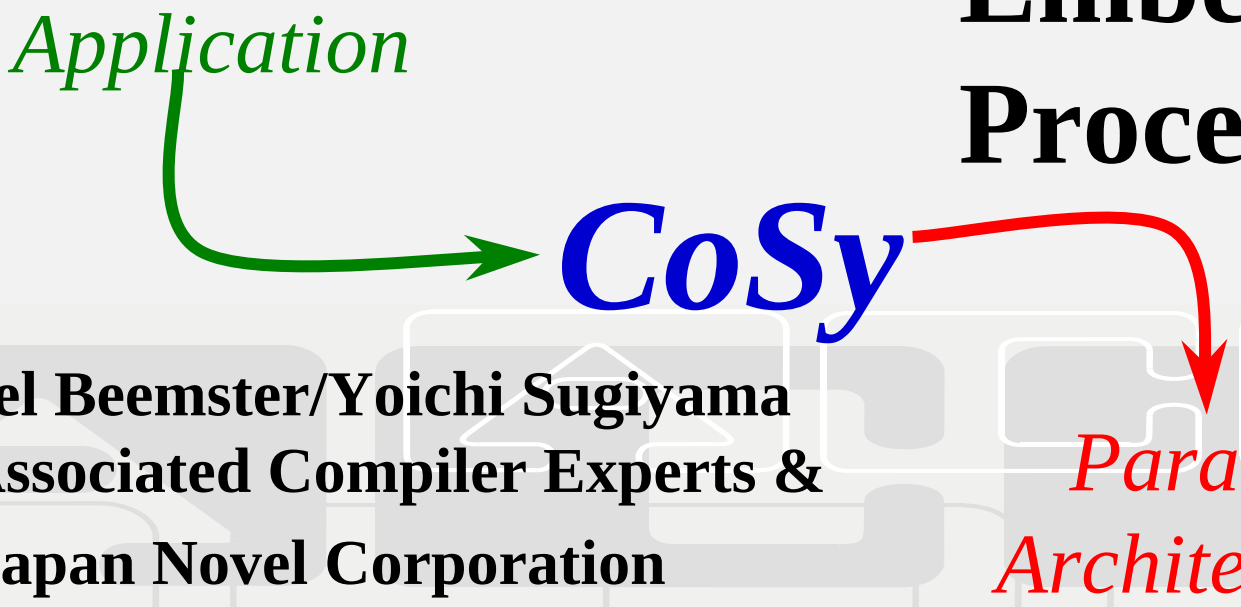
Using the CoSy Compiler Development System for Parallelism in Embedded Processors



Marcel Beemster/Yoichi Sugiyama
ACE Associated Compiler Experts &
Japan Novel Corporation
contact: `yo_sugi@jnovel.co.jp`

