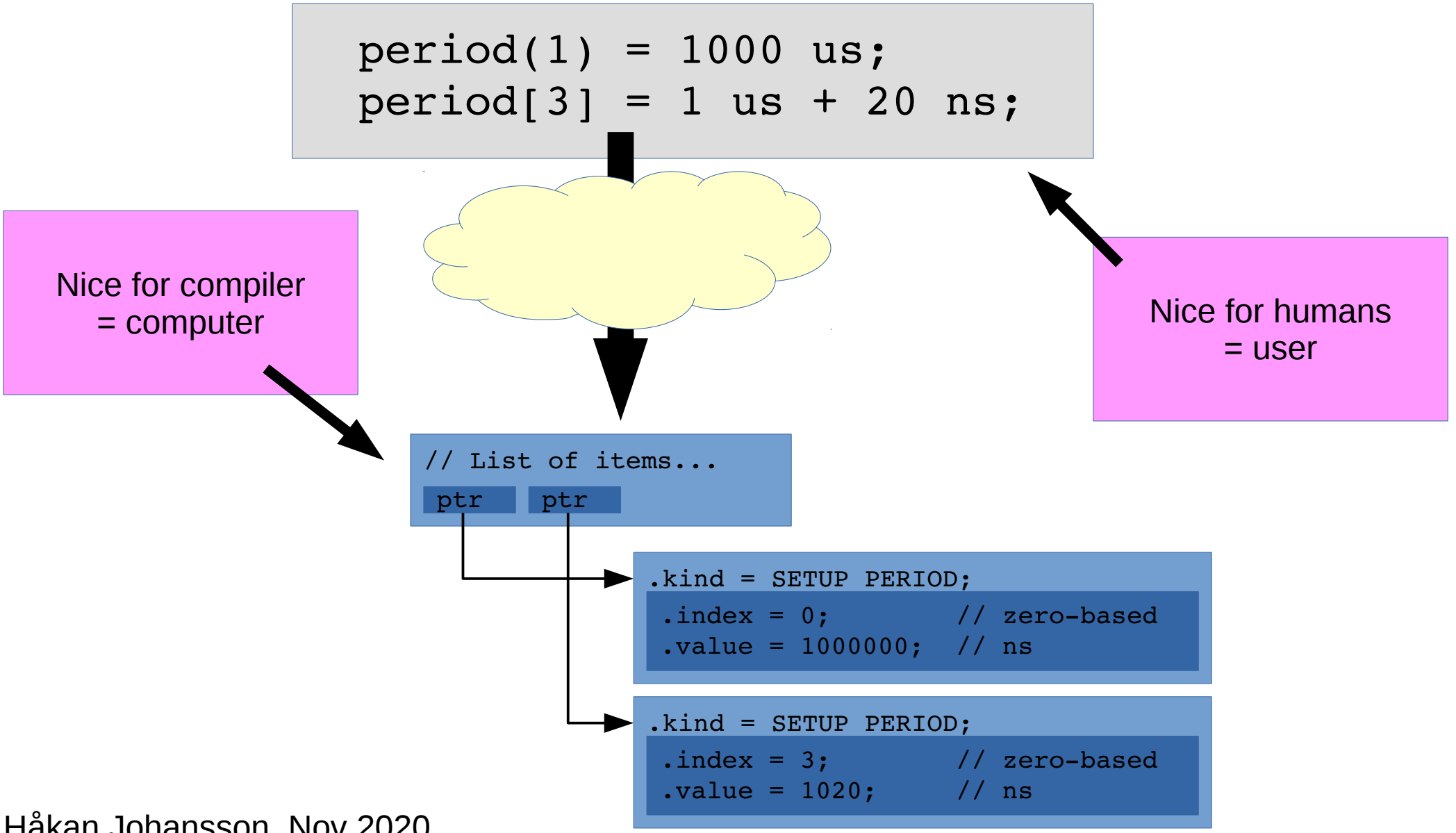
Productive parsing



Structured text files

- Structure!
  - Adapted to information.
- Easy to view and edit.

```
/* OPTION: Additional deadtime are received on ECL_IN 14-16.
 * Note: While deadtime is active, no triggers are processed.
 */
SECTION(option_deadtime_inputs)
{
  DEADTIME_IN(2) = ALL_OR(1);
  all_or_mask[0] = ECL_IN(14) or ECL_IN(15) or ECL_IN(16);
}
```

