# 3L Diamond

Multiprocessor DSP

RTOS

# What is 3L Diamond?

- Diamond is an operating system designed for multiprocessor DSP applications.
- With Diamond you develop efficient applications that use networks of DSPs connected by point-to-point links.
- Your system can have any number of processors.

# What is 3L Diamond?

- Diamond uses a very simple but powerful API to give you efficient:
  - multi-tasking
  - multi-threading
  - link communication
  - host communication
  - semaphores
  - timer control
  - … and much more

# What do I get?

Diamond includes:

- microkernels for selected DSP boards;
- a host server program for I/O and control;
- a full ANSI C library;
- utilities for building applications.

- The DSP manufacturer’s compiler is used

# What's the microkernel?

- The microkernel is a small piece of code that is placed on each processor to support:
  - tasks
  - threads
  - priority-based pre-emptive scheduling
  - interrupt handling
  - semaphores & events
  - timers
  - … and much more

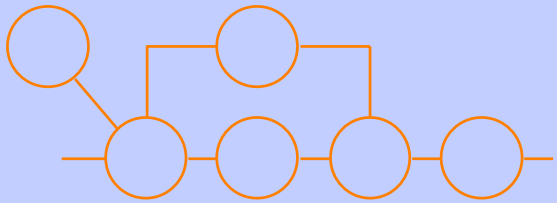
# What's the overhead?

- Very little. For example, on the C6000:
  - microkernel code: < 12KB
  - microkernel data: < 3KB
  - context switch times (thread_deschedule):
    - SMT374 (225MHz C6713) ~470ns.
    - SMT361 (400MHz C6415) ~250ns
    - SMT395 (1GHz C6416T) ~100ns
- If you don't call the kernel it doesn't use any CPU cycles

# How does it all work?

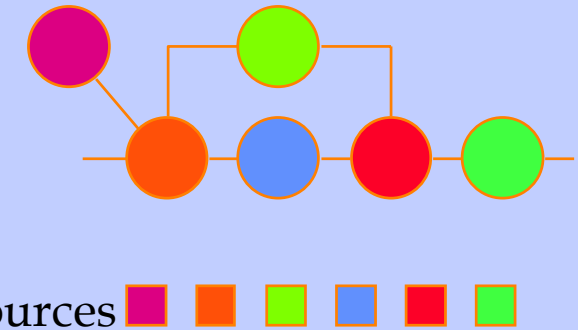
- You write complete C or assembler programs that take in data, do some processing, and send data out. These programs are known as TASKS.
- You build your application by joining tasks together so that they can communicate.
- Processors are loaded with only the code they need.

# How do I use 3L Diamond?
## Start with your block diagram, ...



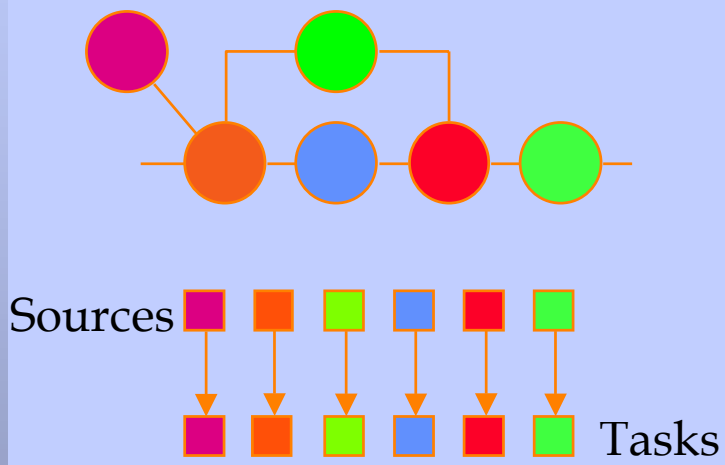
- Block diagram description of your application
- Don’t worry about processors at this stage.

