)
    { if (!(*result).longueur != ("codage->n"))
        { perror ("result->longueur != codage->n");
          exit (1); }
      else if (sizeof(codage->type, "type") == 0)
        { if (sizeof(codage->type, "type") == 0)
            { perror ("result->longueur != codage->n");
              exit (1); }
          else if (sizeof(codage->type, "type") == 0)
            { perror ("unknown code type");
              exit(1); }
        }
    }
}

```

```

struct data
{
  int longueur;
  int *v;
};

struct code
{
  int k;
  int n;
  int **matrix;
  char *type;
  char *opt;
};

extern void encode(struct data *donnees, struct code *codage, struct data *result);

```

```
extern int strlen(char *s1, char *s2);
```

```
extern void encode_crc(int *donnees, int k, int n, int *poly, int *result);
```

```

#include <encode_crc.h>
void
encode_crc(int *donnees, int k, int n, int *poly, int *result)
{
  int i, j;
  i_poly = n - k + 1;
  for (i = 0; i < n; i++)
    result[i] = donnees[i];
  for (i = 0; i < k; i++)
    for (j = 0; j < i_poly; j++)
      result[i + j] ^= poly[j];
  for (i = 0; i < k; i++)
    result[i] = donnees[i];
}

```

```
extern void lin_b_code(int *vector, int k, int n, int **matrix, int *result);
```

```

#include <lin_b_code.h>
#include <mult_mat.h>
void
lin_b_code(int *vector, int k, int n, int **matrix, int *result)
{
  mult_mat(vector, k, n, matrix, result, 0);
}

```

```
extern void code_conv(int *donnees, int l, int *result);
```

```

#include <code_conv.h>
void
code_conv(int *donnees, int k, int *result)
{
  int temp;
  int i;
  for (i = 0; i < k - 2; i++)
    { temp = donnees[i] * donnees[i + 2];
      result[i * 2 + 1] = temp;
      temp *= donnees[i + 2];
      result[i * 2 + 1] = temp;
    }
  temp = donnees[k - 2] * donnees[k - 1];
  result[i * 2 + 1] = temp;
  result[i * 2 + 1] = temp;
}

```

```
extern void mult_mat(int *vector, int k, int n, int **matrix, int *result, int indice);
```

```

#include <mult_mat.h>
void
mult_mat(int *vector, int k, int n, int **matrix, int *result, int indice)
{
  int temp;
  int i, j;
  for (i = 0; i < n; i++)
    { temp = 0;
      for (j = 0; j < k; j++)
        temp ^= vector[j] & matrix[j][i];
      result[i + indice] = temp;
    }
}

```

```

#include <lin_b_code_sys.h>
#include <mult_mat.h>
void
lin_b_code_sys(int *vector, int k, int n, int **matrix, int *result)
{
  int sub_matrix_size;
  int i;
  sub_matrix_size = n - k;
  mult_mat(vector, k, sub_matrix_size, matrix, result, k);
  for (i = 0; i < k; i++)
    result[i] = vector[i];
}

```

```
extern void parity_bit(int l, int *vector, int *result);
```

```

#include <parity_bit.h>
void
parity_bit(int l, int *vector, int *result)
{
  int res;
  int i;
  res = 0;
  for (i = 0; i < l; i++)
    { res ^= vector[i];
      result[i] = vector[i];
    }
  result[l] = res;
}

```

# Module composition

```
#include "multmat.h"
int systLBC (int *data, int k, int n, int **matrix, int * result) {
    int i;
    for(i = 0; i < k; i++)
        result[i] = data[i];
        multmat(data, k, n - k, matrix, result, k);
}
```

**systLBC.c**

```
int multmat (int *data, int k, int n, int **matrix, int * result, int ind) {
    int tmp, i, j;
    for(i = 0; i < n; i++)
        tmp = 0;
        for(j = 0; j < k; j++)
            tmp = tmp ^ (data[j] & matrix[j][i] );
        result[ind + i] = tmp;
}
```

**multmat.c**

# Module composition

```
#include "multmat.h"
int systLBC (int *data, int k, int n, int **matrix, int * result) {
    int i;
    for(i = 0; i < k; i++)
        result[i] = data[i];
        multmat(data, k, n - k, matrix, result, k);
}
```

**systLBC.c**

```
Module mult_mat {
    Defines {
        From multmat.c {
            Btmultmat :: intern multmat (D(int *) data, S(int) k, S(int) n,
                S(int **) matrix, D(int *) result, D(int) ind);
        }
    }
    Exports { Btmultmat; }
```

**multmat.mdl**

# Module composition

```
BtsystLBC :: intern systLBC (D(int *) data, S(int) k, S(int) n,  
                             S(int **) matrix, D(int *) result)
```

**systLBC.mdl**

```
Module mult_mat {  
  Defines {  
    From multmat.c {  
      Btmultmat :: intern multmat (D(int *) data, S(int) k, S(int) n,  
                                   S(int **) matrix, D(int *) result, D(int) ind);  
    }  
  }  
  Exports { Btmultmat; }
```

**multmat.mdl**

# Module composition

```
BtsystLBC :: intern systLBC (D(int *) data, S(int) k, S(int) n,  
                             S(int **) matrix, D(int *) result)  
           needs { Btmultmat; };
```

**systLBC.mdl**

```
Module mult_mat {  
  Defines {  
    From multmat.c {  
      Btmultmat :: intern multmat (D(int *) data, S(int) k, S(int) n,  
                                   S(int **) matrix, D(int *) result, D(int) ind);  
    }  
  }  
  Exports { Btmultmat; } }
```

**multmat.mdl**

# Module composition

```
Imports { From multmat.mdl { Btmultmat; } }
```

```
BtsystLBC :: intern systLBC (D(int *) data, S(int) k, S(int) n,  
                             S(int **) matrix, D(int *) result)  
needs { Btmultmat; };
```

**systLBC.mdl**

```
Module mult_mat {
```

```
  Defines {
```

```
    From multmat.c {
```

```
      Btmultmat :: intern multmat (D(int *) data, S(int) k, S(int) n,  
                                   S(int **) matrix, D(int *) result, D(int) ind);
```

```
    } }
```

```
  Exports { Btmultmat; } }
```

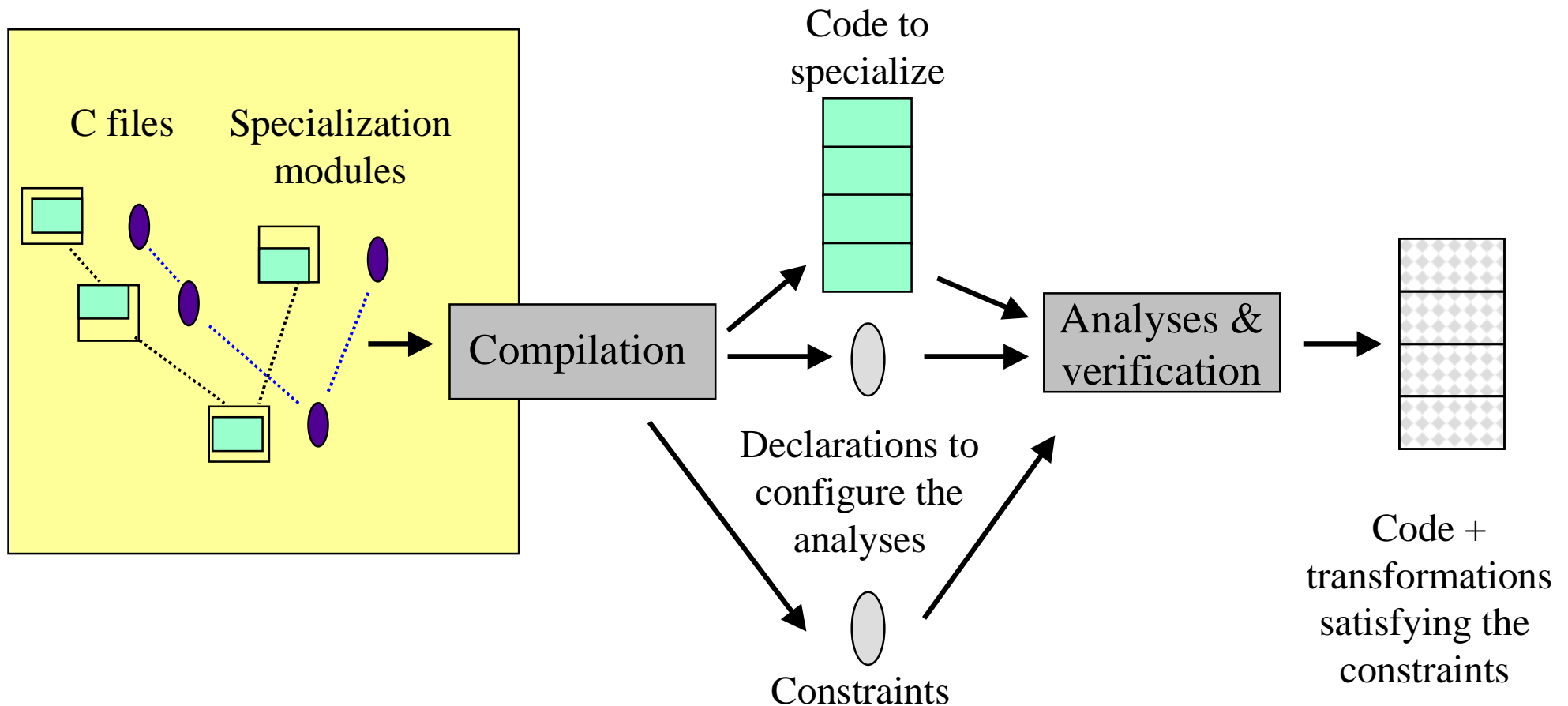
**multmat.mdl**

# Module composition

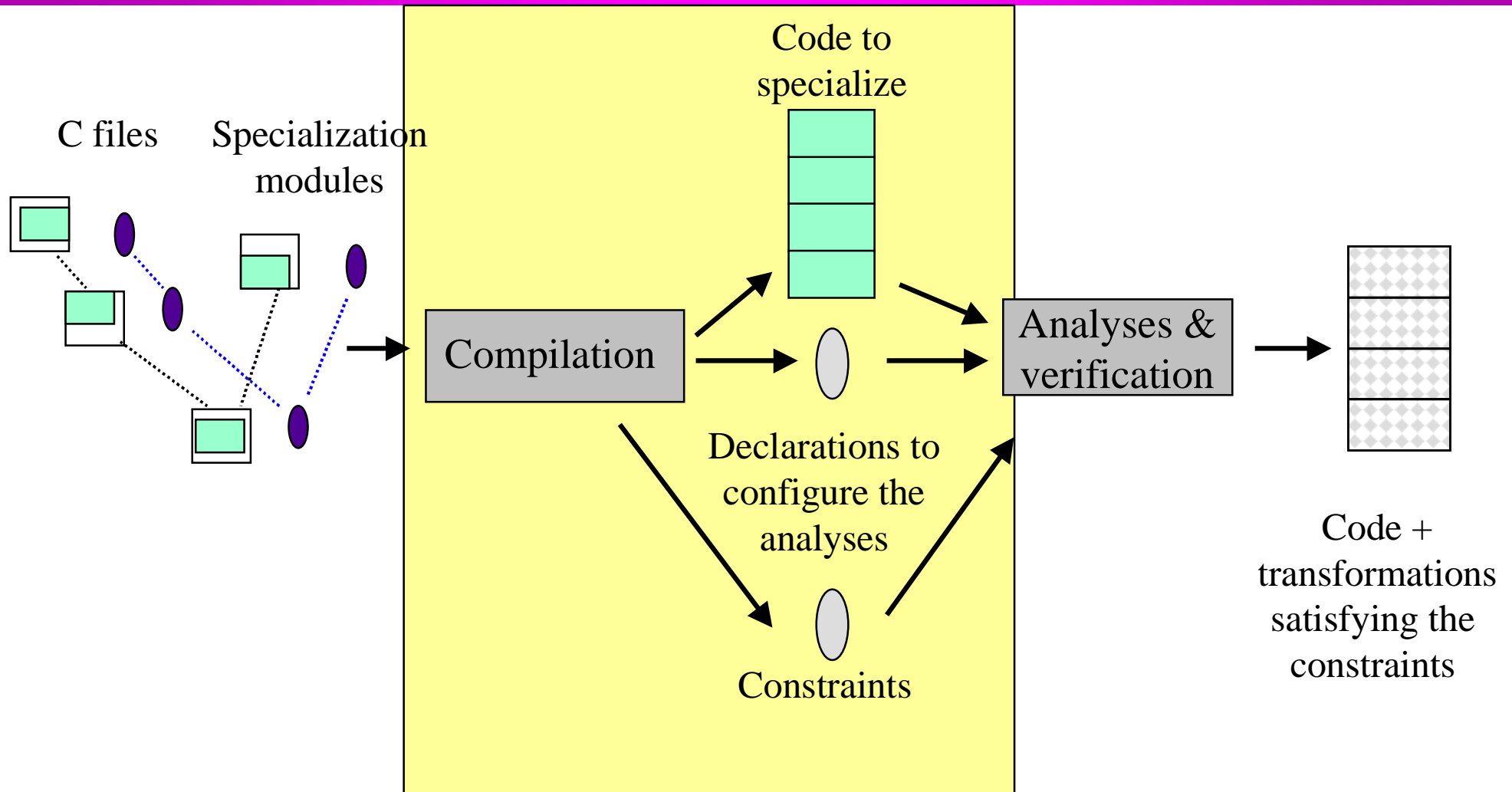
```
Module syst_LBC {  
  Imports { From multmat.mdl { Btmultmat; } }  
  Defines {  
    From systLBC.c {  
      BtsystLBC :: intern systLBC (D(int *) data, S(int) k, S(int) n,  
                                   S(int **) matrix, D(int *) result)  
      needs { Btmultmat; }; } }  
  Exports { BtsystLBC; } }  
systLBC.mdl
```