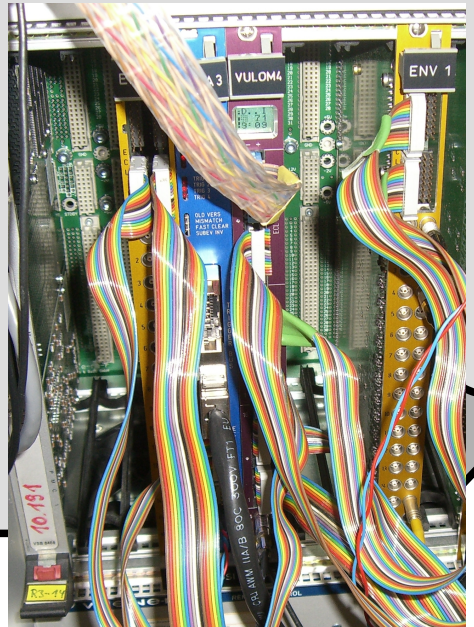
Structured text files

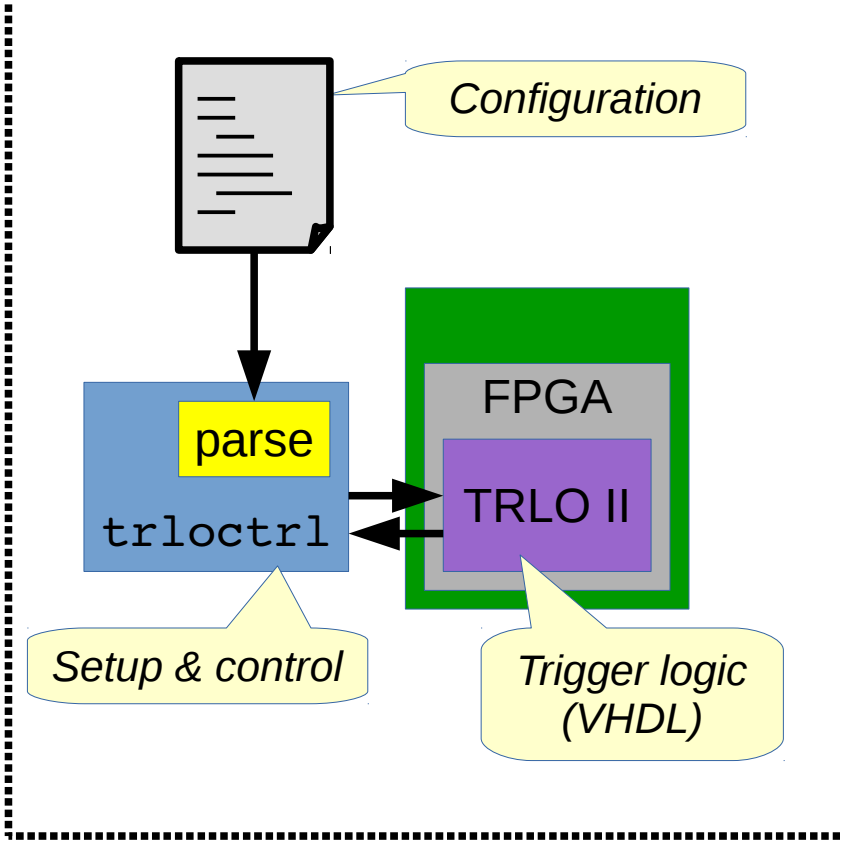
- Structure!
  - Adapted to information.
- Easy to view and edit.
- Suitable for version control.
- Units :-)

```
/* OPTION: Additional deadtime are received on ECL_IN 14-16.
 * Note: While deadtime is active, no triggers are processed.
 */
SECTION(option_deadtime_inputs)
{
  DEADTIME_IN(2) = ALL_OR(1);
  all_or_mask[0] = ECL_IN(14) or ECL_IN(15) or ECL_IN(16);
}

/* OPTION: Send fast trigger output on LEMO 1. */
SECTION(option_master_start)
{
  sum_out_stretch = 50 ns;
  LEMO_OUT(1) = MASTER_START;
}

/* OPTION: Two pulsers (10 Hz, 1000 Hz) on LEMO_OUT 1, 2. */
SECTION(option_pulsers)
{
  /* 1000 Hz Pulser */
  period(1) = 1000 us;
  LEMO_OUT(1) = PULSER(1);

  /* 10 Hz Pulser */
  period(2) = 100000 us;
  LEMO_OUT(2) = PULSER(2);
}
```

Configuration parser

```
/* OPTION: Additional deadtime are received on ECL_IN 14-16.
 * Note: While deadtime is active, no triggers are processed.
 */
SECTION(option_deadtime_inputs)
{
  DEADTIME_IN(2) = ALL_OR(1);
  all_or_mask[0] = ECL_IN(14) or ECL_IN(15) or ECL_IN(16);
}

/* OPTION: Send fast trigger output on LEMO 1. */
SECTION(option_master_start)
{
  sum_out_stretch = 50 ns;
  LEMO_OUT(1) = MASTER_START;
}

/* OPTION: Two pulsers (10 Hz, 1000 Hz) on LEMO_OUT 1, 2. */
SECTION(option_pulsers)
{
  /* 1000 Hz Pulser */
  period(1) = 1000 us;
  LEMO_OUT(1) = PULSER(1);

  /* 10 Hz Pulser */
  period(2) = 100000 us;
  LEMO_OUT(2) = PULSER(2);
}
```