# ...code each block independently, ...



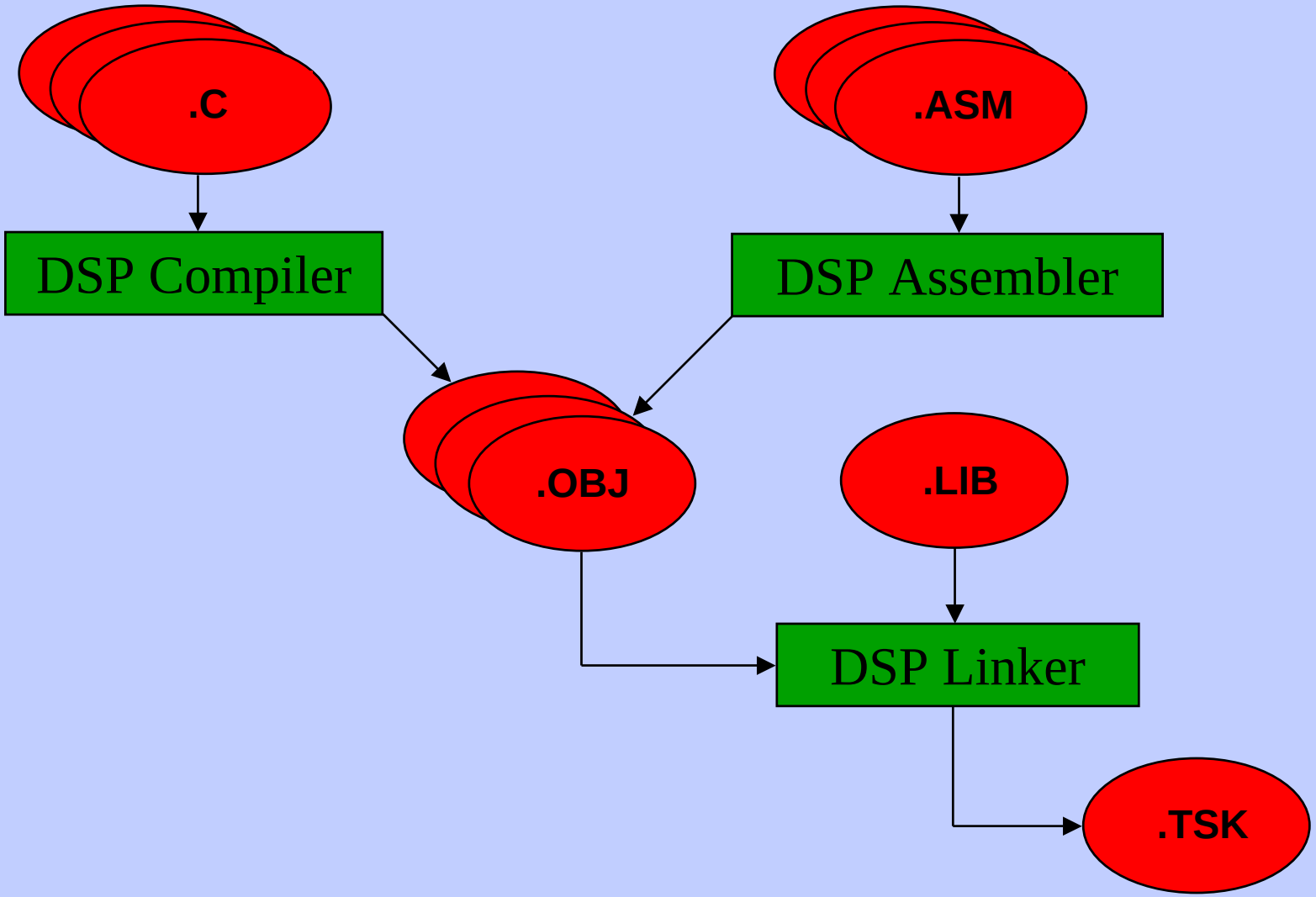

- Code each block as a complete C/asm program
- Include references to Diamond libraries and application libraries

# and build each task.




- For *each* block:
  - Compile or Assemble
  - Link with Diamond run-time libraries (and other libraries) to create a relocatable task.
- The tools run on your PC.

# This is how you create each task:



# How do tasks communicate?
# They are connected by channels.

- A channel lets you send data from one task to another, wherever that task may be.
- Each task when it starts is given a list of channels providing input and a list of channels accepting output.
- These lists appear as arguments to the task's **`main`** function.

# How do I join my tasks together?
## You use the 3L configurer...

- The configurer combines your tasks into a single application file.
- You control the configurer with data in a textual configuration file.
- The configuration file names the tasks, shows how to connect them, and says where to place them when you run the application.