```
Module mult_mat {  
  Defines {  
    From multmat.c {  
      Btmultmat :: intern multmat (D(int *) data, S(int) k, S(int) n,  
                                   S(int **) matrix, D(int *) result, D(int) ind);  
    } }  
  Exports { Btmultmat; } }  
multmat.mdl
```

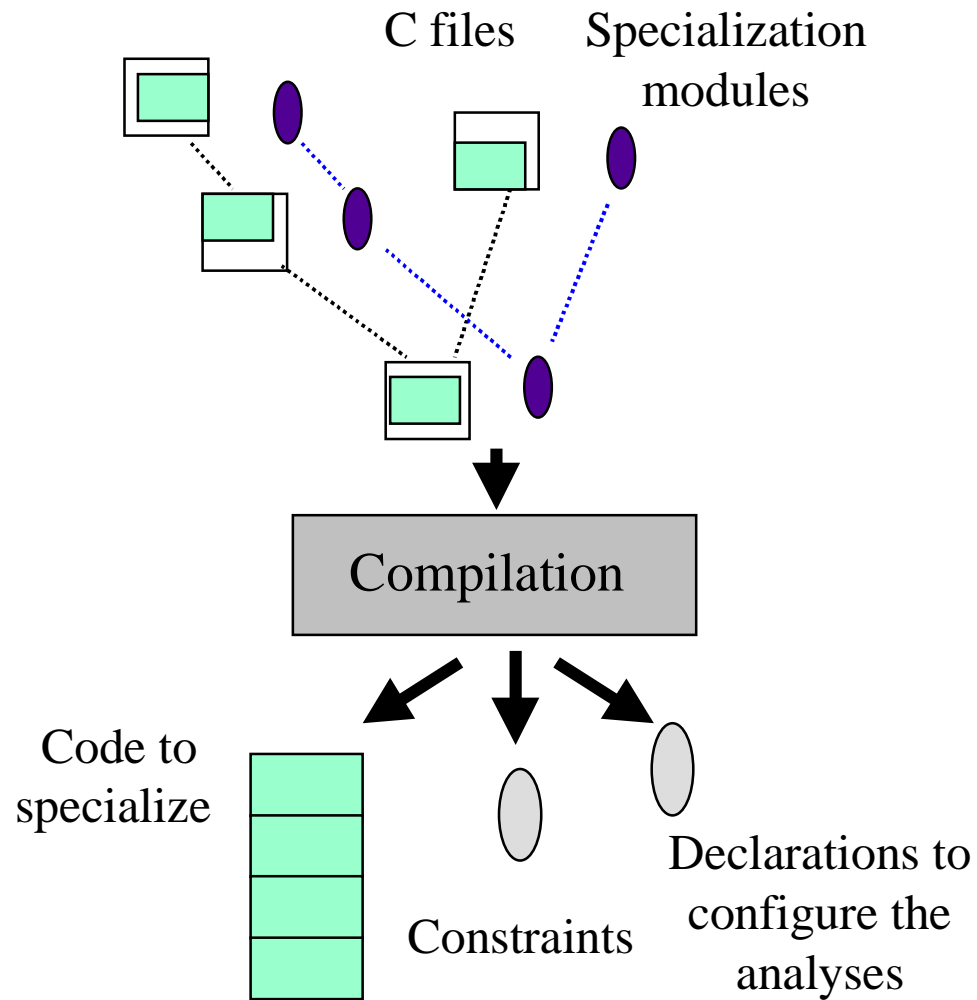
# Global view of our approach



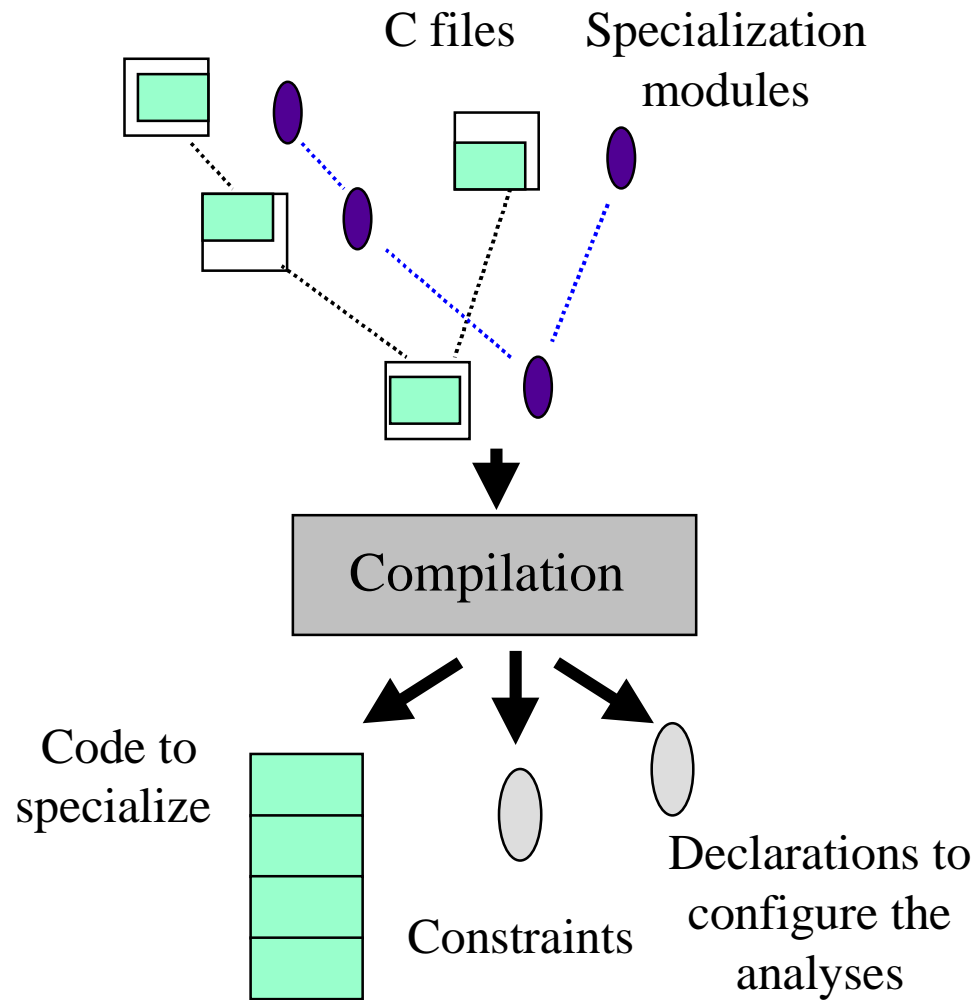
# Global view of our approach



# Compilation

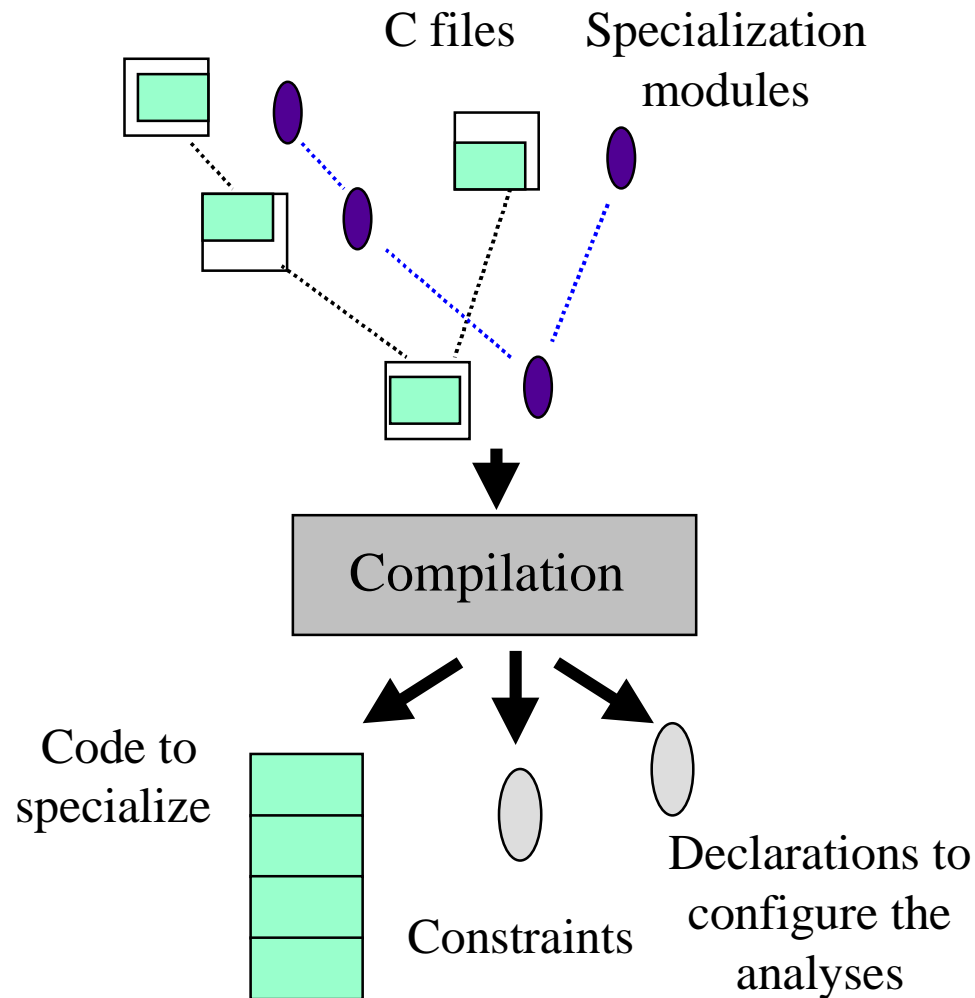


# Compilation



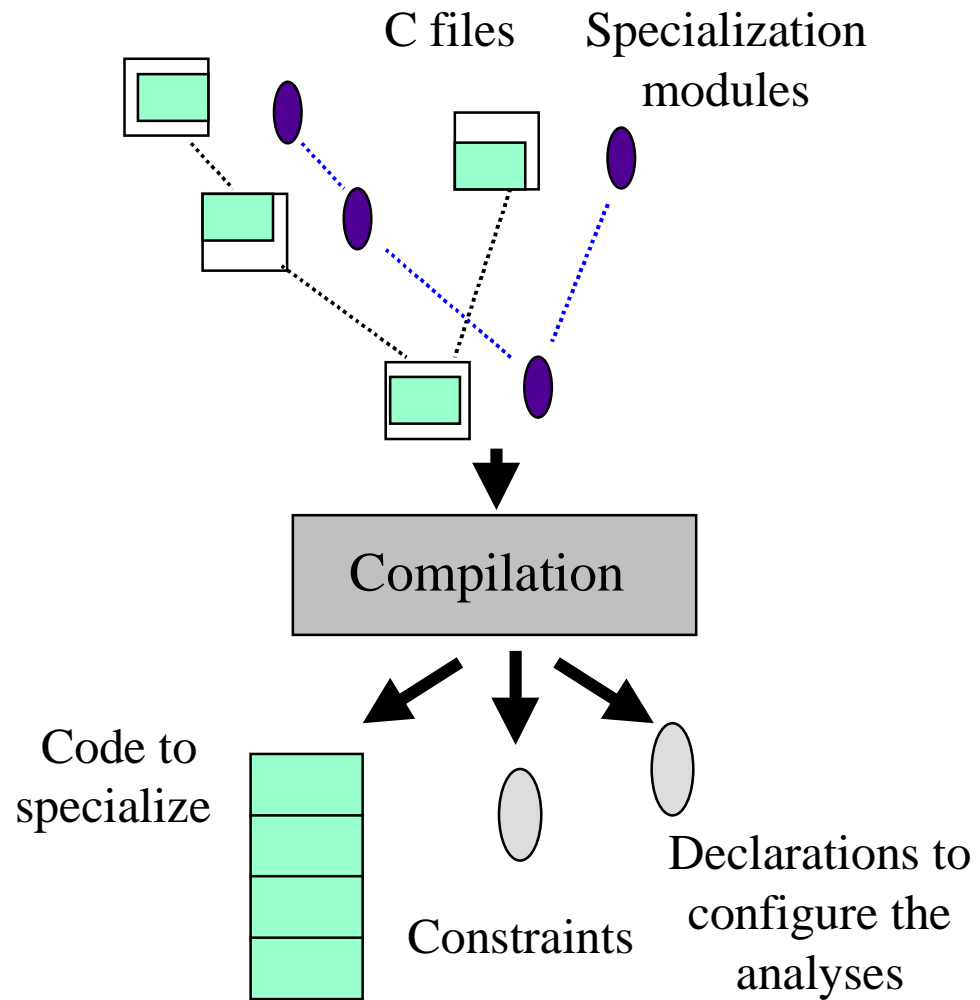
– Scenarios dependency graph

# Compilation



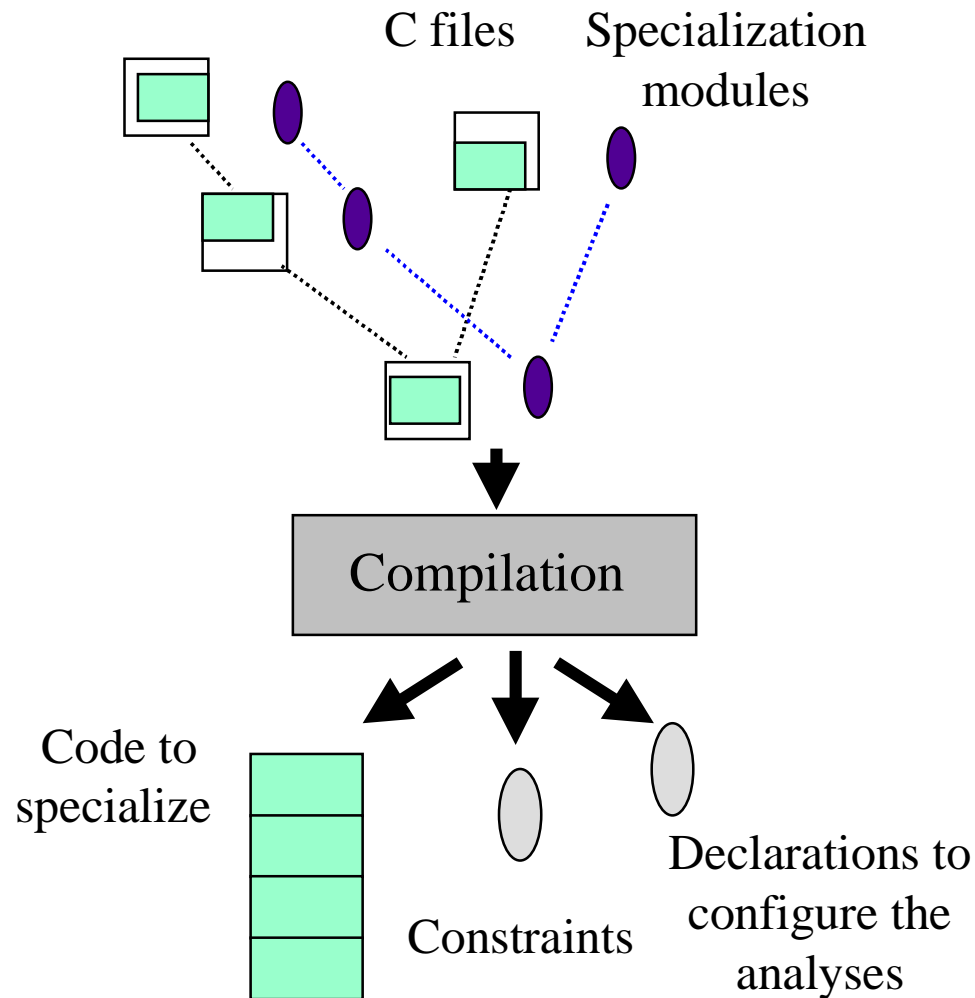
- Scenarios dependency graph
- Check of the declarations' semantics

# Compilation



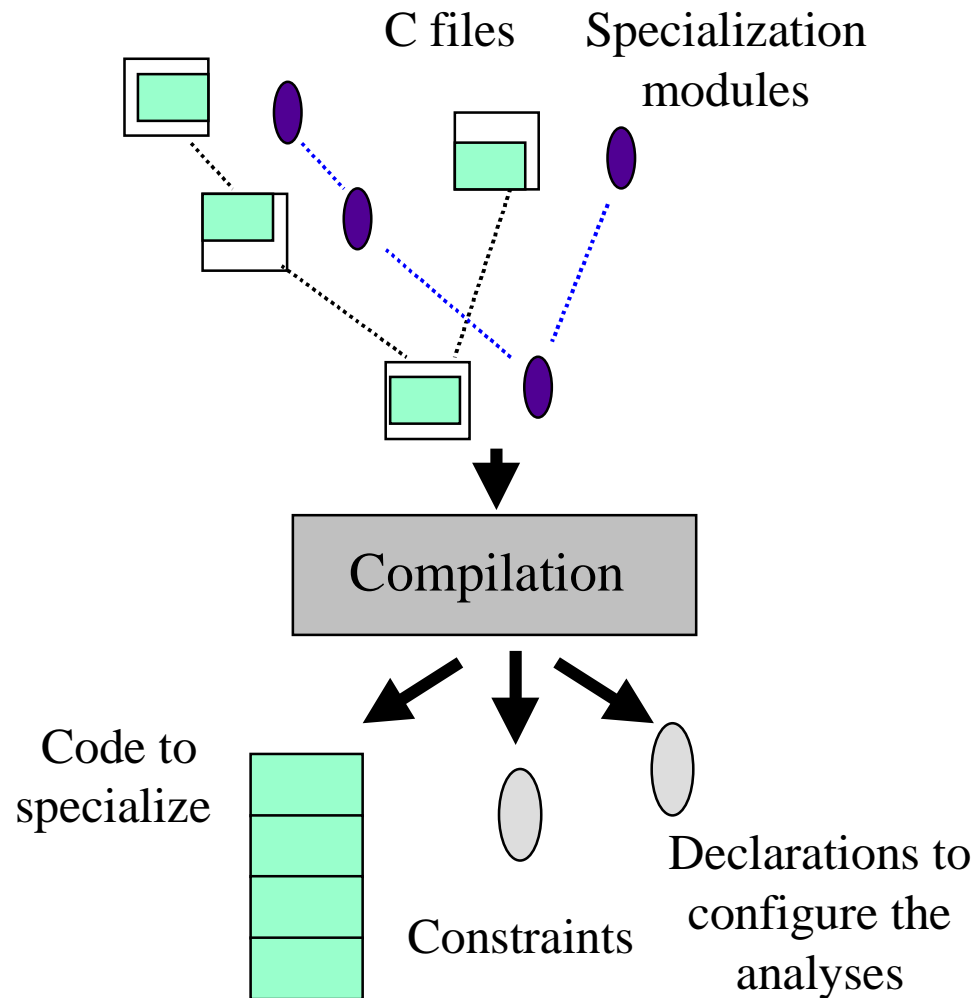