Motivation to use Parallelism in Embedded Systems

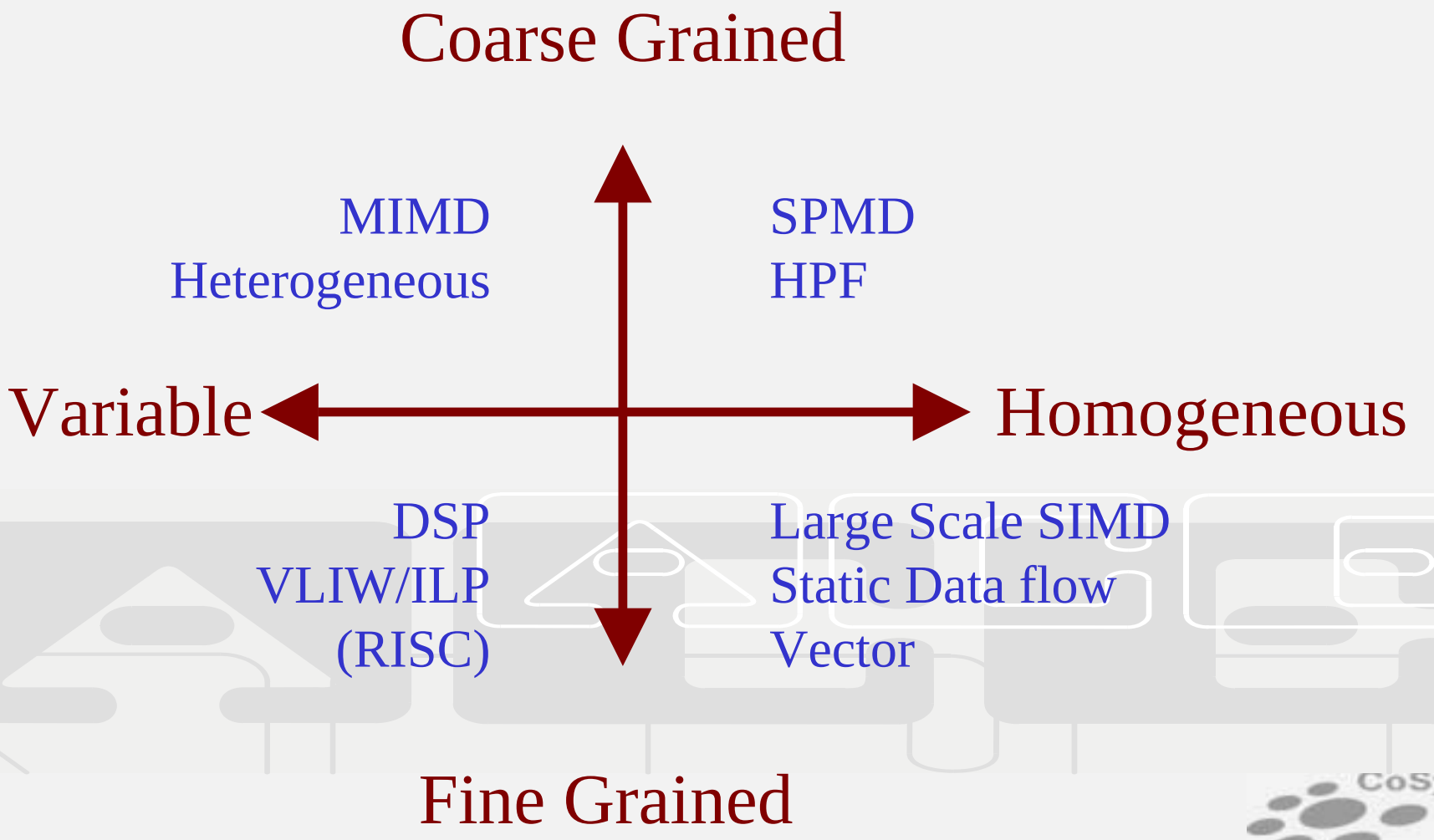
- Ever increasing demand for more compute power
- POWER consumption (battery life, cooling):
  1 processor at 1GHz uses twice the power of
  2 processors at 500MHz
- Required by using pre-existing modules
- Applications are suitable for parallel processing