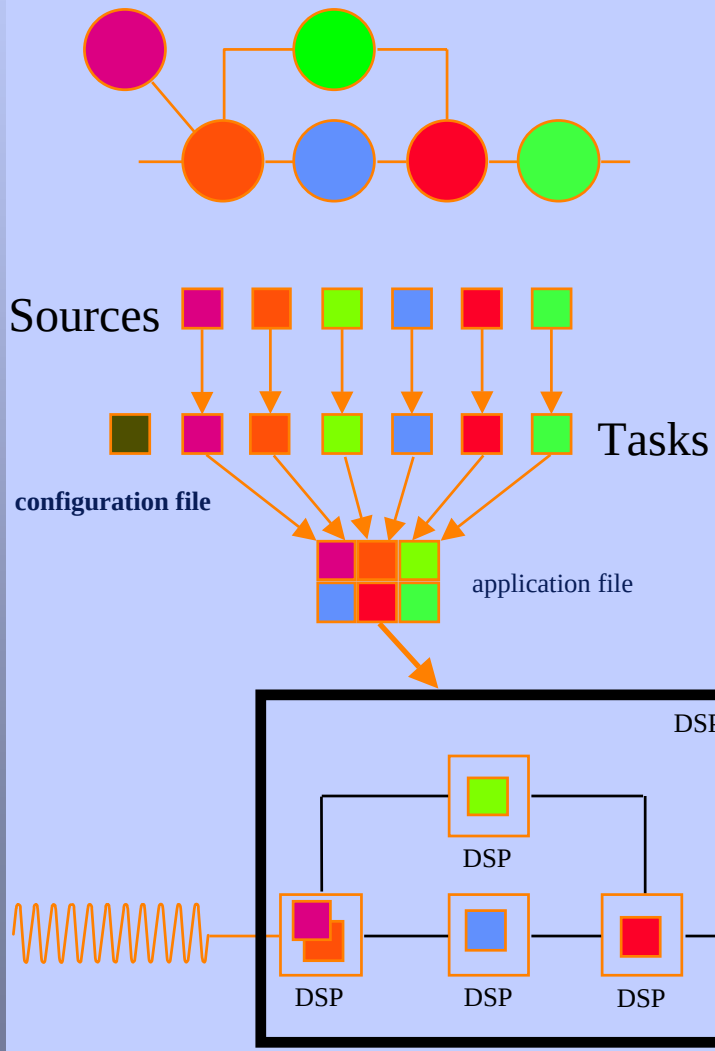
# What does the configurer do?

- The configurer maps your channels onto the links connecting processors where necessary.
- It builds a single application file that contains everything needed to load the DSP network and get your application going when you execute it later.

# What about memory?

- Processor types give the configurer all the information it needs about particular DSPs.
- The configurer usually allocates memory automatically.
- If you really want to, you can control the allocation of memory explicitly.