- Scenarios dependency graph
- Check of the declarations' semantics
- Extraction of the code to specialize

# Compilation



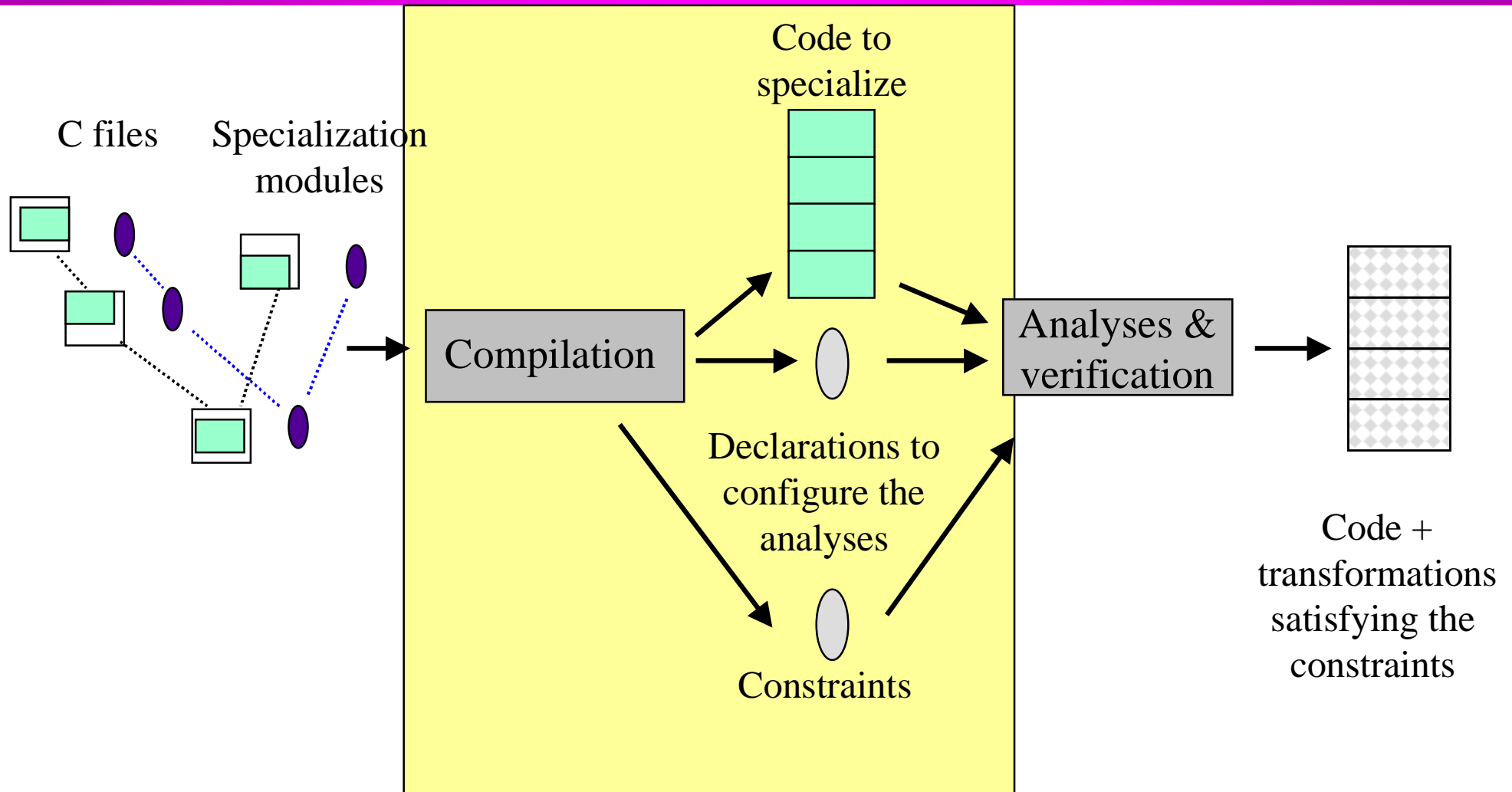
- Scenarios dependency graph
- Check of the declarations' semantics
- Extraction of the code to specialize
- Generation of declarations to configure the analyses

# Compilation

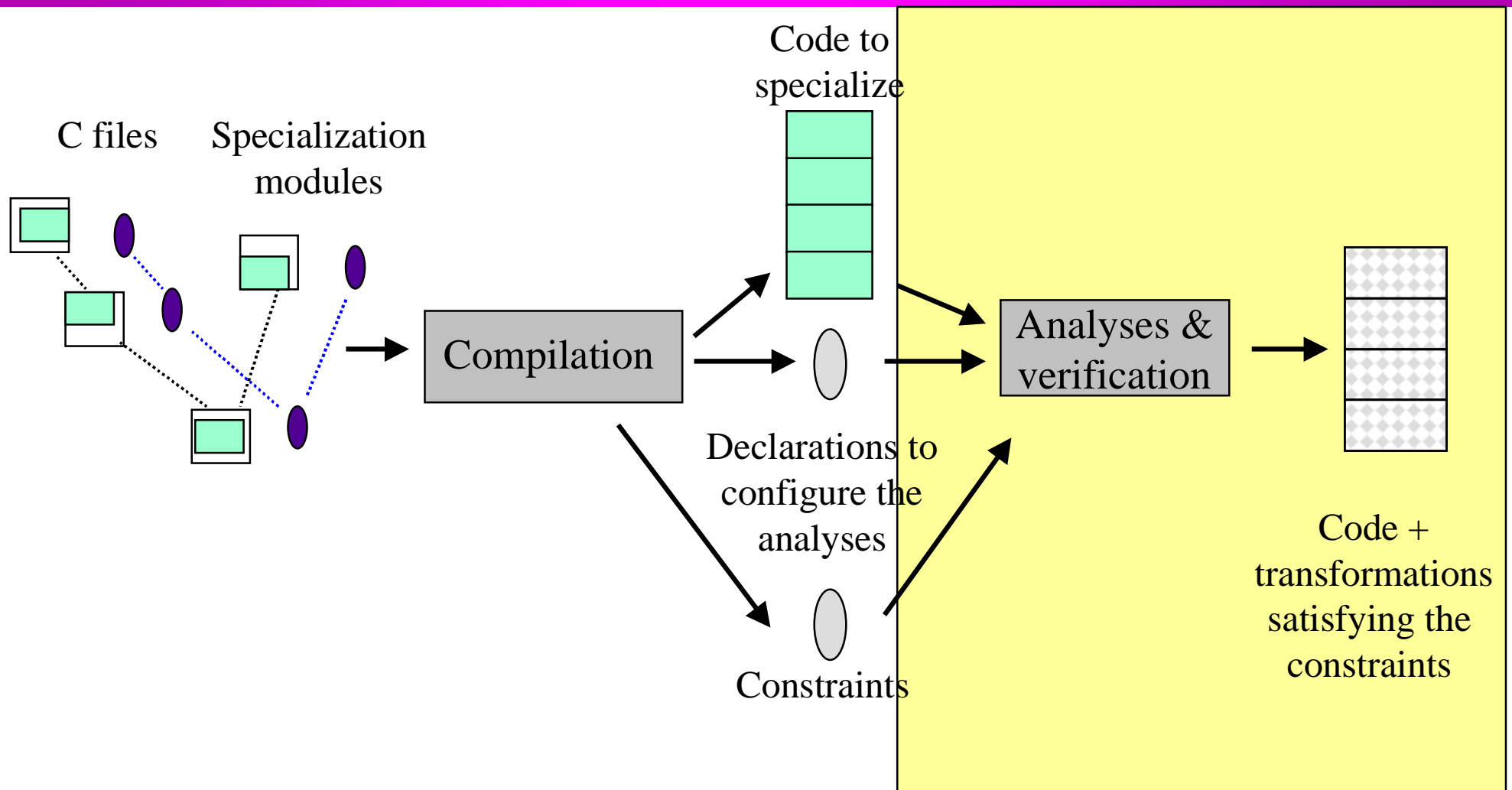


- Scenarios dependency graph
- Check of the declarations' semantics
- Extraction of the code to specialize
- Generation of declarations to configure the analyses
- Extraction of the constraints to satisfy

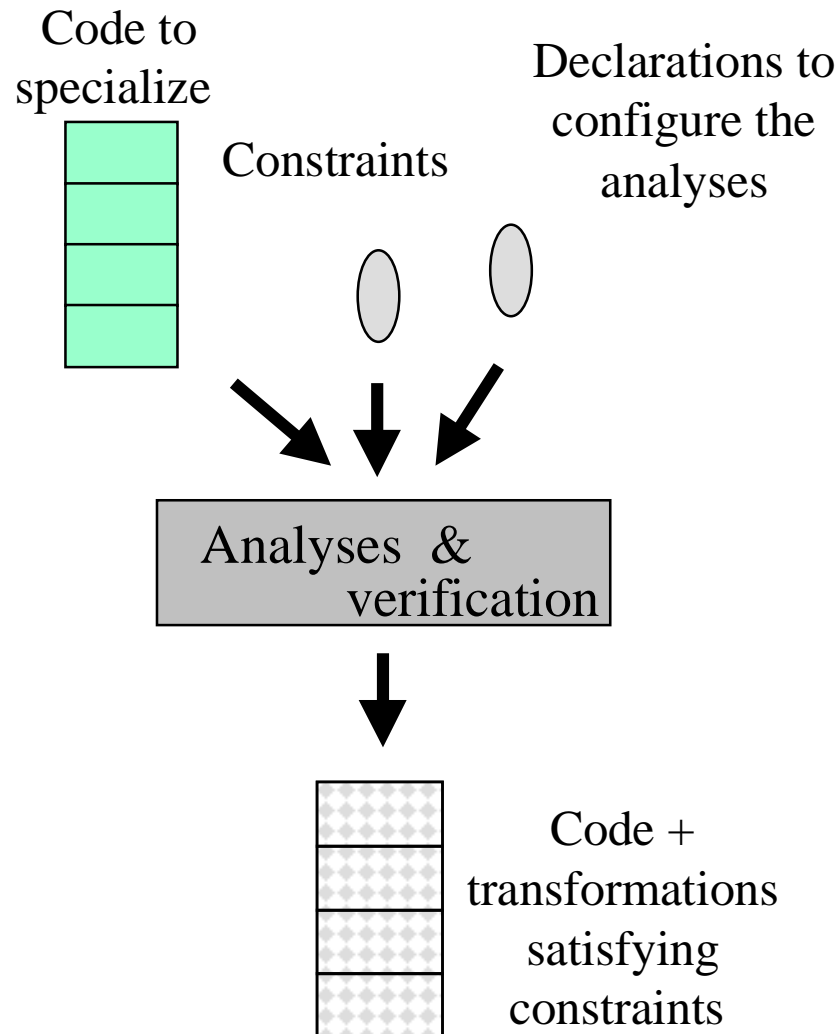
# Global view of our approach



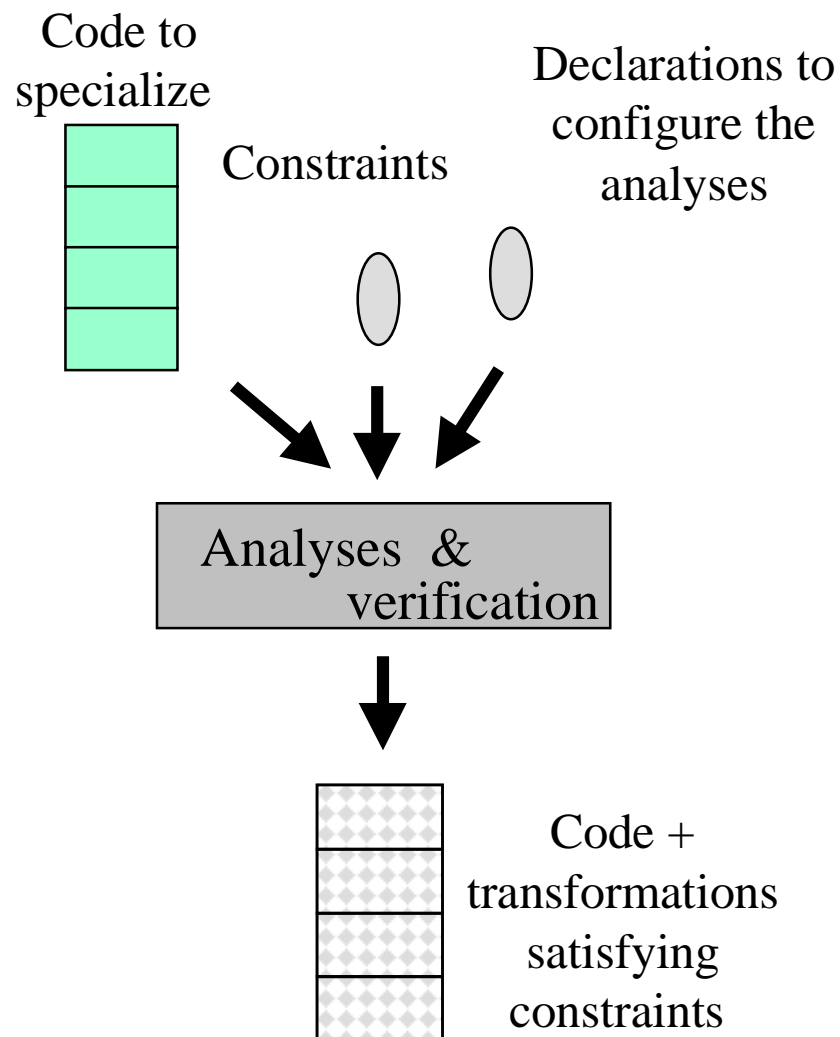