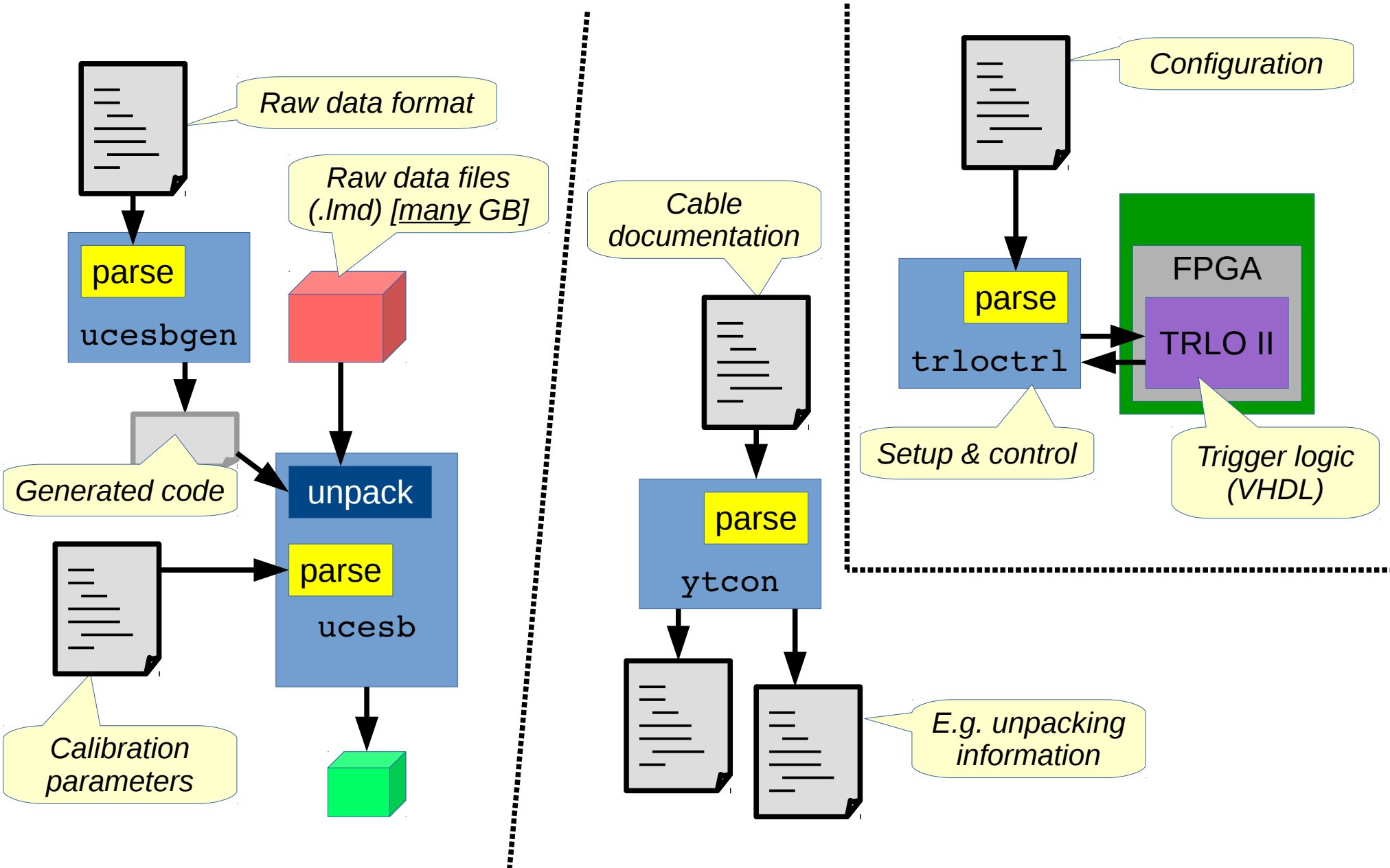




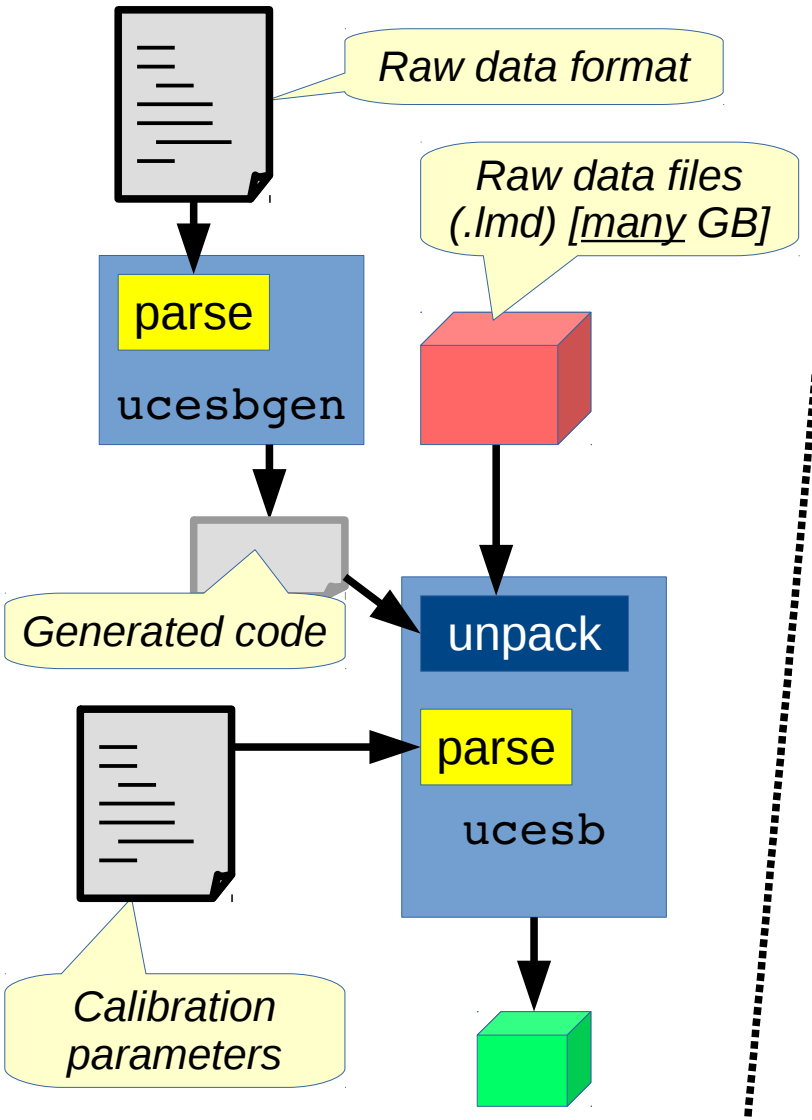
Usage example:

```
./trloctrl –addr=3 option_pulsers
```

Parsers at the heart of tools



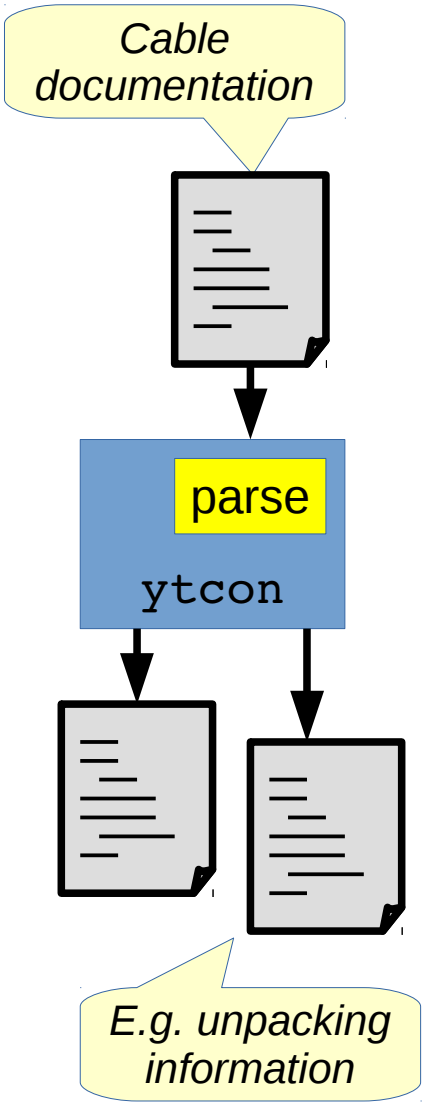
Data format parser



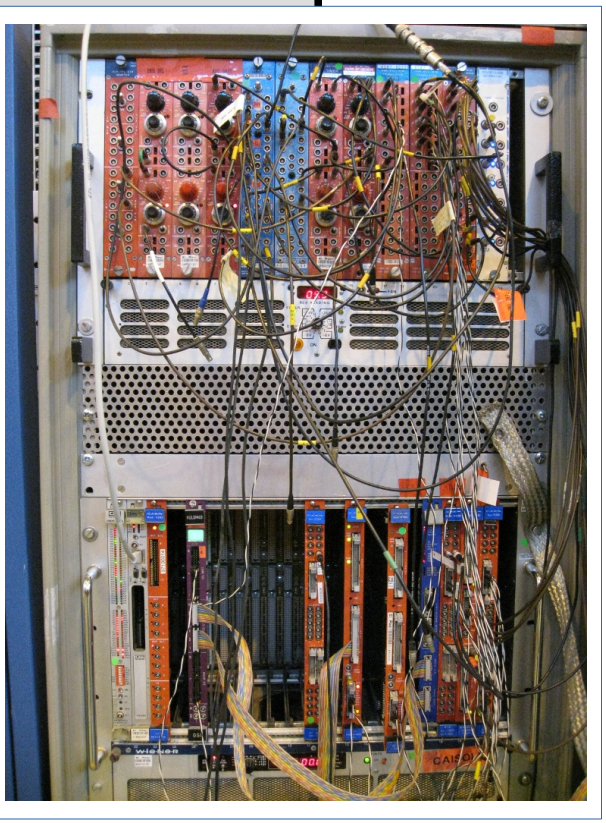

```
VME_CAEN_V830(geom)
{
  MEMBER(DATA32 data[32] ZERO_SUPPRESS);

  UINT32 header
    {
      0_15:  event_number;
      16_17: ts;
      18_23: count;
      24_25: undefined;
      26:    1;
      27_31: geom = MATCH(geom);
    }

  list(0 <= index < header.count)
    {
      UINT32 ch_data NOENCODE
        {
          0_25:  value;
          26:    0;
          27_31: channel;

          ENCODE(data[channel],(value=value));
        }
    }
}
```

(UCESB = unpack & check every single bit)