# Configuration

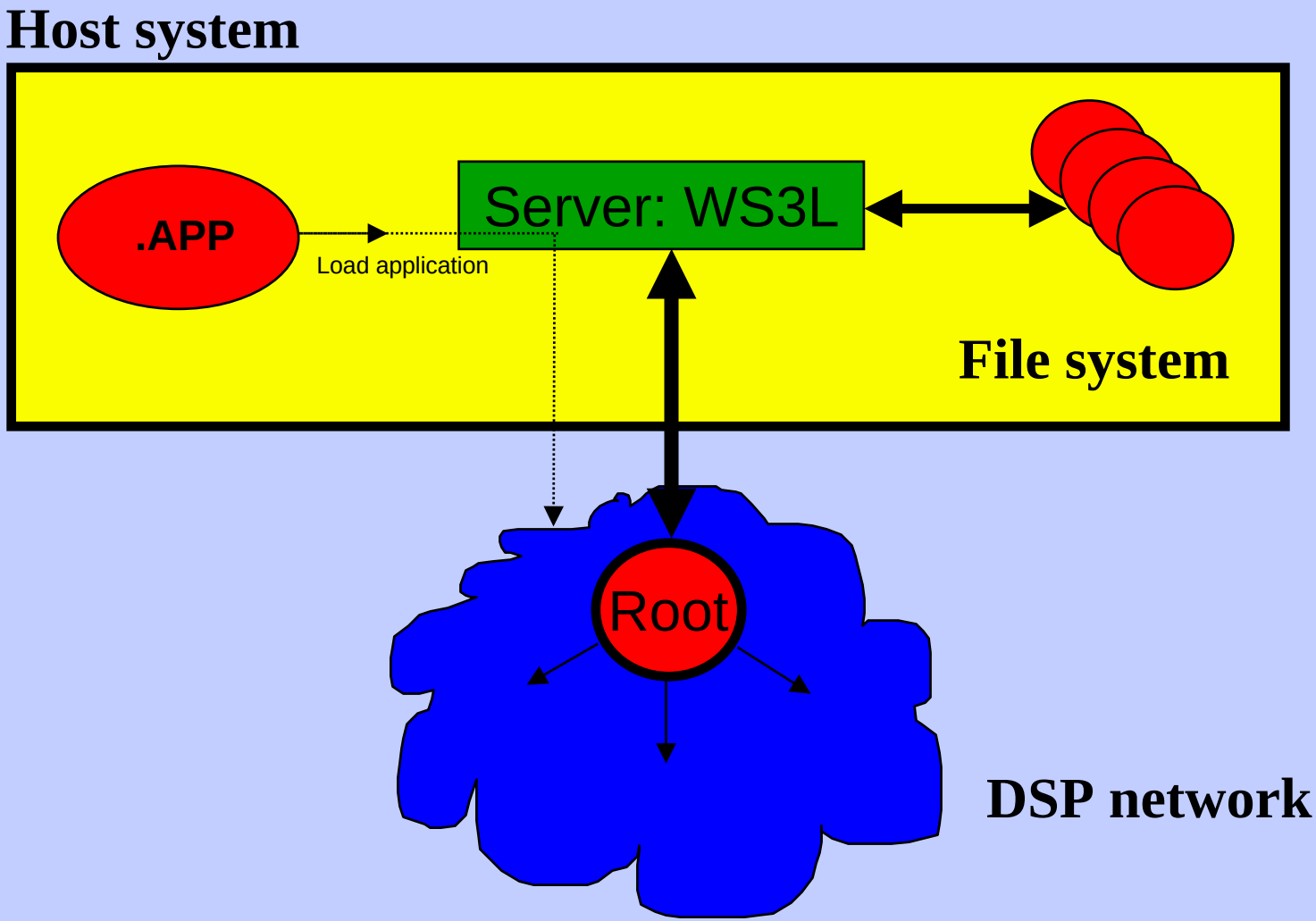



- Configure Application
  - Configuration file
    - Define software: tasks and connections
    - Define hardware: DSPs and links