- However, parallelism is never easy

Parallelism has Many Forms

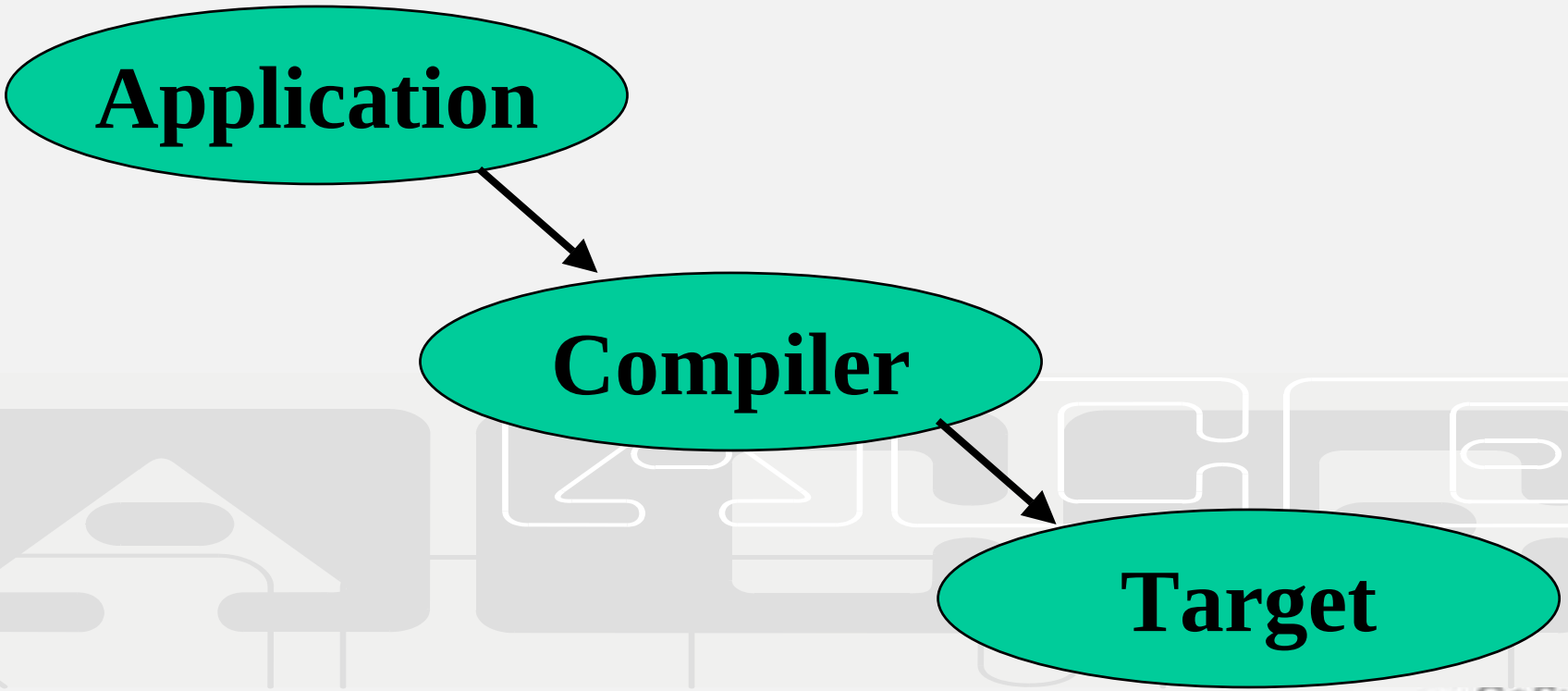
- From tightly integrated to loosely coupled
  - Pipelining, VLIW, SIMD, Vector, Static Dataflow, MIMD, etc.
- Automatic or explicit parallelization
- Type of parallelism has great impact on required tool support, in particular the compiler