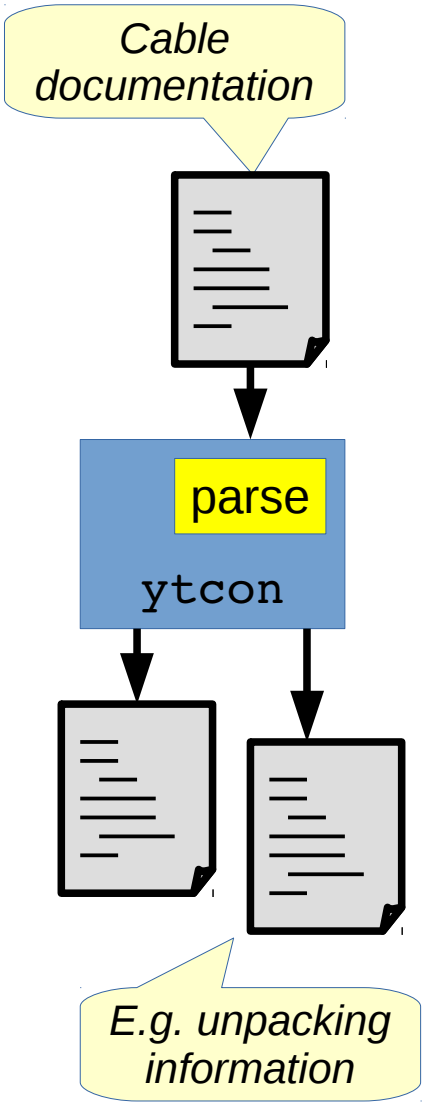
Cable documentation parser




```
// Constant fraction discriminator,
// with output to TDC and scaler, etc...
CF8103(r12c2s1)
{
  SERIAL("LCF6343");

 in1_8: "N11 CFTN1" <- , r13c1s1(SPLIT)/t1_8;
 th1_8: "1/1"       -> , r11c1s1(SCALER)/in0_7;
 tb1_8: "CR2 SL1"   -> , .s15(DELAY)/in1_8;

 m:    .c11s3/in1;
 test:   .s23/out1;
 mux_tb: .s22/in1a;   mux_e: .s22/in5a;   mux_mon: .s22/in9a;
}

// Delay of timing signal (before going to TDC)
DL1610(r12c2s15)
{
 in1_8:  <- "CR2 SL1"  , .s1/tb1_8;
 in9_16: <- "CR2 SL2"  , .s2/tb1_8;

 outa1_16:              r11c2s7(TDC)/in0_15;
}

// A LeCroy 1875 Fastbus TDC
LECROY1875(r11c2s7)
{
  SERIAL("L_T48834");

 in0_15:  r12c2s15/outa1_16;
 in16_31: r12c3s15/outa1_16;
 in32_47: r12c4s15/outa1_16;
  ;
}
```



Cable documentation parser

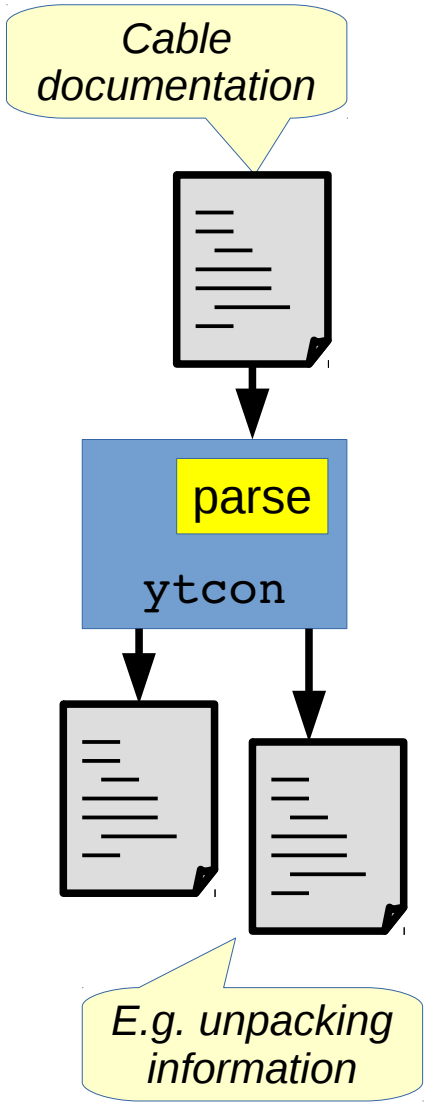



```
// Constant fraction discriminator,
// with output to TDC and scaler, etc...
CF8103(r12c2s1)
{
  SERIAL("LCF6343");

 in1_8: "N11 CFTN1" <- , r13c1s1(SPLIT)/t1_8;
 th1_8: "1/1"       -> , r11c1s1(SCALER)/in0_7;
 tb1_8: "CR2 SL1"   -> , .s15(DELAY)/in1_8;

 m:    .c11s3/in1;
 test:   .s23/out1;
 mux_tb: .s22/in1a;   mux_e: .s22/in5a;   mux_mon: .s22/in9a;
}

// Delay of timing signal (before going to TDC)
DL1610(r12c2s15)
{
 in1_8:  <- "CR2 SL1"  , .s1/tb1_8;
 in9_16: <- "CR2 SL2"  , .s2/tb1_8;

 outa1_16:              r11c2s7(TDC)/in0_15;
}

// A LeCroy 1875 Fastbus TDC
LECROY1875(r11c2s7)
{
  SERIAL("L_T48834");

 in0_15:  r12c2s15/outa1_16;
 in16_31: r12c3s15/outa1_16;
 in32_47: r12c4s15/outa1_16;
  ;
}
```

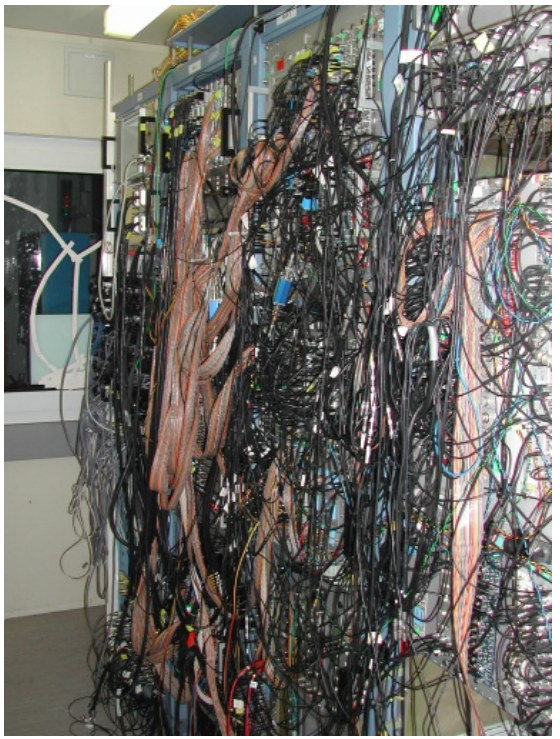
Every cable documented twice.