# Global view of our approach



# Analyses and verification

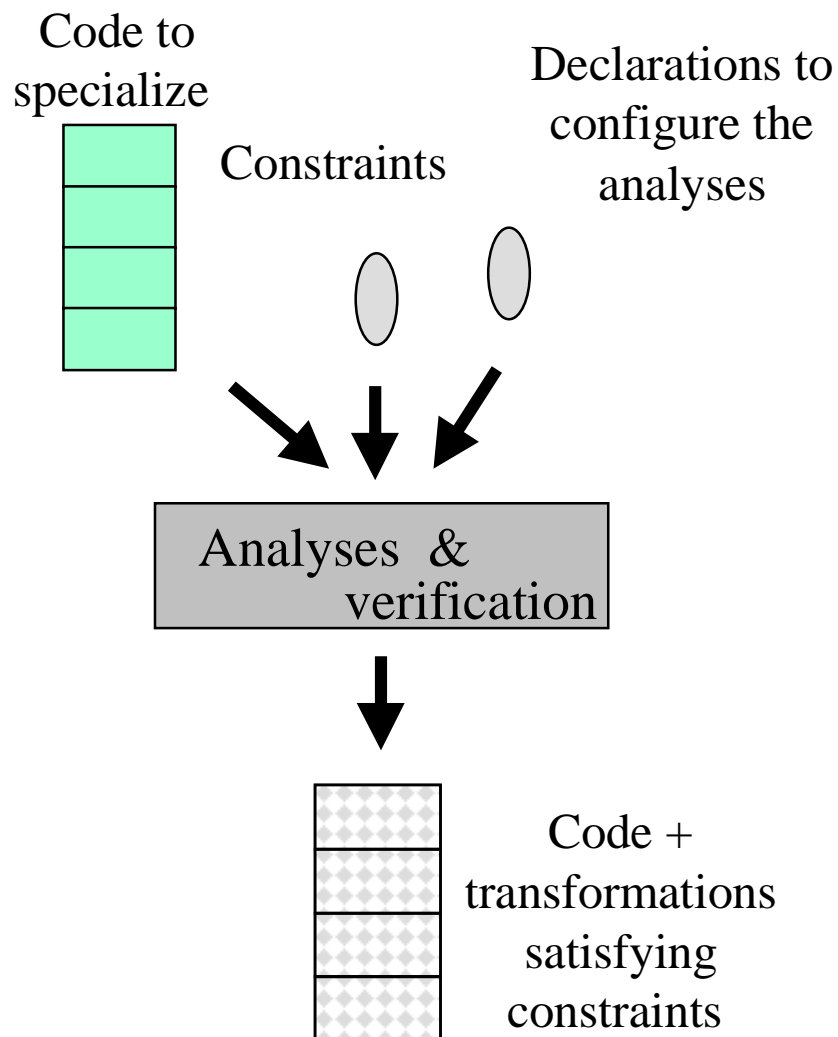


# Analyses and verification



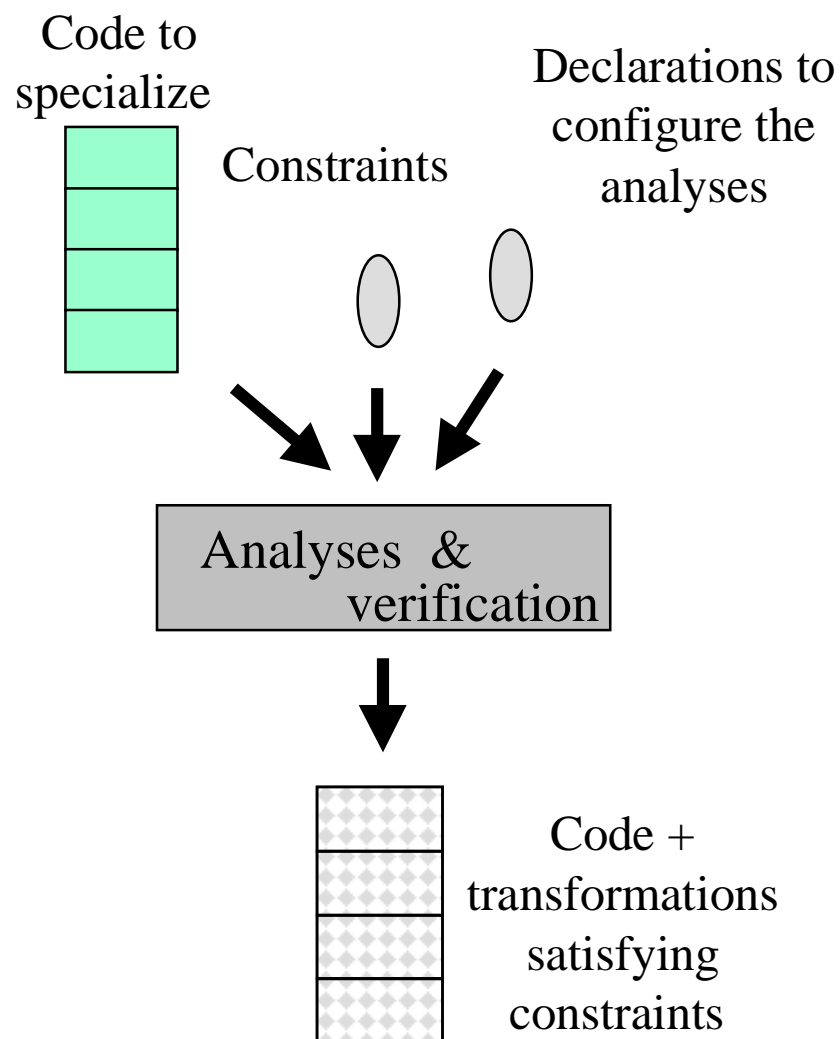
- Verification that specialization properties determined by the analysis satisfy the constraints

# Analyses and verification



- Verification that specialization properties determined by the analysis satisfy the constraints
- If incoherence then raise error
  - similar to type checking

# Analyses and verification



- Verification that specialization properties determined by the analysis satisfy the constraints
- If incoherence then raise error
  - similar to type checking
- Otherwise, calculated transformations will yield the desired specialization

# Verification example

---