Architecture Characteristics

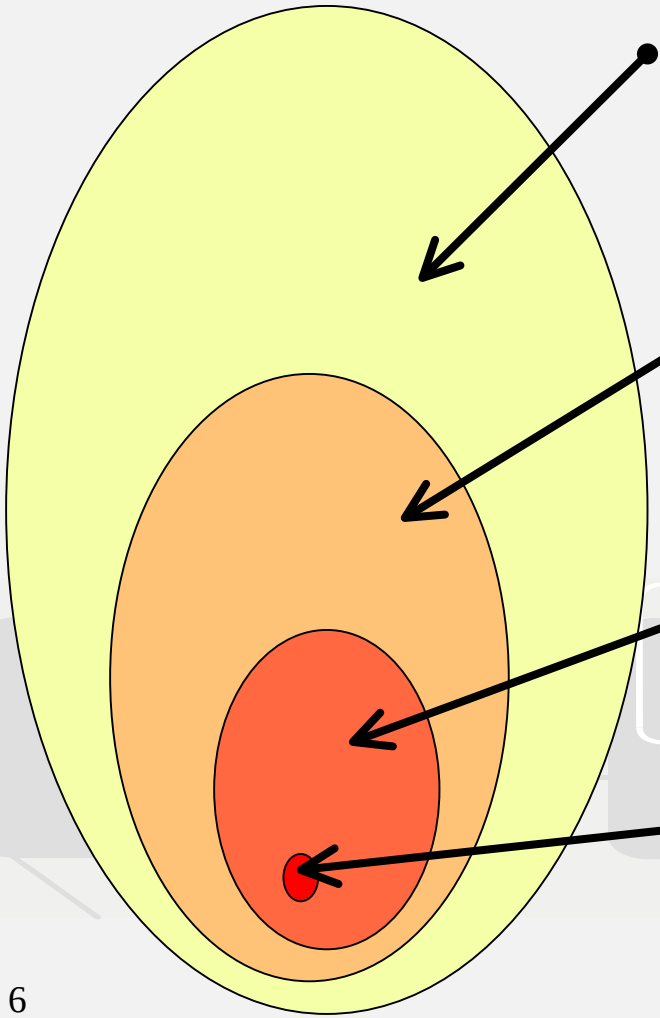


Role of the Compiler

- The compiler maps parallelism from the application to the target architecture



Automatic Parallelization is Not Always Possible



All applications

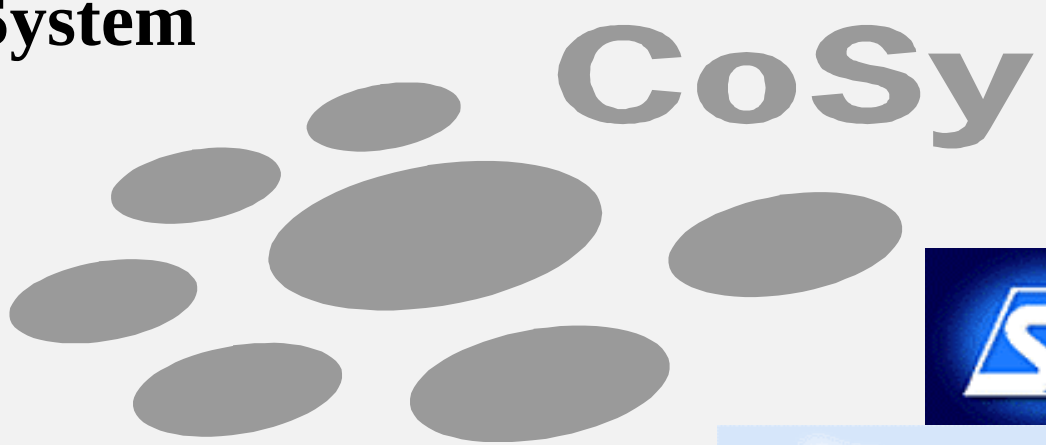
Applications that can be parallelized manually

(Legacy) applications that can be parallelized by explicit annotations

Automatically parallelizable sequential applications

CoSy is:

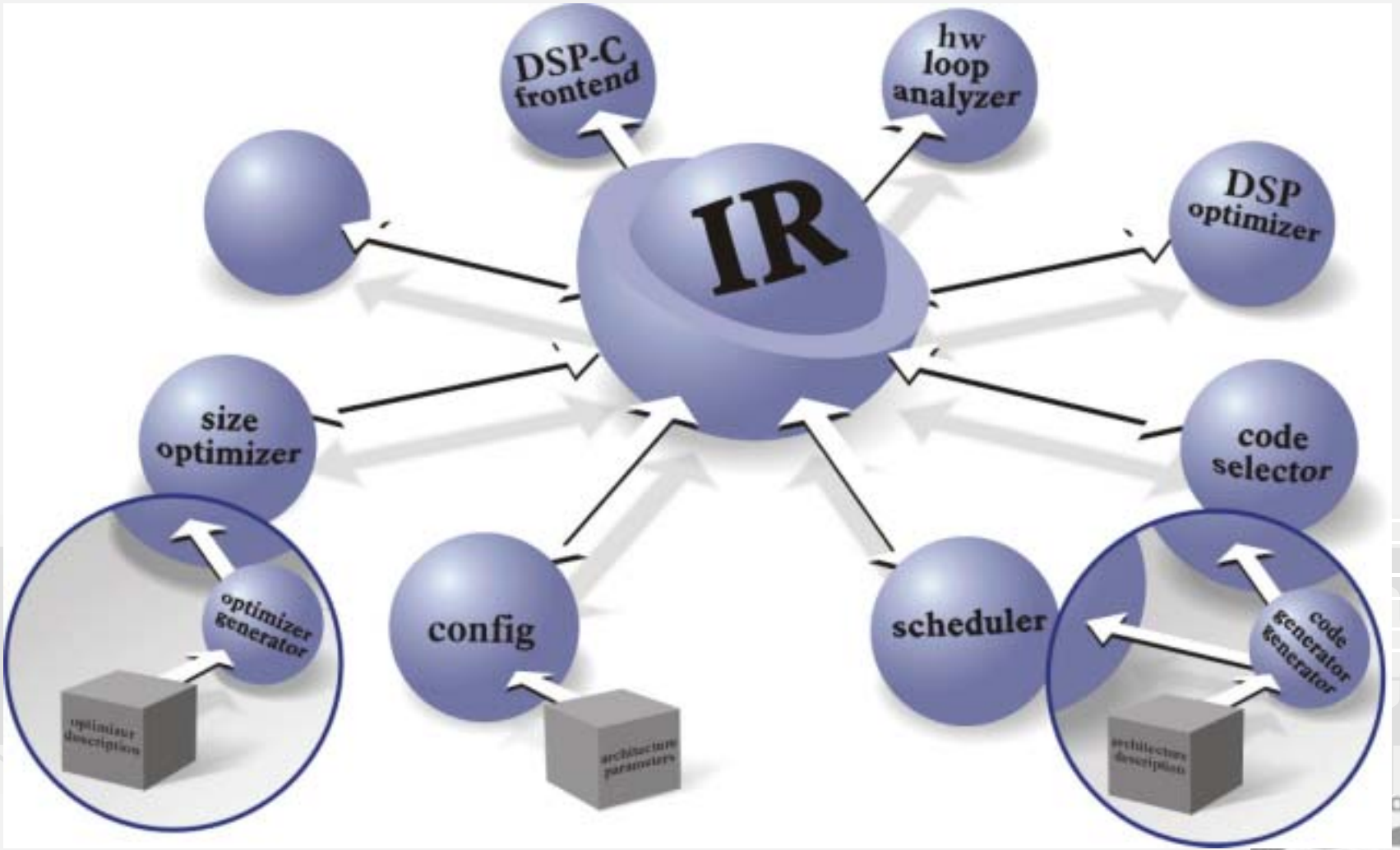
- **The world's most advanced Compiler Development System**
- **Used by major corporations world-wide**



CoSy Qualities

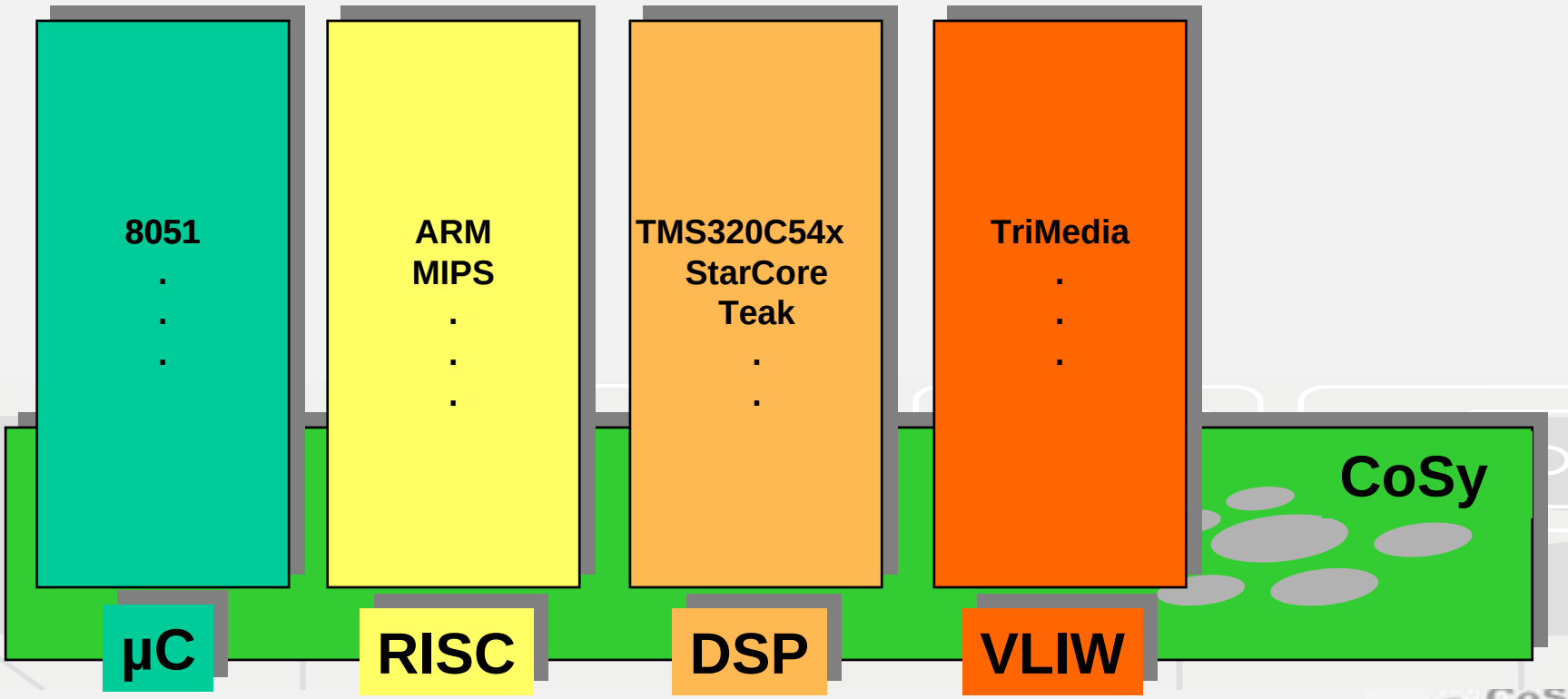
- Compiler Generator System
- Modular design
- Configurable
- Retargetable
- Robust
- Extensible
- High Quality
- Highly optimising
- Build and supported by ACE
- Supported by Japan Novel in Japan