    - Map software to hardware
  - Generate an application file ready for running.

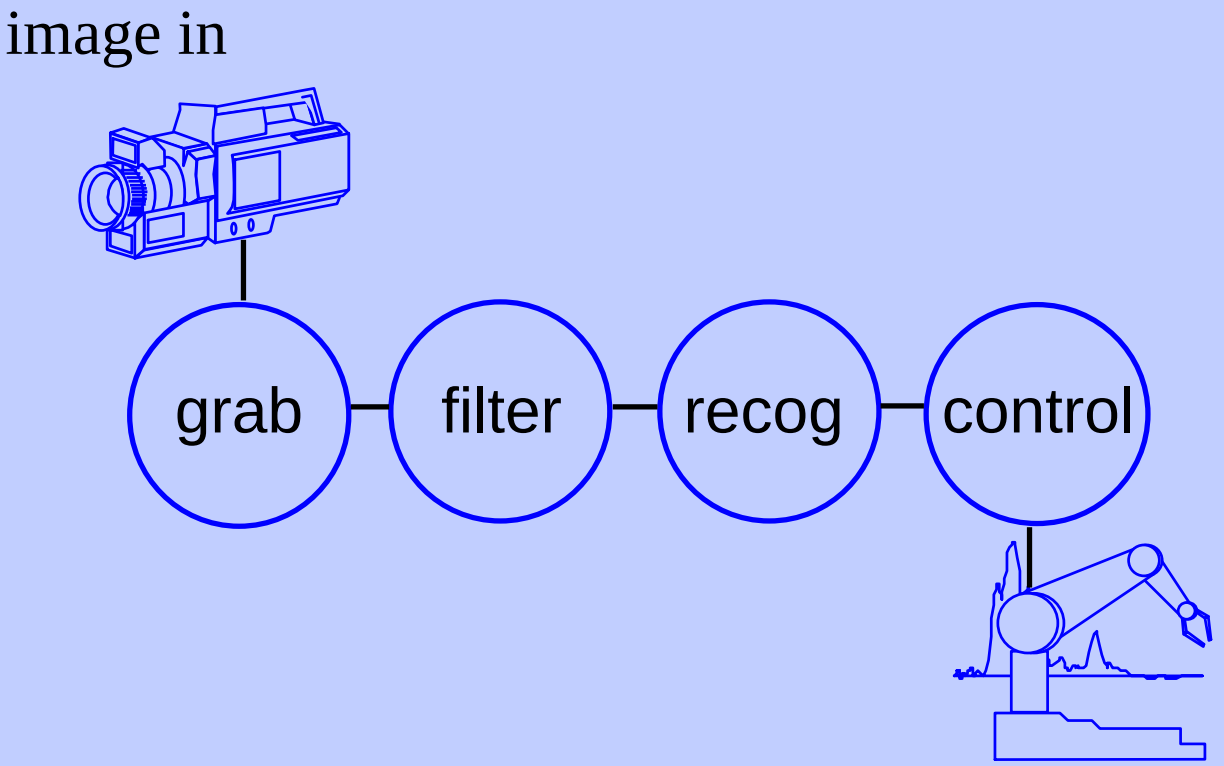
# How do I run my application?
# You use the host server program: WS3L.