```
int systLBC (intD *D data, intS k, intS n, intS *S *S matrix, intD *D result) {  
  ...  
  multmat(data, k, n - k, matrix, result, k);  
}
```

```
int multmat (intD *D data, intS k, intS n,  
             intS *S *S matrix, intD *D result, intD ind) {  
  ...  
}
```

$S$  and  $D$  : developer's constraints

# Verification example

---

```
int systLBC (intD *D data, intS k, intS n, intS *S *S matrix, intD *D result) {  
  ...  
  multmat(data, k, n - k, matrix, result, k);  
}
```

```
int multmat (intD *D data, intS k, intS n,  
             intS *S *S matrix, intD *D result, intD ind) {  
  ...  
}
```

# Verification example

---

```
int systLBC (intD *D data, intS k, intS n, intS *S *S matrix, intD *D result) {  
  ...  
  multmat(data, k, n - k, matrix, result, k);  
}
```

```
int multmat (intD *D data, intS k, intS n,  
             intS *S *S matrix, intD *D result, intD ind) {  
  ...  
}
```

# Verification example

```
int systLBC (intD *D data, intS k, intS n, intS *S *S matrix, intD *D result) {  
  ...  
  multmat(data, k, n - k, matrix, result, k);  
}
```

```
int *  
intD *D
```

```
int multmat (intD *D data, intS k, intS n,  
             intS *S *S matrix, intD *D result, intD ind) {  
  ...  
}
```

# Verification example

---

```
int systLBC (intD *D data, intS k, intS n, intS *S *S matrix, intD *D result) {  
  ...  
  multmat(data, k, n - k, matrix, result, k);  
}
```

int  
int<sup>S</sup>

```
int multmat (intD *D data, intS k, intS n,  
             intS *S *S matrix, intD *D result, intD ind) {  
  ...  
}
```

# Verification example

```
int systLBC (intD *D data, intS k, intS n, intS *S *S matrix, intD *D result) {  
  ...  
  multmat(data, k, n - k, matrix, result, k);  
}
```

int  
int<sup>D</sup>

```
int multmat (intD *D data, intS k, intS n,  
             intS *S *S matrix, intD *D result, intD ind) {  
  ...  
}
```

# Verification example

```
int systLBC (intD *D data, intS k, intS n, intS *S *S matrix, intD *D result) {  
  ...  
  multmat(data, k, n - k, matrix, result, k);  
}
```

int  
int<sup>D</sup>

```
int multmat (intD *D data, intS k, intS n,  
             intS *S *S matrix, intD *D result, intD ind) {  
  ...  
}
```

# Verification example

---

```
int systLBC (intD *D data, intS k, intS n, intS *S *S matrix, intD *D result) {  
  ...  
  multmat(data, k, n - k, matrix, result, k);  
}
```

**OK**