CoSy Structure



CoSy 2003

- **CoSy is a flexible compiler development environment for any architecture**



CoSy for Pipelined RISC Architectures

- **Memory load delay filling (Scheduler)**
- **Branch delay slot filling (Scheduler)**
- **Register allocation (RegAlloc)**

CoSy for DSP Architectures

- Multiple memory loads (Scheduler)
- Post-increment addressing (Scheduler)
- Optimal usage of specialized registers (RegAlloc)
- Zero overhead loop support

CoSy for VLIW Architectures

- **Instruction packing with resource and latency model (Scheduler)**
- **Predicated execution**
- **Inlining**
- **Loop unrolling**
- **Software pipelining**

Software Pipelining Example

```
void
func(float * restrict p, float * q, float * r)
{
   int i;

   for (i = 0; i < 10; i++) {
      *p++ = *q++ * *r++;
   }
}
```

```
func:
        save    %sp,-104,%sp
        add     %g0,10,%l5
.L1:
! --- cycle 0 --------------
        ld      [%i2+0],%f0
        ld      [%i1+0],%f1
        add     %i2,4,%i2
! --- cycle 1 --------------
        add     %i1,4,%i1
! --- cycle 3 --------------
        fmuls   %f0,%f1,%f0
! --- cycle 7 --------------
        st      %f0,[%i0+0]
        add     %i0,4,%i0
        subcc   %l5,1,%l5
        bne     .L1
        nop
! --- cycle 0 --------------
        ret
```