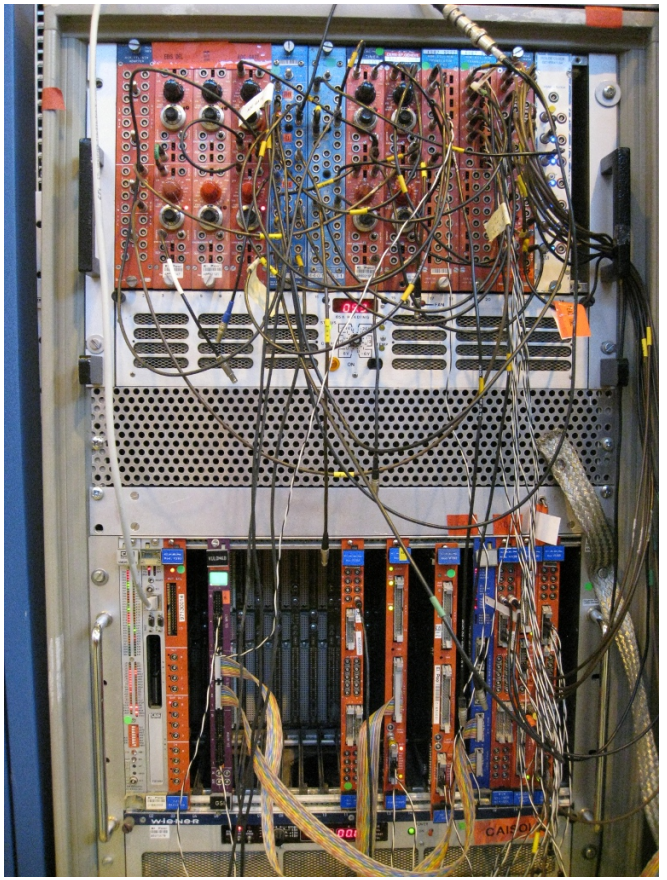
if not:
...
`ytcon`
gives error message.

Cable documentation parser

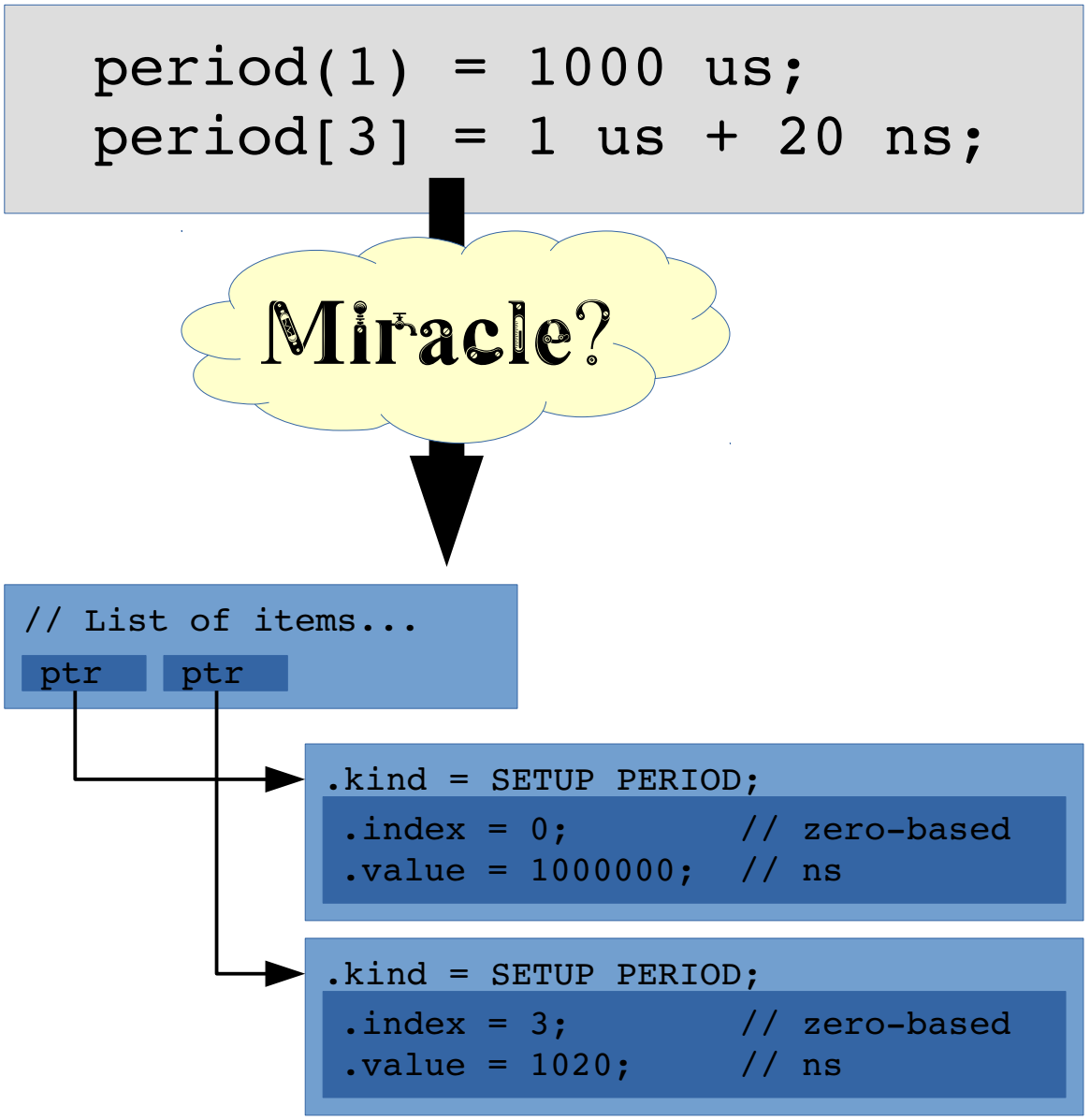



```
// Constant fraction discriminator,
//
CF8
{
  S

 in                                    t1_8;
 th                                    /in0_7;
 th                                    _8;

 m:
 te                                      mux
 mu
}

//                                     to TD
DL1
{
 in
 in

 ou                                    0_15;
}

//
LEC
{
  SERIAL("L_T48834");

 in0_15:  r12c2s15/outa1_16;
 in16_31: r12c3s15/outa1_16;
 in32_47: r12c4s15/outa1_16;
  ;
}
```





Lexical analysis & parsing



Matryoshka / Babushka method

```
period(1) = 1000 us;
period[3] = 1 us + 20 ns;
```

Miracle?