```
int multmat (intD *D data, intS k, intS n,  
             intS *S *S matrix, intD *D result, intD ind) {  
  ...  
}
```

# Declarative approach: summary

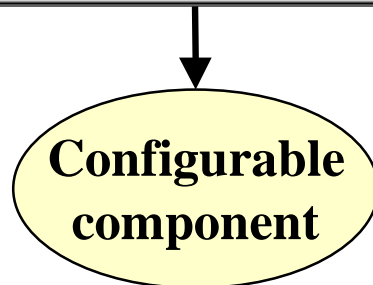
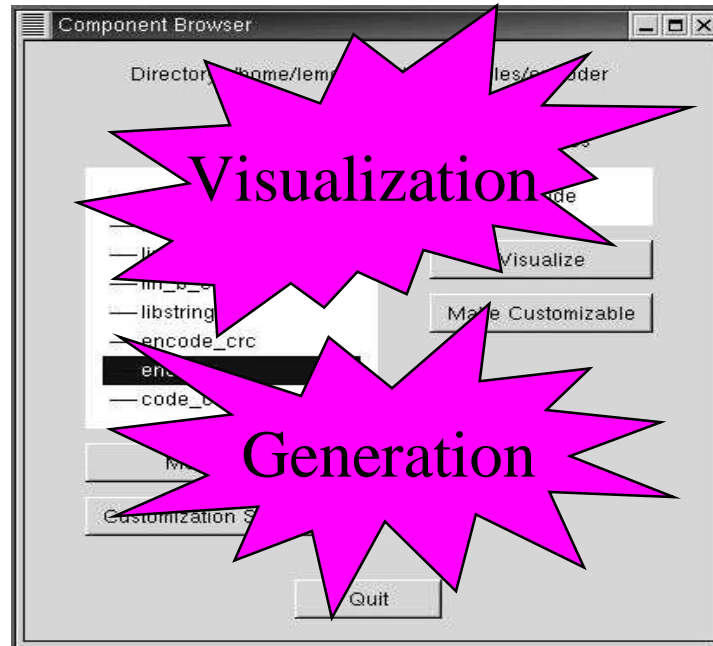
---

---

- Specialization modules
  - developer's intention
  - programming framework
- Compilation + verification
  - configuration of the specializer
  - specialization predictable with respect to the developer's declarations
- Systematic use of specialization now possible

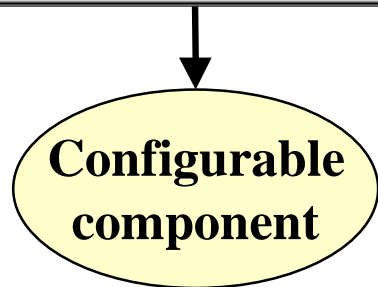
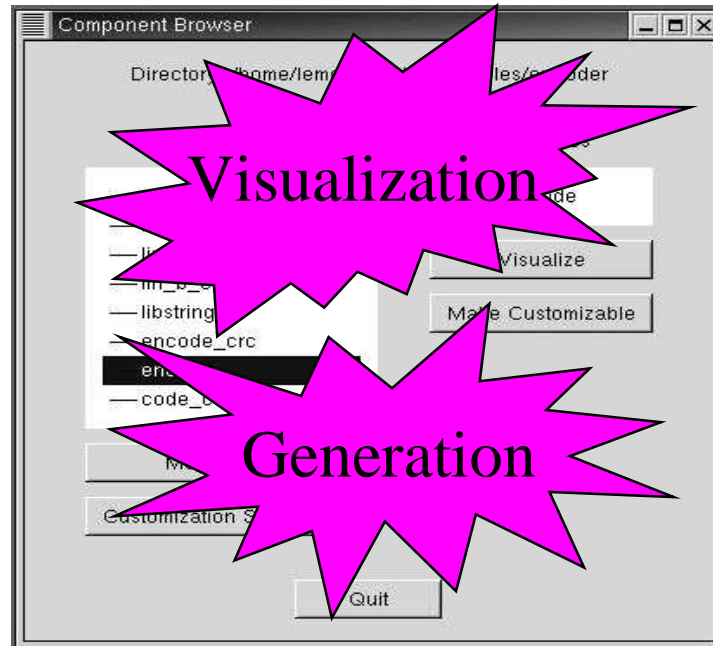
# Environment to build configurable components

## Developer interface

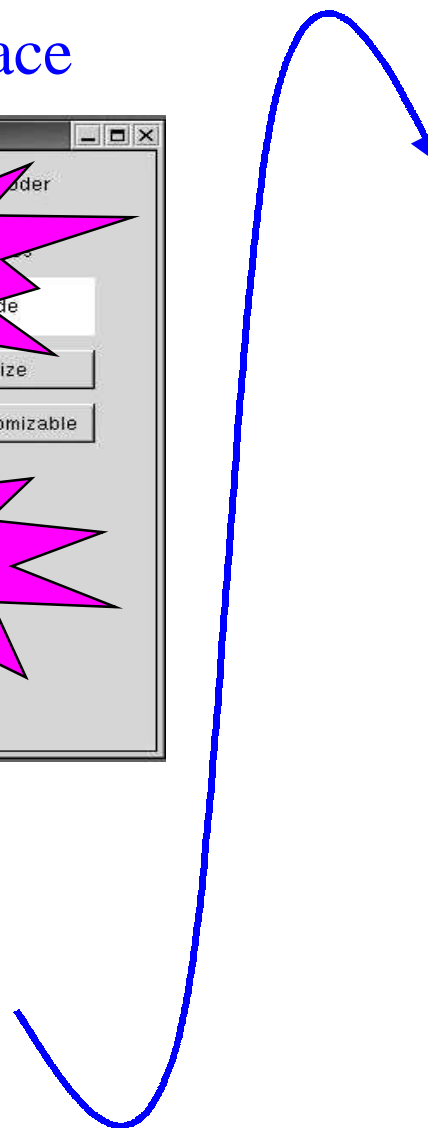
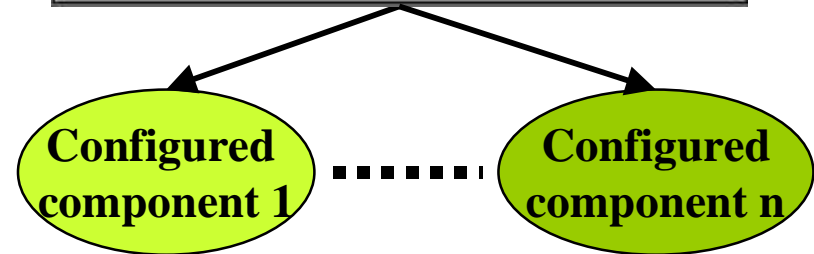
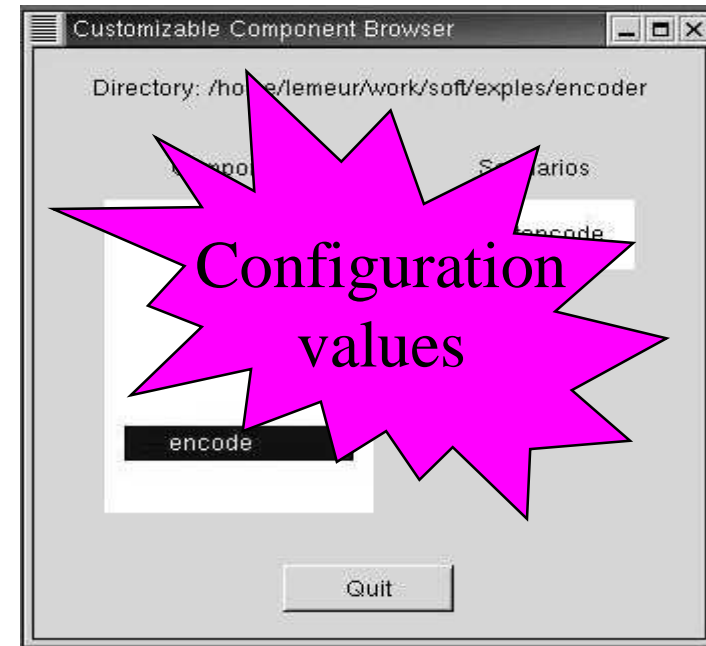


# Environment to build configurable components

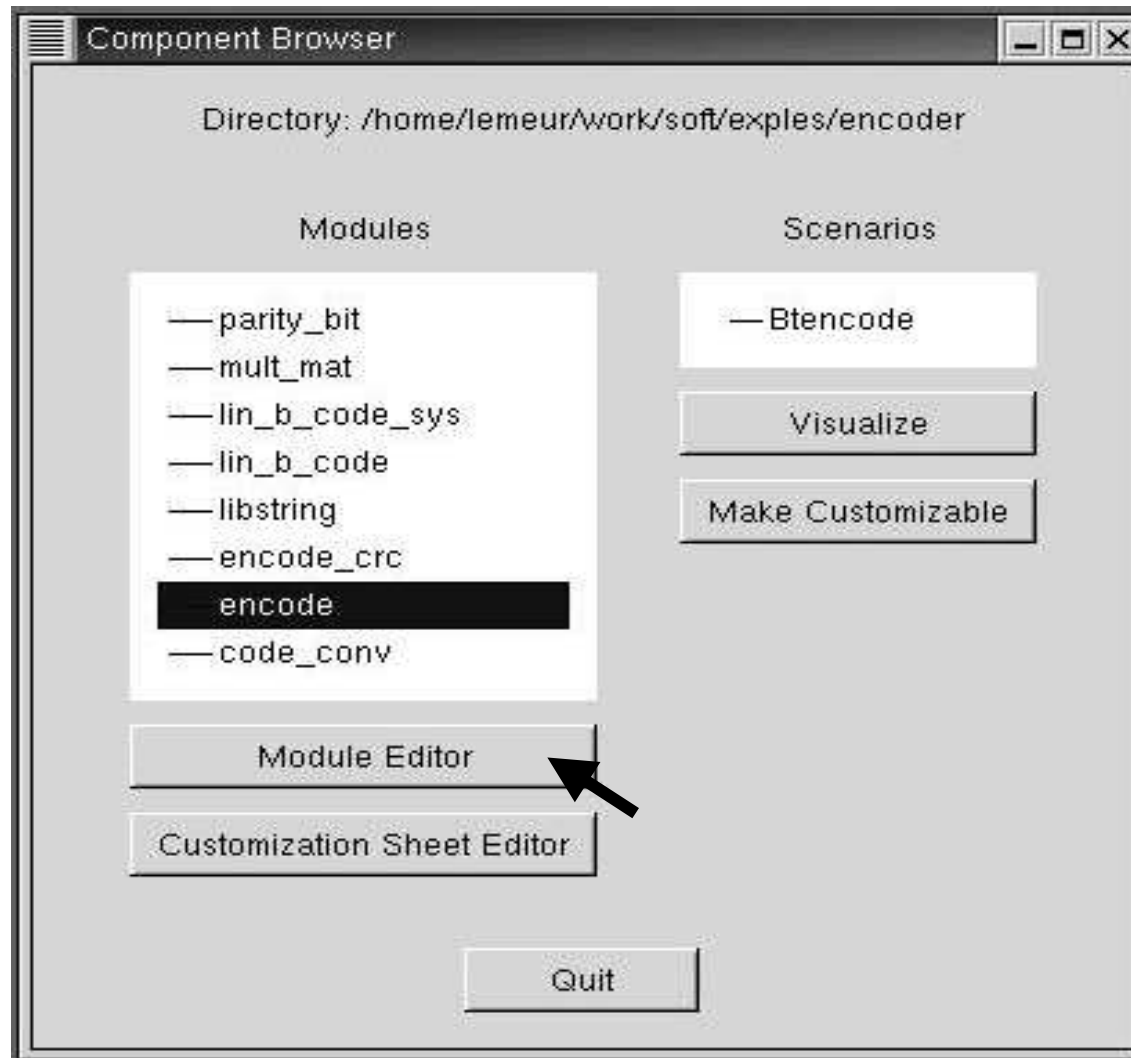
Developer interface



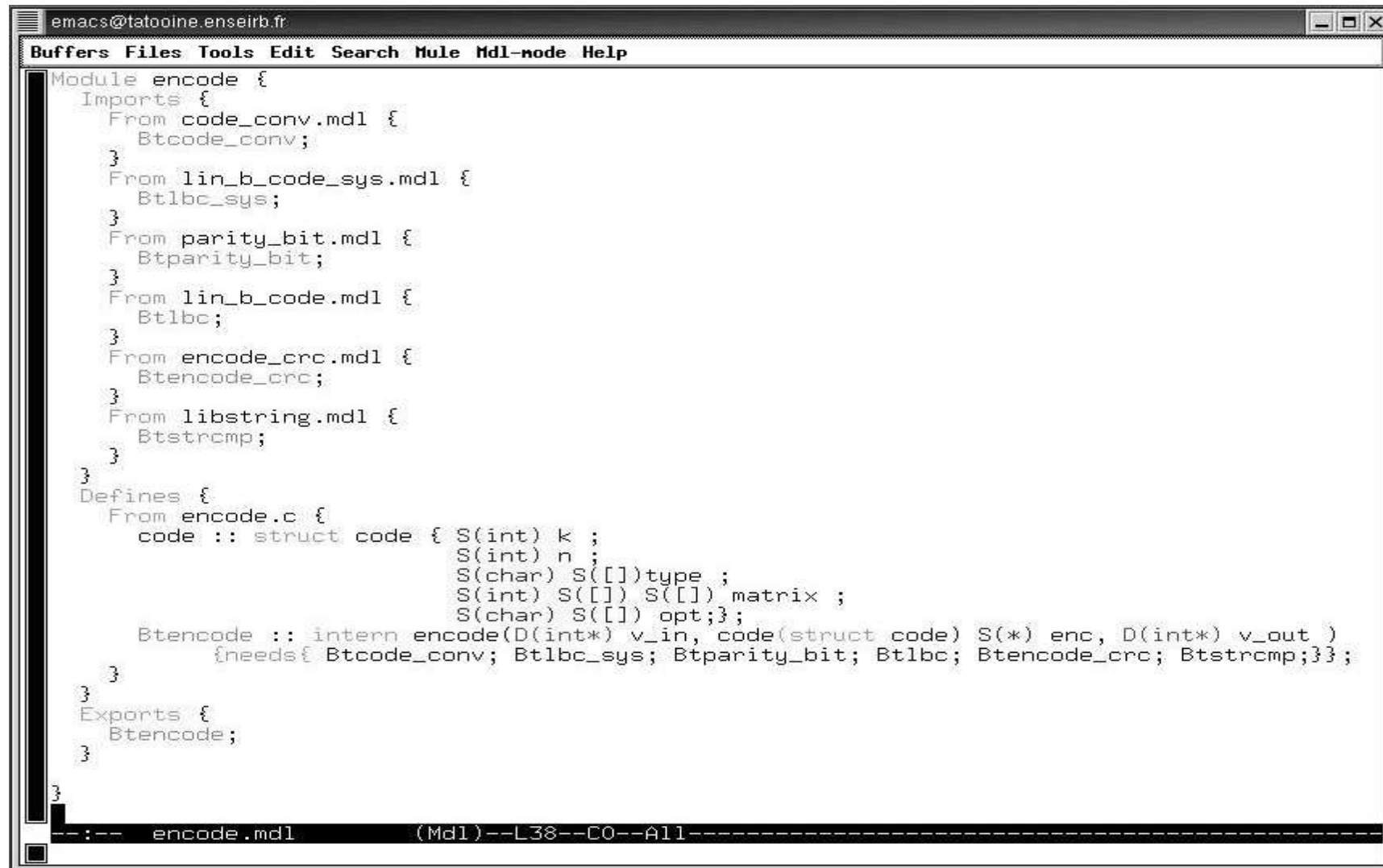
User interface



# Developer interface: configurable component generation



# Developer interface: configurable component generation



```
emacs@tatooine.enseirb.fr
Buffers Files Tools Edit Search Mule Mdl-mode Help
Module encode {
  Imports {
    From code_conv.mdl {
      Btcode_conv;
    }
    From lin_b_code_sys.mdl {
      Btlbc_sys;
    }
    From parity_bit.mdl {
      Btparity_bit;
    }
    From lin_b_code.mdl {
      Btlbc;
    }
    From encode_crc.mdl {
      Btencode_crc;
    }
    From libstring.mdl {
      Btstrcmp;
    }
  }
  Defines {
    From encode.c {
      code :: struct code { S(int) k ;
                          S(int) n ;
                          S(char) S([])type ;
                          S(int) S([]) S([]) matrix ;
                          S(char) S([]) opt;};