- The server is a program that runs in the host computer connected to your network of DSPs.
- It sends your application into the DSP network and provides host input/output services.
- It communicates directly with one of the DSPs: the ROOT processor.

# Running applications

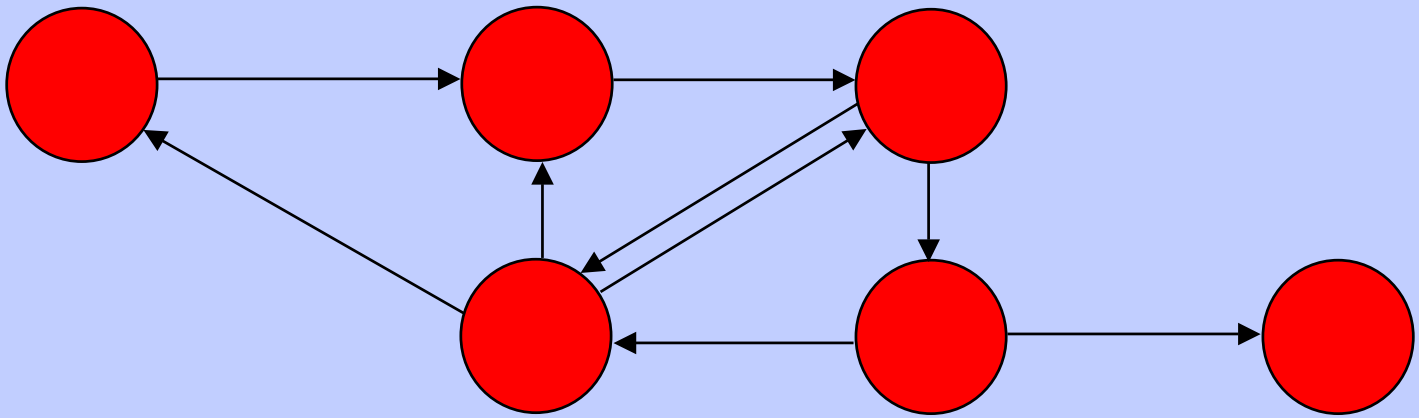


# What about an example?
## Here's a simple block diagram.



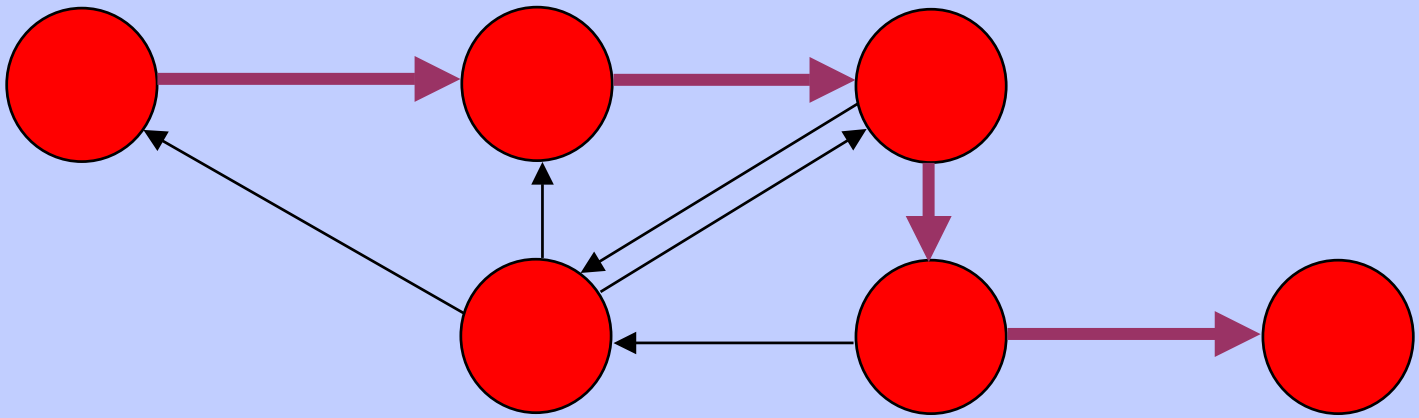
# That's a pipeline. Is that all I can use? **You can use any structure you wish.**

- Tasks may have as many input channels and output channels as you like, for example:



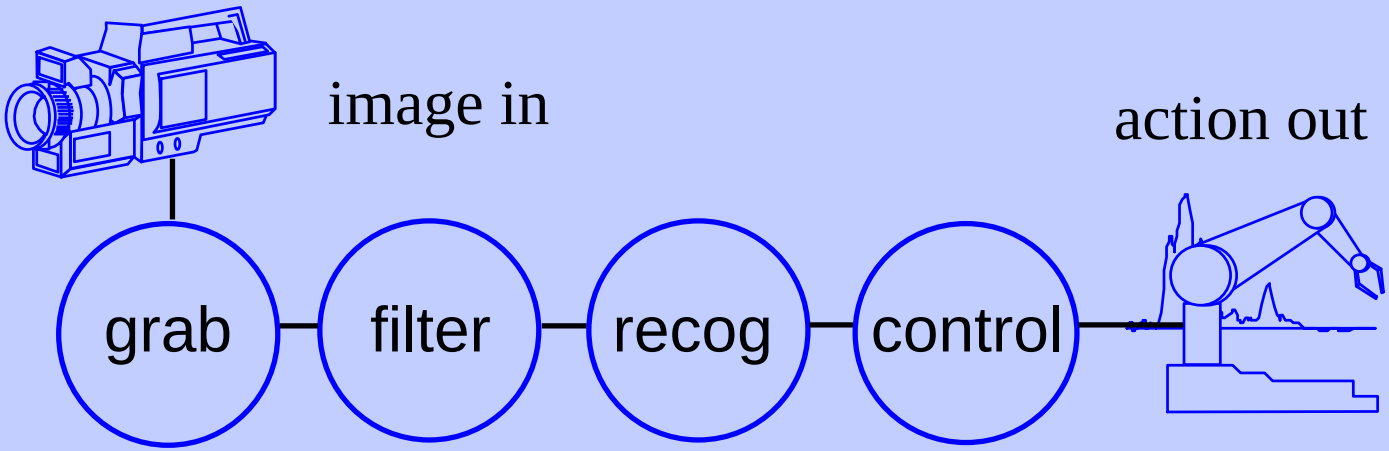
# You can use any structure you wish.

- Diamond will automatically route messages from one processor to any other along the links.
- The routing is guaranteed deadlock-free.



# Back to the example ...

- Take a simple application.
- Describe it as a block diagram.




- Each block will become a task.