SPARC Loop

← **Before: 8 cycles**

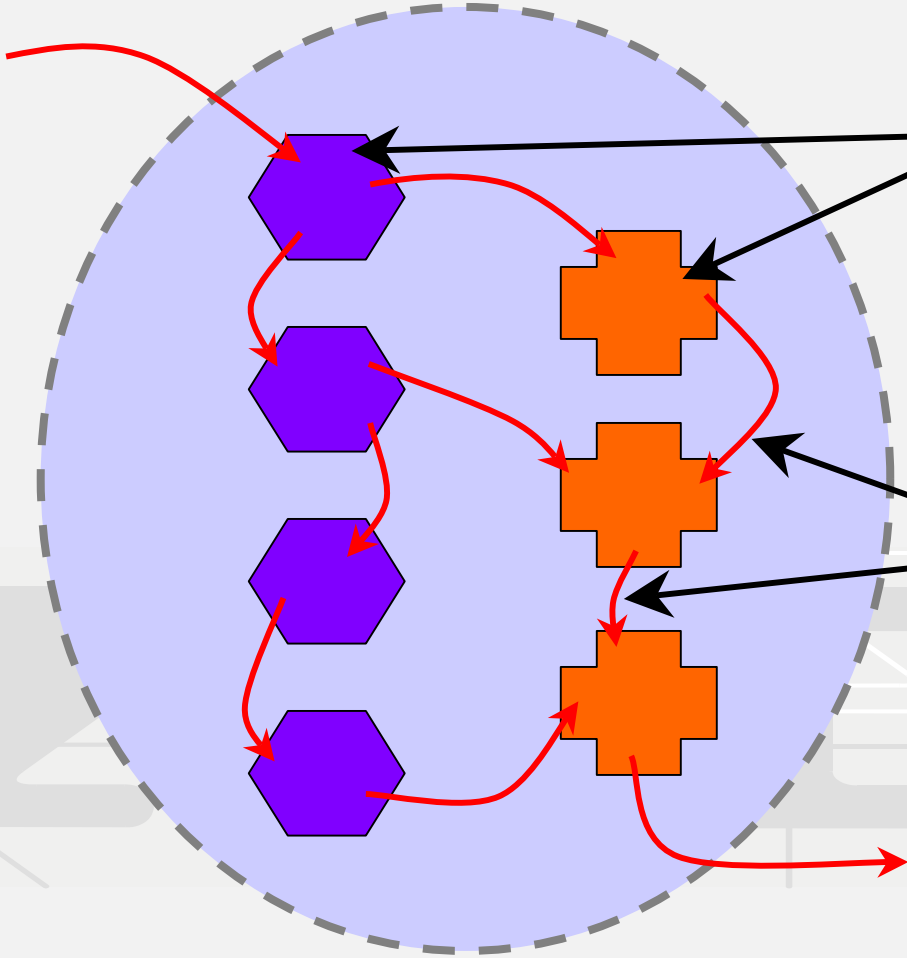
→ **After: 4 cycles**

```
func:
        save    %sp,-104,%sp
        ld      [%i1+0],%f1
        ld      [%i2+0],%f0
! --- cycle 1 --------------
        add     %i1,4,%i1
        add     %i2,4,%i2
! --- cycle 3 --------------
        fmuls   %f0,%f1,%f2
        add     %g0,9,%l5
.L1:
! --- cycle 0 --------------
        ld      [%i1+0],%f1
        ld      [%i2+0],%f0
! --- cycle 2 --------------
        add     %i1,4,%i1
        add     %i2,4,%i2
! --- cycle 3 --------------
        st      %f2,[%i0+0]
        fmuls   %f0,%f1,%f2
        add     %i0,4,%i0
        subcc   %l5,1,%l5
        bne     .L1
        nop
! --- cycle 0 --------------
        st      %f2,[%i0+0]
        add     %i0,4,%i0
        ret
```

CoSy for SIMD and Vector Processors

- Data dependence analysis
- Automatic vectorization (under development)
- Alignment analysis (for SIMD)
- Dynamic Intrinsic (compiler known functions) support with scheduling (also for FPGA)
- Compiler known types (e.g. XYZw, RGBa structures)

CoSy for Reconfigurable - Static Data Flow



Programmable computational units

Configurable communication paths

Extracting the Stream Representation for SDF

- Works on nested loop programs
- Extract the Memory Input-Output commands
- Extract Data Flow Relation
- Create a synchronous Stream program
  - Can be mapped to reconfigurable architecture
  - Can be mapped to FPGA
  - Can be mapped directly to hardware
  - And also to vector/SIMD architectures

Memory I/O Analysis

From Matrix multiply:

```
for (i=0;i<N;i++){
   for (j=0;j<N;j++){
      for (k=0;k<N;k++){
         .. = .. a2[k][j] ..;
```

Translates to:

```
StreamInStream3( (int*)a2, N, 0,
                 N, 1,
                 N, N );
```