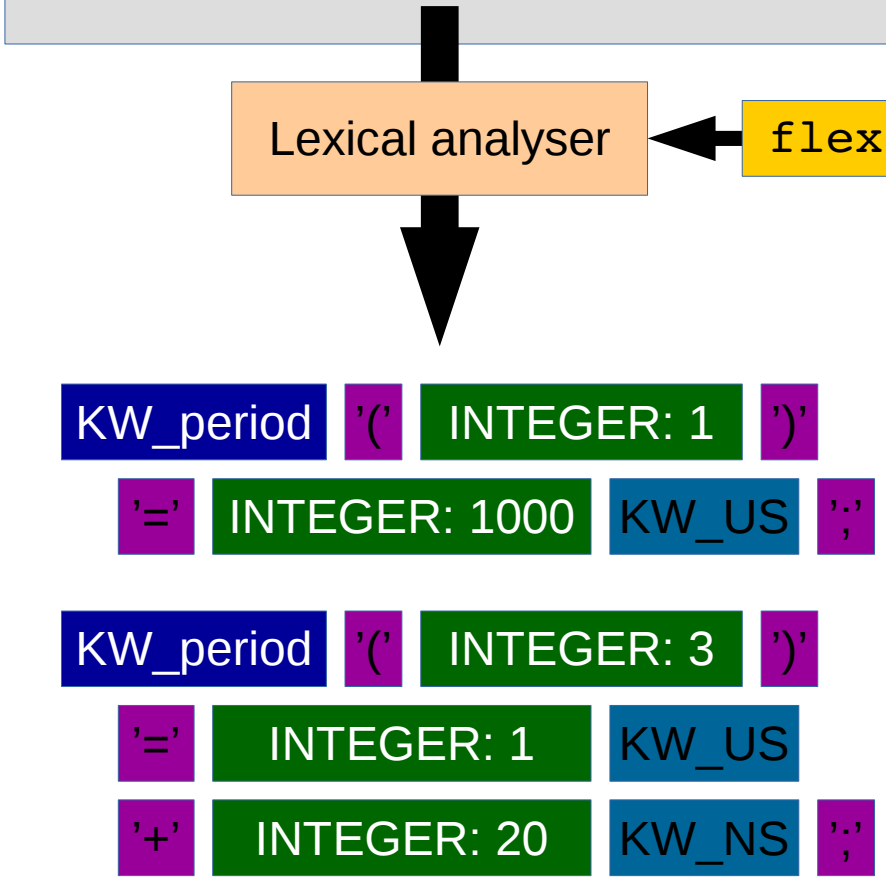
KW_period '(' INTEGER: 1 ')'
'=' INTEGER: 1000 KW_US ';'

KW_period '(' INTEGER: 3 ')'
'=' INTEGER: 1 KW_US
'+' INTEGER: 20 KW_NS ';'

https://upload.wikimedia.org/wikipedia/commons/d/d2/Russian-Matroshka_no_bg.jpg

Lexical analysis

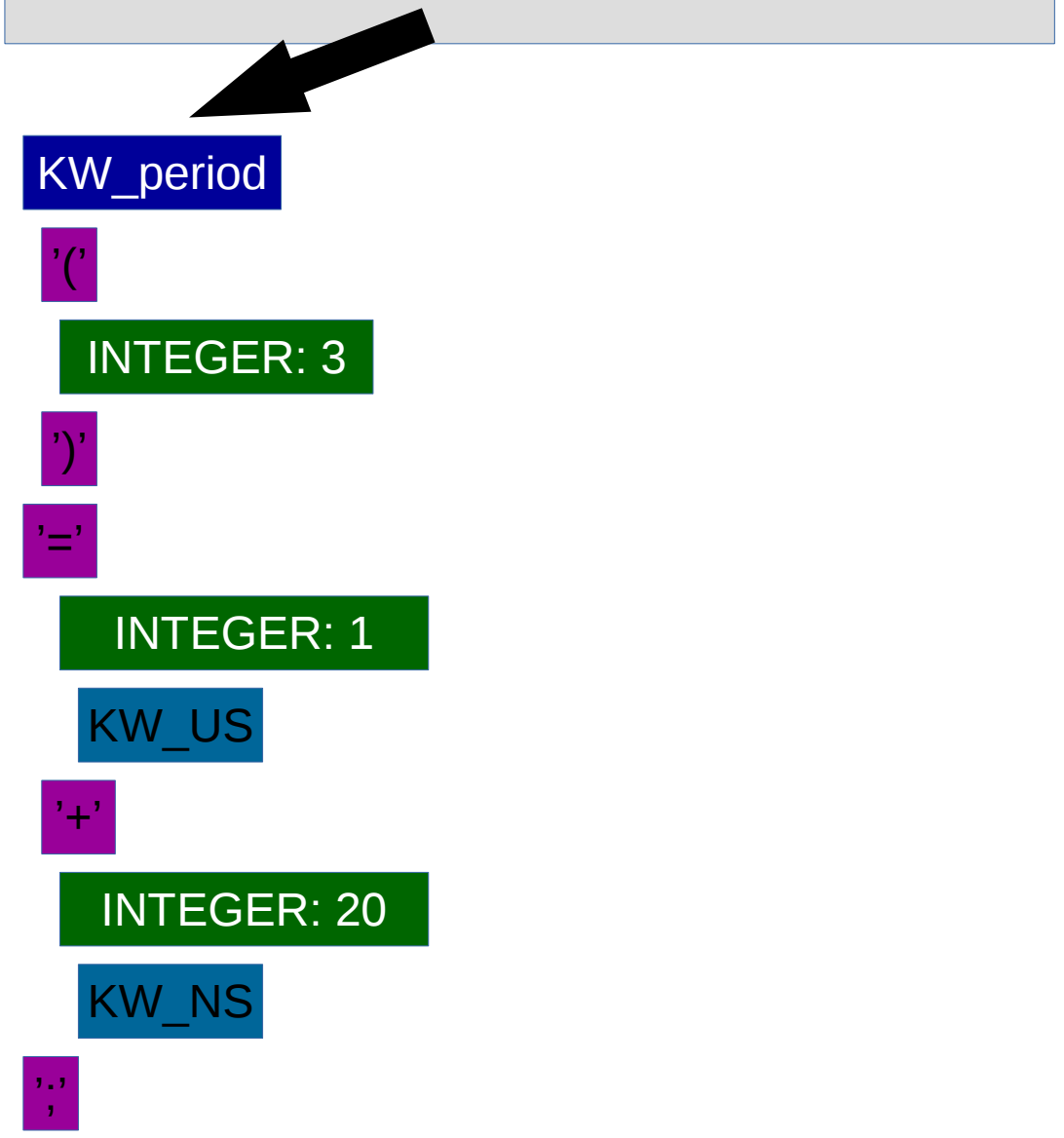
```
period(1) = 1000 us;
period[3] = 1 us + 20 ns;
```




```
[0-9]+       {
               yylval.iValue = atoi(yytext);
               return INTEGER;
             }

0x[0-9a-fA-F]+ {
               yylval.iValue = (int) strtol(yytext+2,NULL,16);
               return HEXADECIMAL;
             }

[+-]?[0-9]+"."[0-9]*([eE][+-]?[0-9]+)? {
               yylval.fValue = atof(yytext);
               return DOUBLE;
             }

"period"     { return KW_period; }
"delay"      { return KW_delay; }
"stretch"    { return KW_stretch; }

"ns"         { return      ; }
"us"         { return      ; }
"ms"         { return      ; }

[_a-zA-Z][_a-zA-Z0-9]* {
               int token;
               yylval.strValue = setup_get_string(yytext);
               token = setup_get_var(yylval.strValue,&yylval);
               if (token)
                 return      ;
               return IDENTIFIER;
             }

 /* Individual tokens that we accept */

[-+*/;:(){},=\.|<>\[\]] {
               return      ;
             }

[ \t]+       {  }       /* ignore whitespace */
[\n]+        {  }       /* ignore whitespace */

"//".*       { /* Consume, discard C++ comment. */ }

.            { FATAL("Unknown character: '%s'.",yytext); }
```