# Code each block (task), e.g., filter, ...

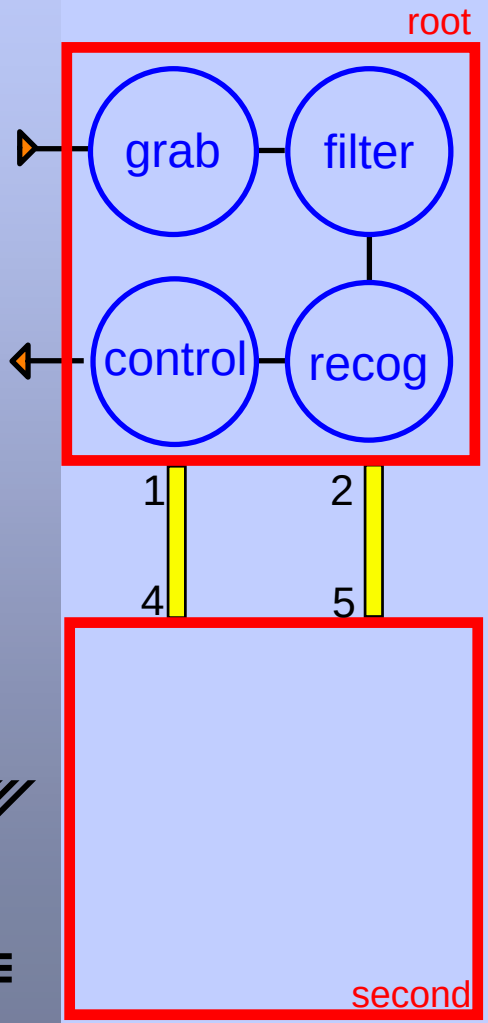
```
#include <chan.h>          // Diamond communication functions
#include <imagelib.h>      // your image processing functions
#define N 65536            // image size

static char bits[N];       // to hold the image

void main(int argc, char *argv[], char *envp[],
          CHAN *in[],  int ins,
          CHAN *out[], int outs)

// in and out are the lists of channels that connect this task to
// other tasks
{
    for(;;) {
        chan_in_message(N,  bits,  in[0]); // receive image
        filter(x, N);                      // process image
        chan_out_message(N, bits, out[0]); // send new image
    }
}
```

# Build each task on your PC ...

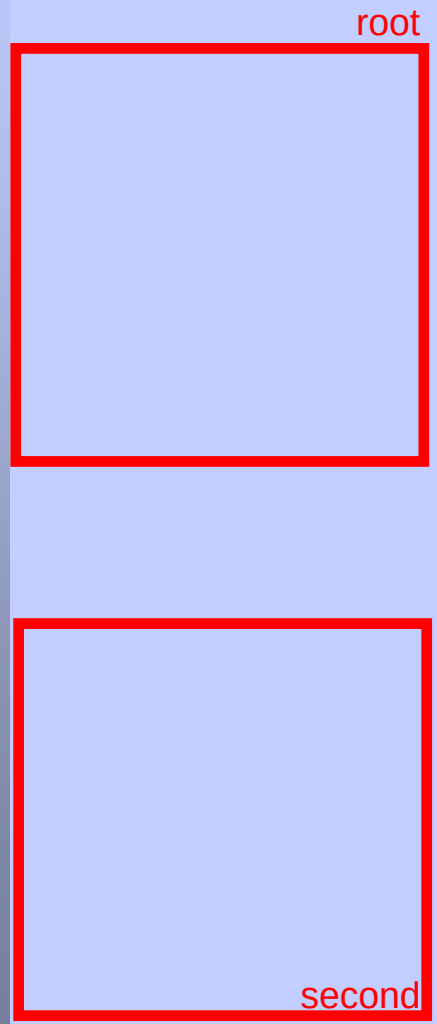
- C>C6xc grab
- C>C6xTask grab
- C>C6xc filter
- C>C6xTask filter
- C>C6xc recog
- C>C6xTask recog
- C>C6xc control
- C>C6xTask control
- C>REM makefiles are useful for this