      Btencode :: intern encode(D(int*) v_in, code(struct code) S(*) enc, D(int*) v_out )
        {needs{ Btcode_conv; Btlbc_sys; Btparity_bit; Btlbc; Btencode_crc; Btstrcmp;}};
    }
  }
  Exports {
    Btencode;
  }
}
--:-- encode.md1 (Mdl)--L38--E0--A11-----
```

# Developer interface: configurable component generation

Customization Sheet Editor

STRUCTURES

struct code{int k; int n; char [] type; int [] [] matrix; char [] opt}

k:	<input type="text" value="number of bits to treat"/>	<input checked="" type="radio"/> USER	<input type="radio"/> P_CALL
n:	<input type="text" value="number of resulting bits"/>	<input checked="" type="radio"/> USER	<input type="radio"/> P_CALL
type:	<input type="text" value="type of encoding"/>	<input checked="" type="radio"/> USER	<input type="radio"/> P_CALL
matrix:	<input type="text" value="data generator"/>	<input type="radio"/> USER	<input checked="" type="radio"/> P_CALL
opt:	<input type="text" value="matrix properties"/>	<input checked="" type="radio"/> USER	<input type="radio"/> P_CALL

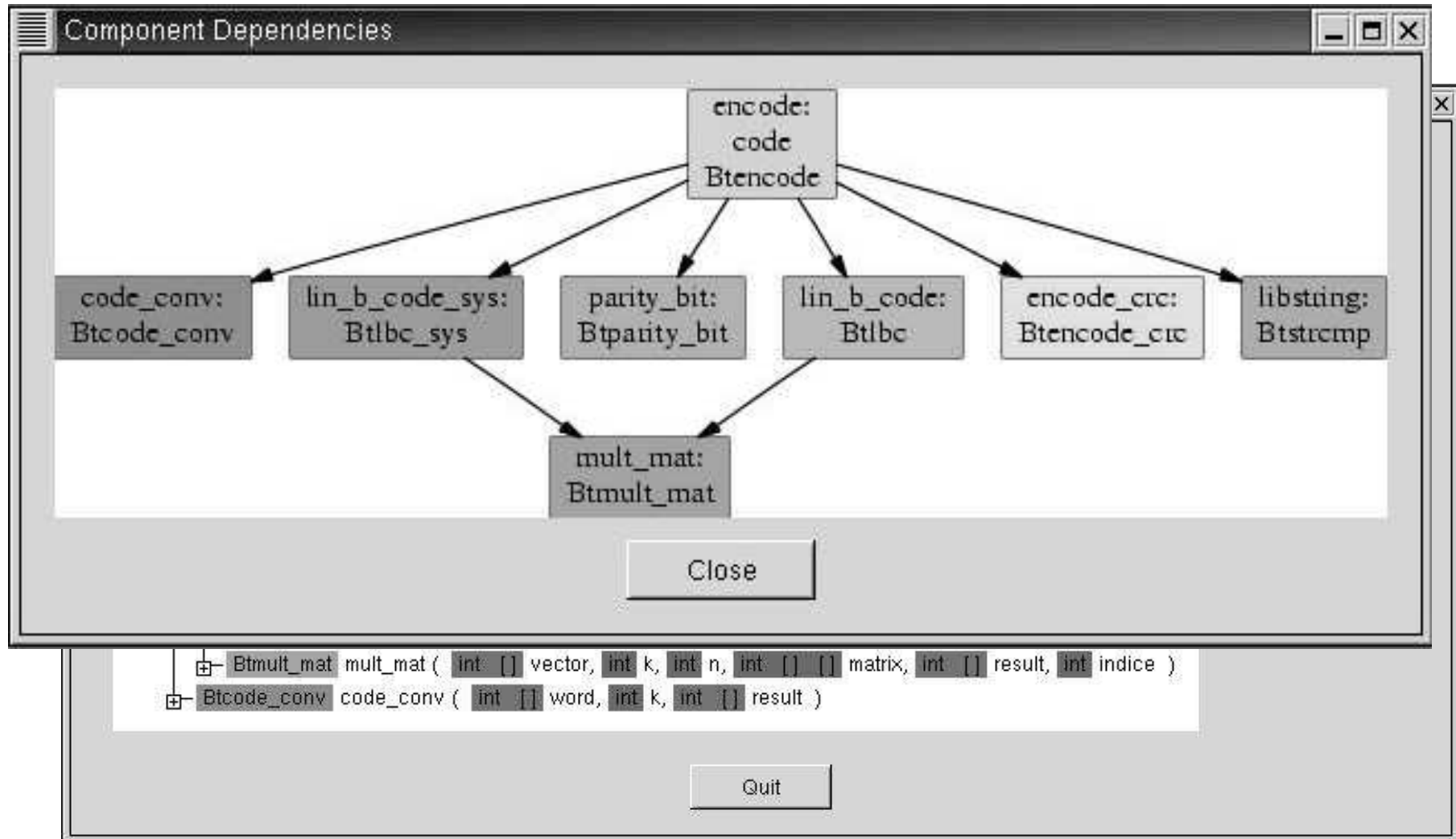
GLOBAL VARIABLES

FUNCTIONS

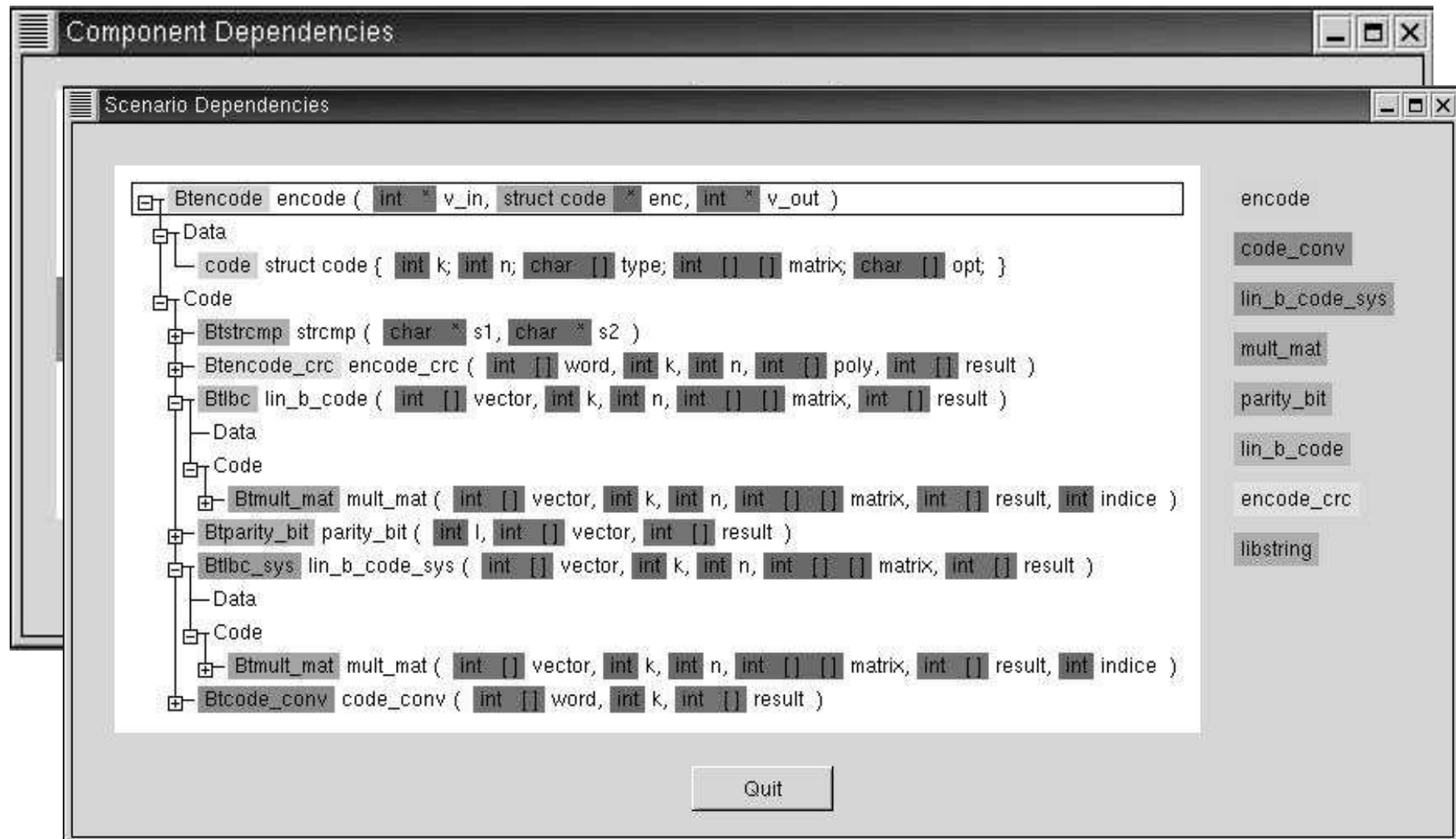
encode(int \* v\_in, struct code \* enc, int \* v\_out)

v_in:	<input type="text" value="word to encode"/>	<input checked="" type="radio"/> USER	<input type="radio"/> P_CALL
enc:	<input type="text" value="encoder properties"/>	<input checked="" type="radio"/> USER	<input type="radio"/> P_CALL
v_out:	<input type="text" value="encoded word"/>	<input checked="" type="radio"/> USER	<input type="radio"/> P_CALL

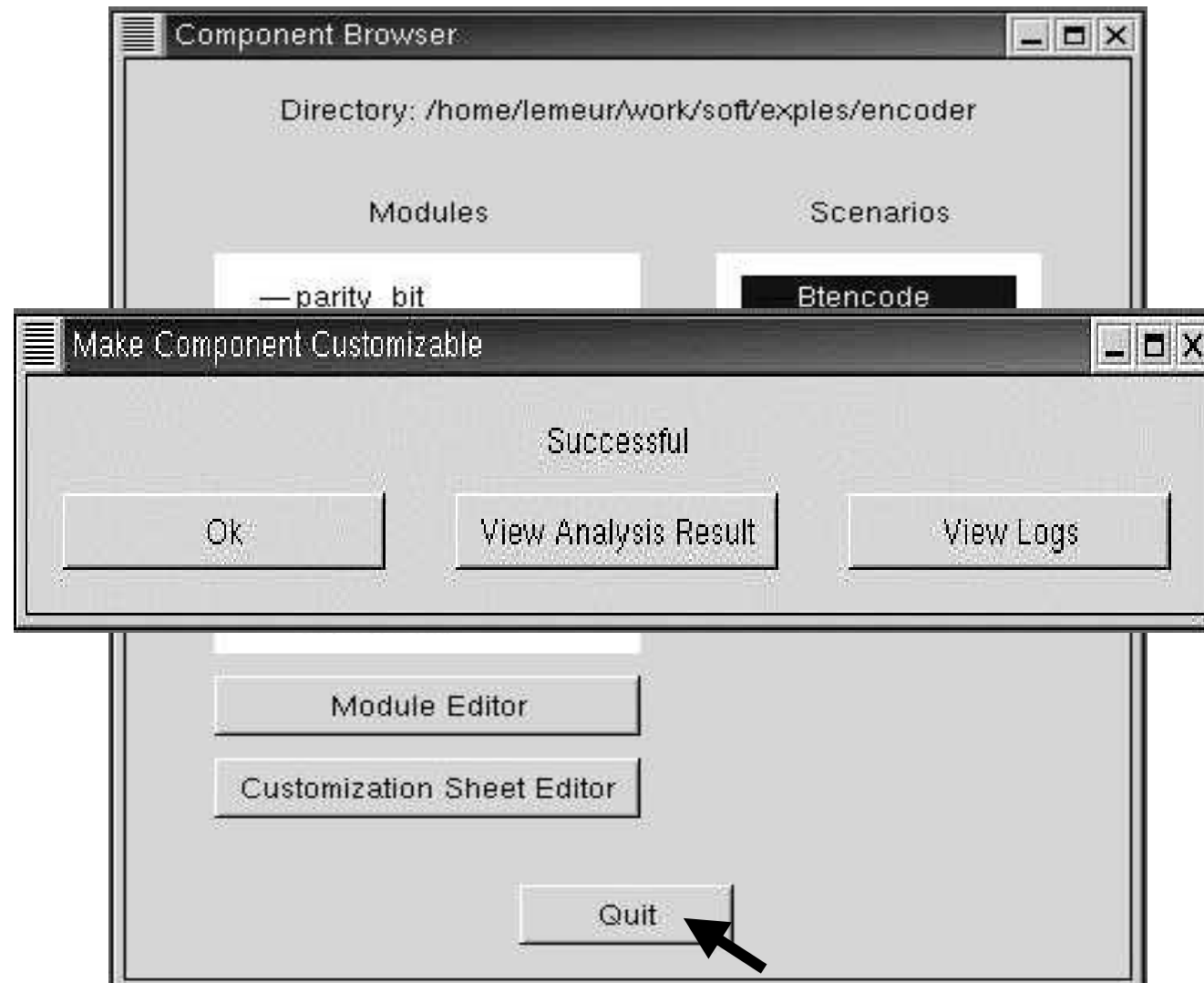
# Developer interface: configurable component generation



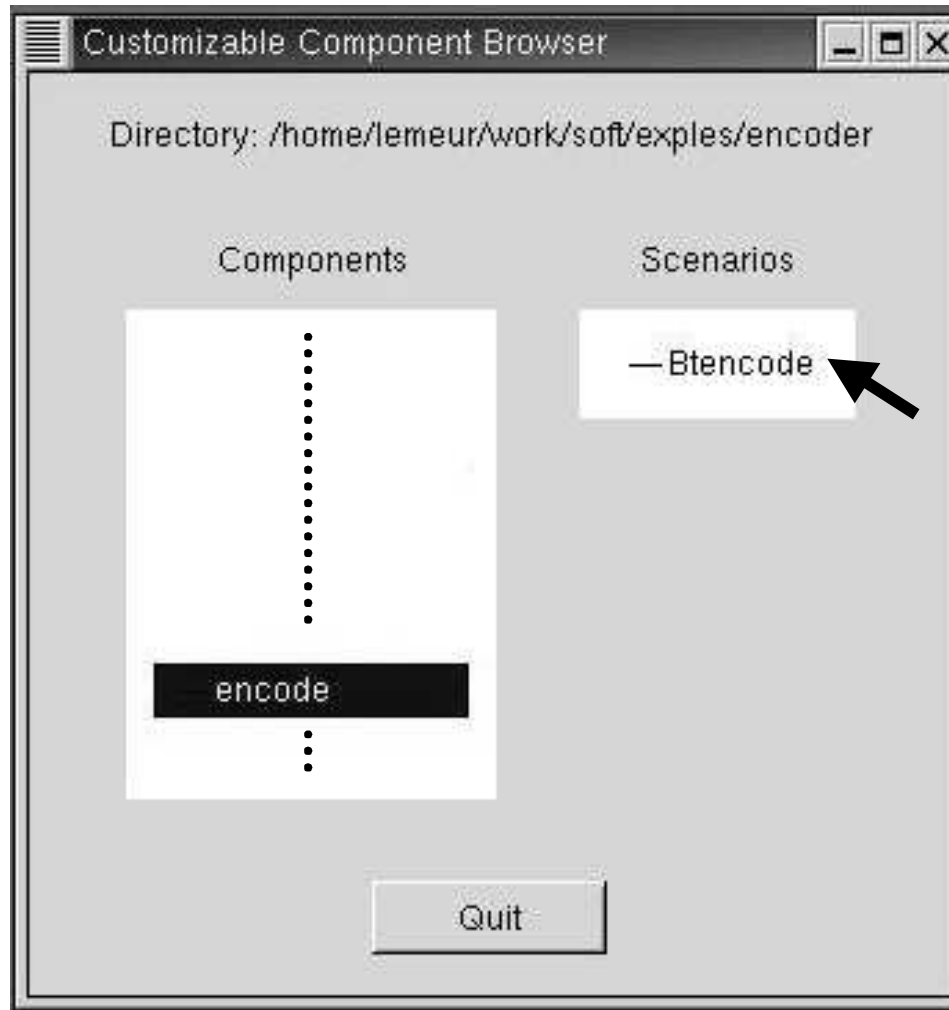
# Developer interface: configurable component generation



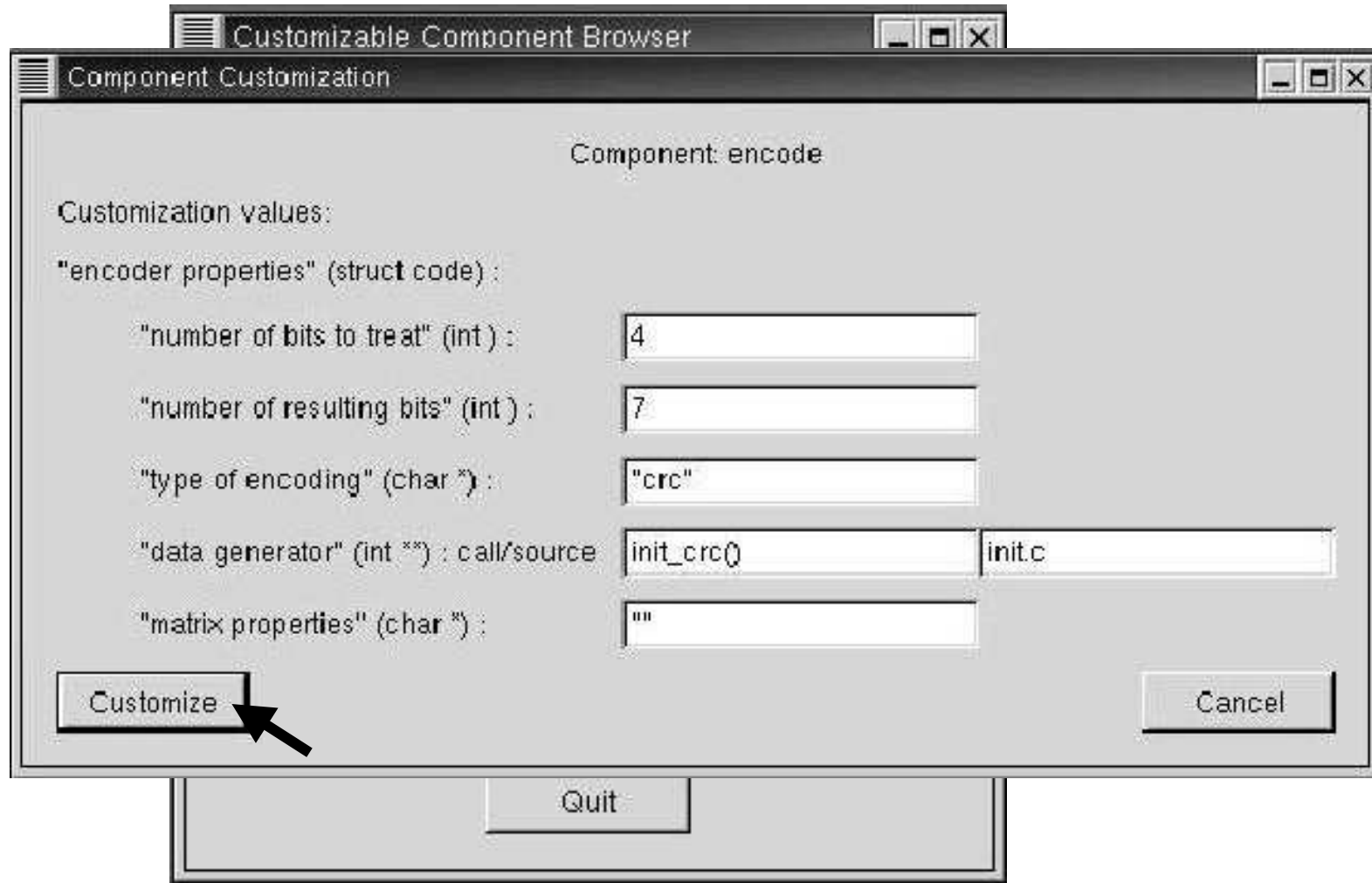
# Developer interface: configurable component generation



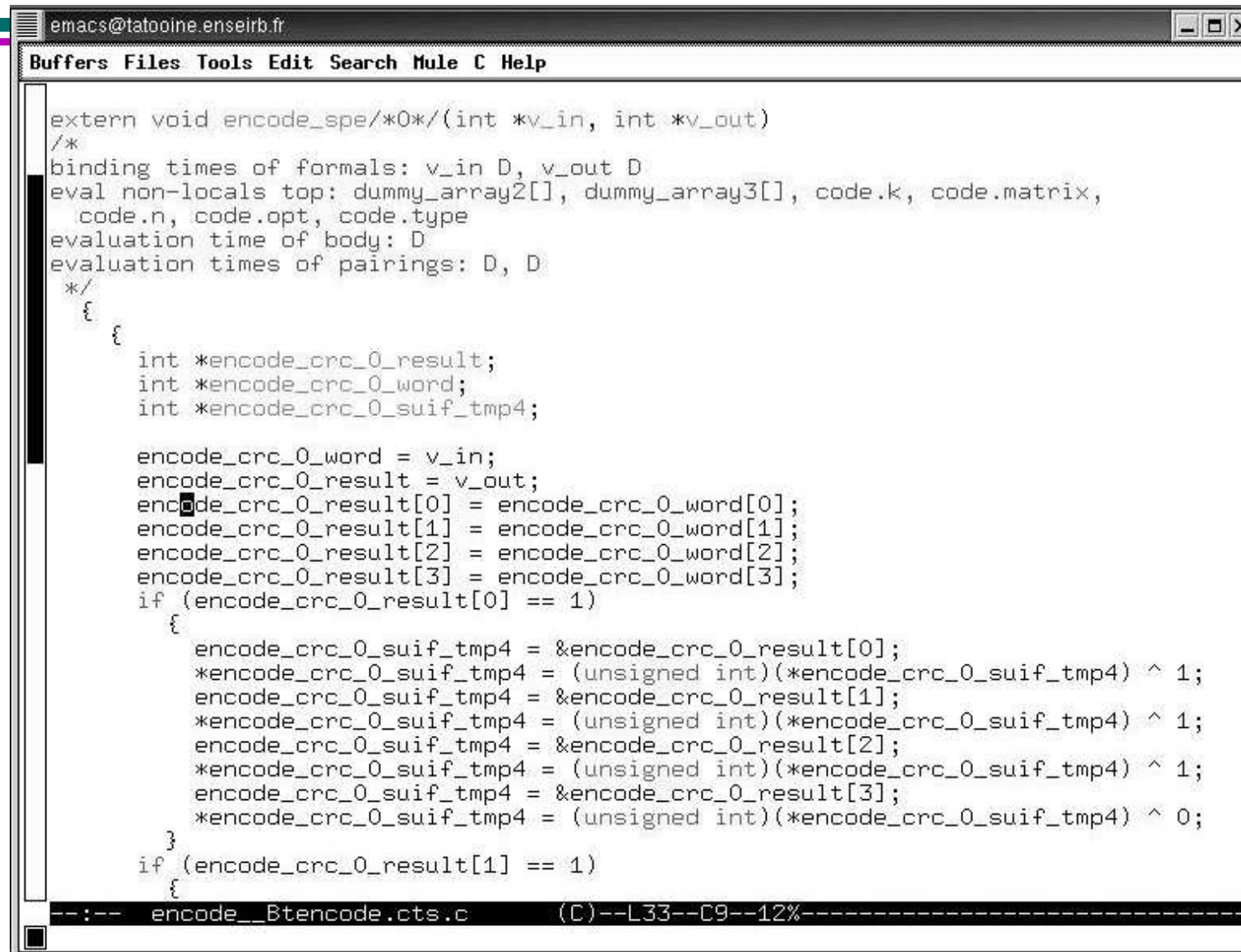
# User interface: configured component generation



# User interface: configured component generation



# User interface: configured component generation