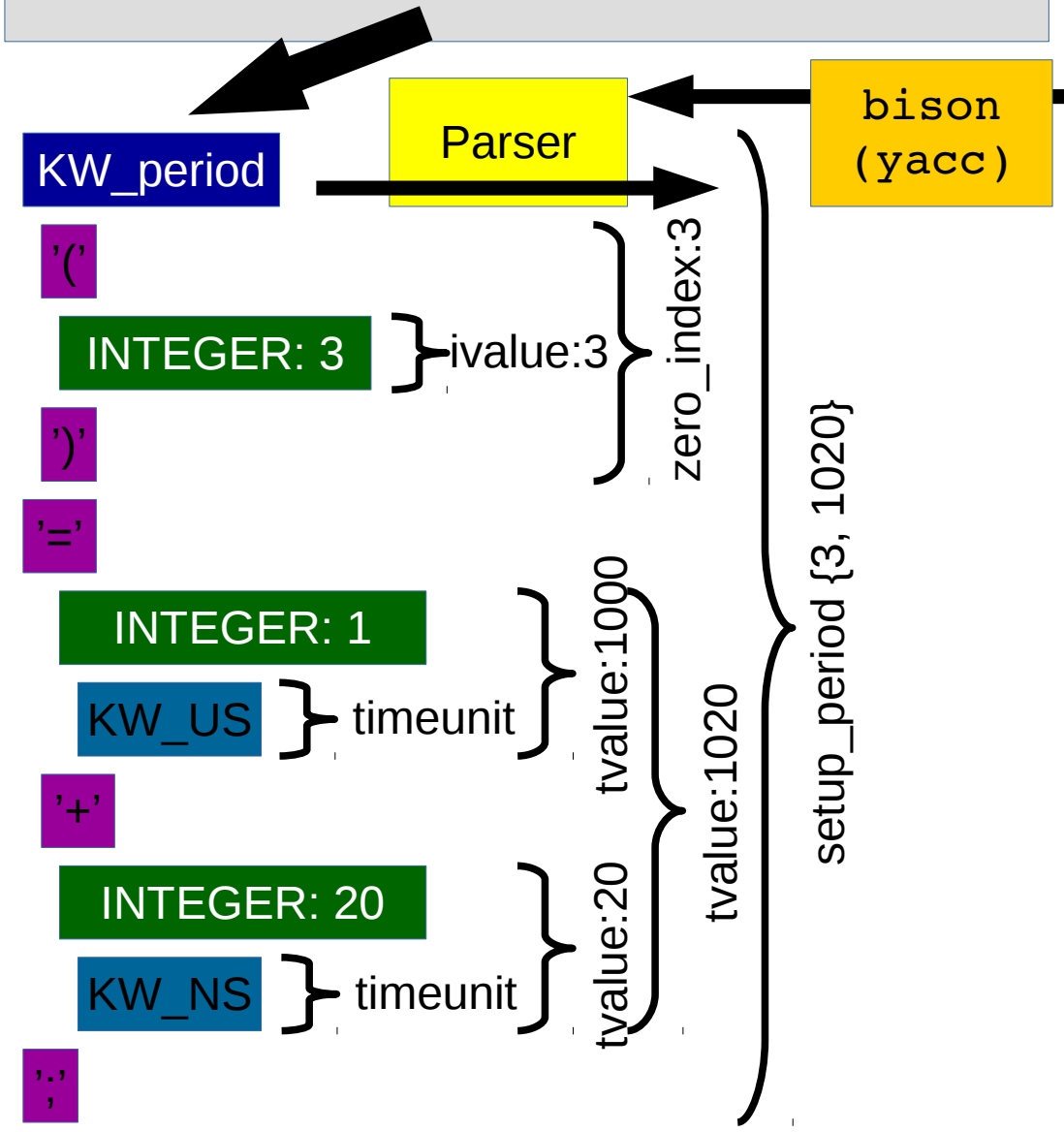
[rearrange]

```
period[3] = 1 us + 20 ns;
```



Parsing

```
period[3] = 1 us + 20 ns;
```




```
stmt:
          [ ]                   { $$ = NULL; }
        | simple_setup          { $$ = $1; }
        ;

simple_setup:
          ...
        | setup_period          { $$ = $1; }
        ;

setup_period:
          KW_period index [ ] tvalue [ ]
                                { $$ = SETUP_IND_TIME(PERIOD,$2,$4); }
        ;

index:
          zero_index            { $$ = $1; }
        | one_index             { $$ = $1; }
        ;

zero_index:
          [ ] ivalue [ ]        { if ($2.val < 0)
                                    FATAL("Zero-based index [%d] < 0 "
                                          "not allowed.",$2.val);
                                  $$ = $2.val; }
        ;

one_index:
          [ ] ivalue [ ]        { if ($2.val <= 0)
                                    FATAL("One-based index (%d) < 1 "
                                          "not allowed.",$2.val);
                                  $$ = $2.val - 1; }
        ;

tvalue:
          INTEGER timeunit              { $$ = $1 * $2; }
        | TIMEVALUE                     { $$ = $1; }
        | [ ] tvalue %prec UMINUS       { $$ = -$2; }
        | tvalue [ ] tvalue             { $$ = $1 + $3; }
        | tvalue [ ] tvalue             { $$ = $1 - $3; }
        | tvalue [ ] ivalue             { $$ = $1 * $3.val; }
        | ivalue [ ] tvalue             { $$ = $1.val * $3; }
        | tvalue [ ] ivalue             { $$ = tvalue_div ($1,$3.val); }
        | [ ] tvalue [ ]                { $$ = $2; }
        ;

timeunit:
          [ ]           { $$ = 1; }
        | [ ]           { $$ = 1000; }
        | [ ]           { $$ = 1000000; }
        ;

ivalue:
          ...
```

Parsing

```
period[3] = 1 us + 20 ns;
```

```
stmt:
          [ ]                   { $$ = NULL; }
        | simple_setup          { $$ = $1; }
        ;

simple_setup:
          ...
        | setup_period          { $$ = $1; }
        ;

setup_period:
          KW_period index [ ] tvalue [ ]
                                { $$ = SETUP_IND_TIME(PERIOD,$2,$4); }
        ;
```

bison (yacc)

Parser

KW_period

'('

INTEGER: 3 } ivalue:3

')'

'='

INTEGER: 1

KW_US } timeunit

'+'