Example Array based DCT

```
for (block = 0; block < NBLOCKS; block++) {
    for (y = 0; y < SIZE; y++) {
        for (x = 0; x < SIZE; x++) {
            Result[block][y][x] = 0;
            for (v = 0; v < SIZE; v++) {
                for (u = 0; u < SIZE; u++) {
                    int32 tmp;
                    int32 t1 = cosines[x][u];
                    int32 t2 = cosines[y][v];
                    tmp = MUL(t1, t2);
                    tmp = UNSCALE(tmp);
                    tmp = MUL(tmp, inData[block][v][u]);
                    Result[block][y][x] += tmp;
            }   }
            Result[block][y][x] = (Result[block][y][x] >> 2) + SCALE(128);
            Result[block][y][x] = UNSCALE(Result[block][y][x]);
            Result[block][y][x] = (Result[block][y][x]);
            if (Result[block][y][x] > 255)
                Result[block][y][x] = 255;
            else if (Result[block][y][x] < 0)
                Result[block][y][x] = 0;
}   }   }
```

CoSy Generated Stream code for DCT

```
in0 = new in_stream(inData, stride(4,256), stride(8,0),
             stride(8,0), stride(8,32), stride(8,4));
in1 = new in_stream(cosines, stride(4,0), stride(8,0),
               stride(8,32), stride(8,0), stride(8,4));
in2 = new in_stream(cosines, stride(4,0), stride(8,32),
                 stride(8,0), stride(8,4));
calc0 = StreamMultiply(in1, in2)
calc1 = StreamAddition(calc0, 8192)
calc2 = StreamShiftright(calc1, 14)
calc3 = StreamMultiply(in0, calc2)
calc4 = StreamAccumulate(calc3, ?)
calc5 = StreamShiftright(calc4, 2)
calc6 = StreamAddition(calc5, 2097152)
calc7 = StreamAddition(calc6, 8192)
calc8 = StreamShiftright(calc7, 14)
calc9 = StreamSatCeiling(calc8, 255)
calc10 = StreamSatFloor(calc9, 0)
StreamOutStream(calc10, Result, stride(4,256), stride(8,32),
                                                stride(8,4));
```

CoSy for SPMD architectures

- High Performance Fortran compiler front end to IR
- IR extensions for aggregate Array operations
- Generates data partitioning
- Generates communication stubs
- Generates program synchronization

CoSy for MIMD and Heterogeneous Multi-Processors

- Explicitly/pre-partitioned application
- Pragma steered compilation to multiple targets
- Unification of data models
- Emulation of missing functionality (like fixed point on RISC)
- Generation of communication stubs
- Subsuming OS functionality by Intrinsics (compiler known functions)

CoSy Express for HW/SW co-design

- **OEM product based on CoSy**
- **Pre-configured CoSy, requires data-model and code generator rules to generate compiler**
- **Includes optimizations, library instantiation, testing framework, …**

⇒Allows for very rapid compiler generation (minutes)

⇒Ideal for embedding in HW/SW evaluation environment

And CoSy Includes Much More…

- Front-ends for C-89, C-99, DSP-C, Embedded C, C++, Fortran, GNU extensions
- Dwarf2 debugging info generation
- Extensive loop optimization (with zero overhead loop support)
- Target configuration to the bit
- Emulator generation
- Example compiler and code generator to jump-start compiler development
- Calling convention and stack layout configurability
- …

ACE

- **Based in Amsterdam, the Netherlands**
- **30 years young; 30 people company**
- **Fully dedicated to the CoSy compiler development system**
- **Licenses CoSy to companies worldwide to do their own compiler development**
- **Provides CoSy *WITH* support**
- **Represented by Japan Novel in Japan**

Japan Novel and CoSy

- **Japan Novel is an exclusive agent in Japan for ACE**
- **Japan Novel provides a products and services to improve the quality of today's complex embedded software**
  - **Compiler evaluation services**
  - **Automated test&evaluation system - Quality Commander**
  - **C/C++ comformance test suites - PlumHall's products**
- **Compiler evaluation services provide a thorough testing of C/C++, Embedded－C, DSP-C compilers**
- **With it's high reliability, CoSy compiler development system contributes to the embedded system development in Japan**

Go Parallel with CoSy!

ACE Associated Compiler Experts

Home of CoSy

the Compiler Development System

`yo_sugi@jnovel.co.jp/marcel@ace.nl`

What you get ‘out-of-the-box’

- The CoSy Compiler Development System
  - including many optimizations
  - with code generator generator
- Example Compilers and Techniques
- SuperTest C/C++ Test and Validation Suite
- Standard C Libraries
- CADESE Version Management System
- CoSy Support Program