```
emacs@tatooine.enseirb.fr
Buffers Files Tools Edit Search Mule C Help

extern void encode_spe/*0*/(int *v_in, int *v_out)
/*
binding times of formals: v_in D, v_out D
eval non-locals top: dummy_array2[], dummy_array3[], code.k, code.matrix,
code.n, code.opt, code.type
evaluation time of body: D
evaluation times of pairings: D, D
*/
{
    {
        int *encode_crc_0_result;
        int *encode_crc_0_word;
        int *encode_crc_0_suif_tmp4;

        encode_crc_0_word = v_in;
        encode_crc_0_result = v_out;
        encode_crc_0_result[0] = encode_crc_0_word[0];
        encode_crc_0_result[1] = encode_crc_0_word[1];
        encode_crc_0_result[2] = encode_crc_0_word[2];
        encode_crc_0_result[3] = encode_crc_0_word[3];
        if (encode_crc_0_result[0] == 1)
        {
            encode_crc_0_suif_tmp4 = &encode_crc_0_result[0];
            *encode_crc_0_suif_tmp4 = (unsigned int)(*encode_crc_0_suif_tmp4) ^ 1;
            encode_crc_0_suif_tmp4 = &encode_crc_0_result[1];
            *encode_crc_0_suif_tmp4 = (unsigned int)(*encode_crc_0_suif_tmp4) ^ 1;
            encode_crc_0_suif_tmp4 = &encode_crc_0_result[2];
            *encode_crc_0_suif_tmp4 = (unsigned int)(*encode_crc_0_suif_tmp4) ^ 1;
            encode_crc_0_suif_tmp4 = &encode_crc_0_result[3];
            *encode_crc_0_suif_tmp4 = (unsigned int)(*encode_crc_0_suif_tmp4) ^ 0;
        }
        if (encode_crc_0_result[1] == 1)
        {

```

---:-- encode\_\_Btencode.cts.c (C)--L33--C9--12%-----

# Conclusion

---

---

- Declaration language
  - specialization predictable with respect to the declarations
- Environment
  - compilation and verification (targeted to Tempo)
  - graphical interfaces
- Real size applications: FEC, RPC, FFT, BPF, ...
- A step towards integrating specialization in the software development process

# Future work

---

---

- Declaration language extensions
  - aliases, external functions
  - finer grained control over transformations
- Tools to assist specialization
  - debugging environment
  - estimation of the size and execution time of the result
- New applications
  - graphic stack (Philips) , network stack -- both ongoing
  - runtime component re-configuration