INTEGER: 20

KW_NS } timeunit

';'

tvalue:1000

tvalue:20

tvalue:1020

zero_index:3

setup_period

```
// List of items...
ptr   ptr
```

Item created

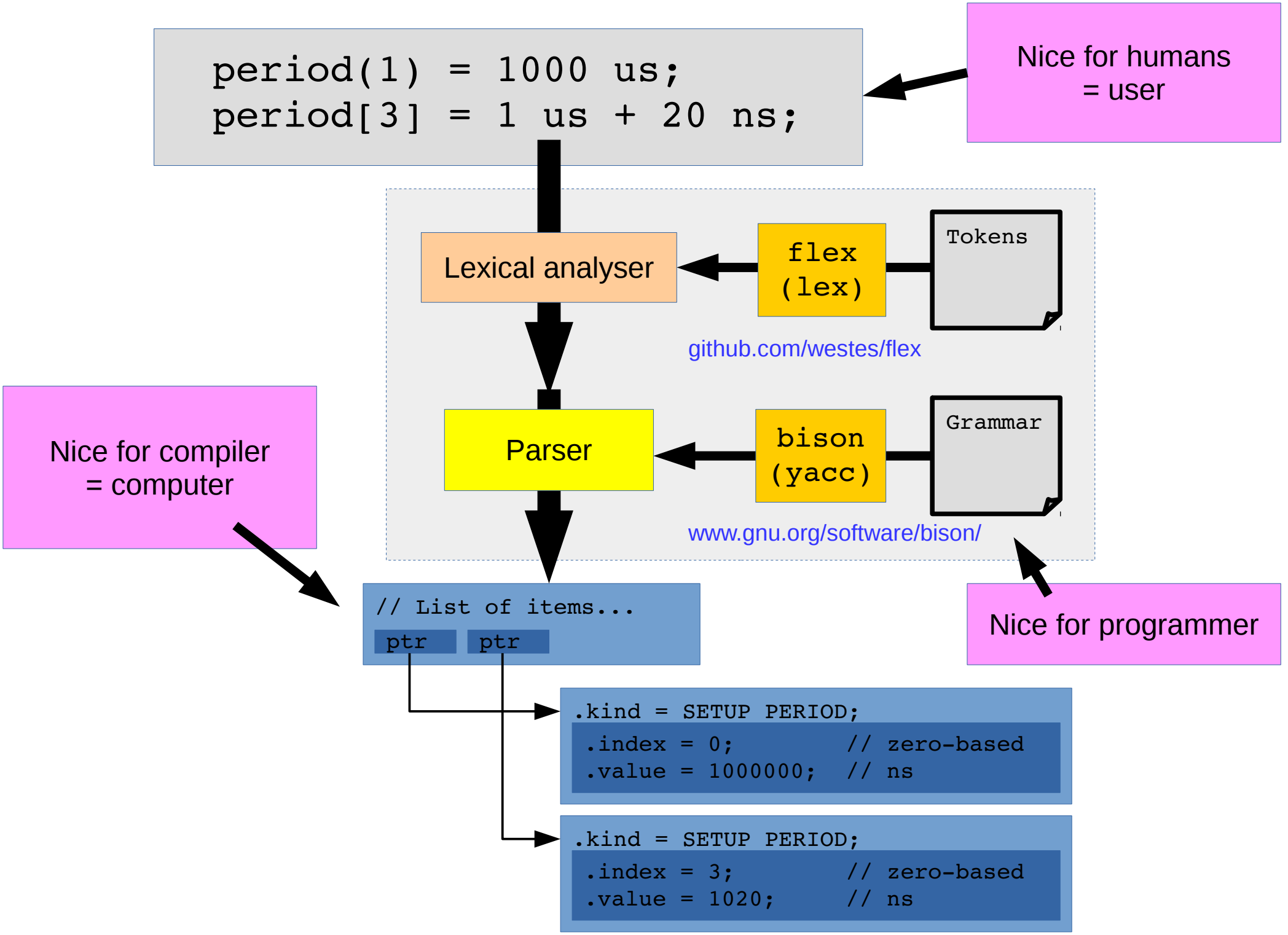
```
.kind = SETUP_PERIOD;
.index = 3;          // zero-based
.value = 1020;       // ns
```

Then inserted into list...

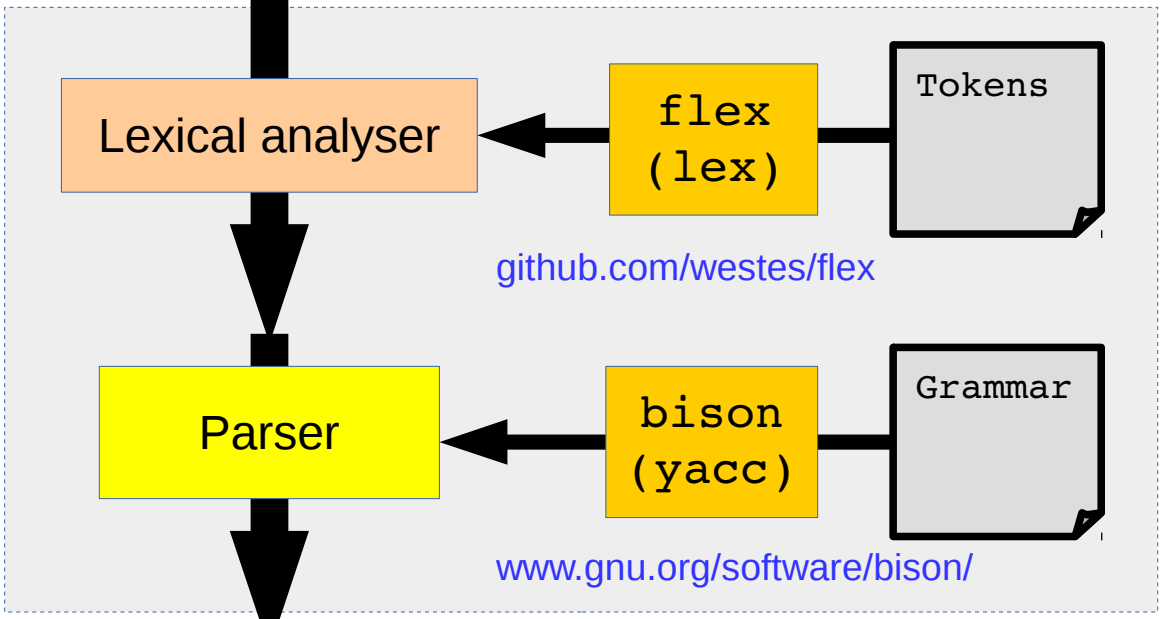
simple_setup

stmt

...


period(1) = 1000 us;
period[3] = 1 us + 20 ns;
Nice for humans
= user
Lexical analyser
flex
(lex)
Tokens
github.com/westes/flex
Parser
bison
(yacc)
Grammar
www.gnu.org/software/bison/
Nice for compiler
= computer
Nice for programmer
// List of items...
ptr
ptr
.kind = SETUP PERIOD;
.index = 0; // zero-based
.value = 1000000; // ns
.kind = SETUP PERIOD;
.index = 3; // zero-based
.value = 1020; // ns

```
period(1) = 1000 us;
period[3] = 1 us + 20 ns;
```




```
// List of items...
ptr   ptr
```

```
.kind = SETUP PERIOD;
.index = 0;          // zero-based
.value = 1000000;    // ns
```

```
.kind = SETUP PERIOD;
.index = 3;          // zero-based
.value = 1020;       // ns
```

Thank you!