# Can I put all tasks on a single DSP?

## Yes. Write a configuration file that ...

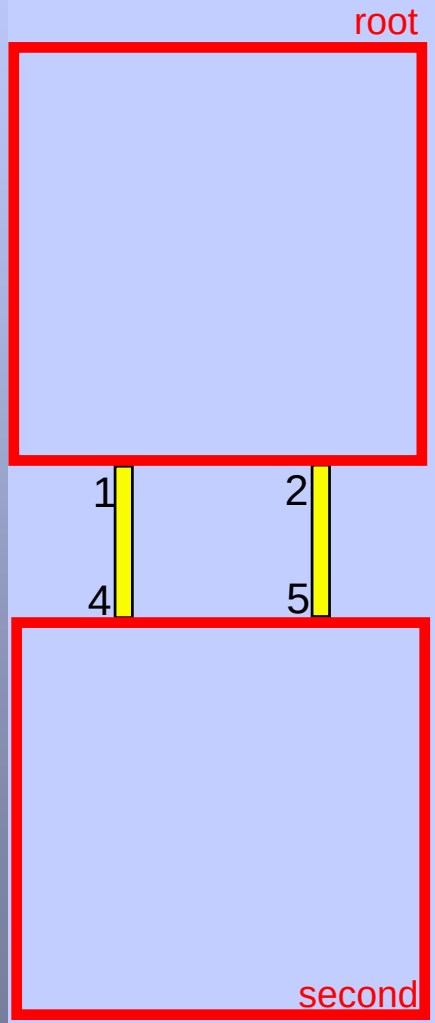



# ...names processors and gives their types ...



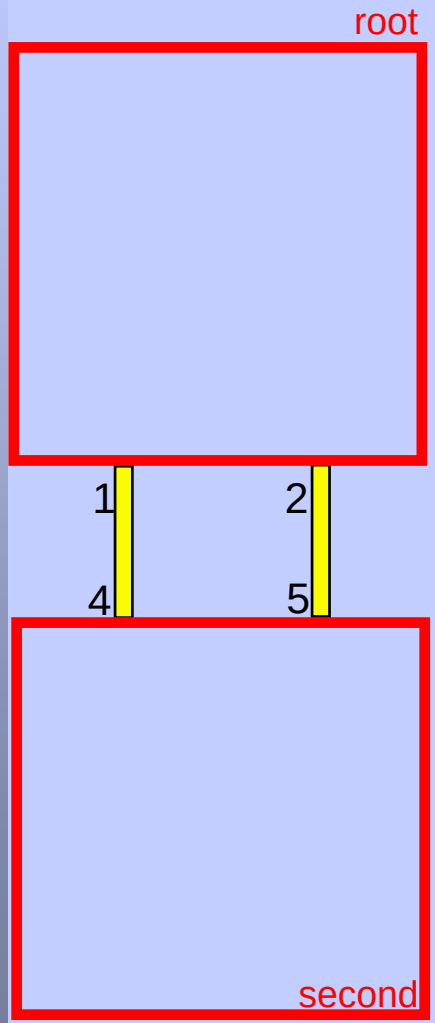

```
! Hardware description
processor root   SMT361
processor second SMT361
```

# ...describes the link connections, ...

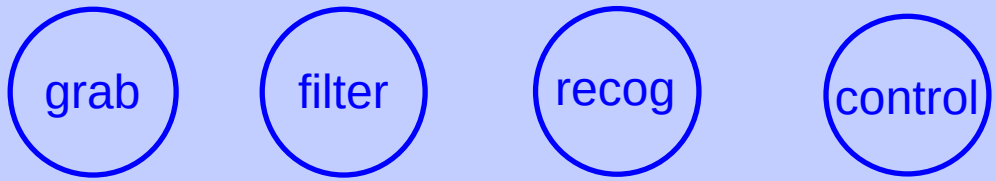



```
! Hardware description
processor root   SMT361
processor second SMT361
wire ? root[1] second[4]
wire ? root[2] second[5]
```

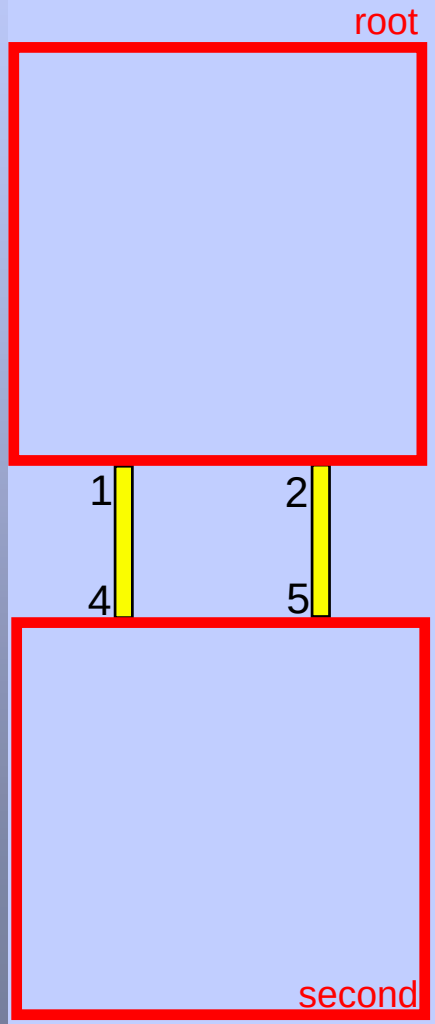
# ...describes your tasks, ...



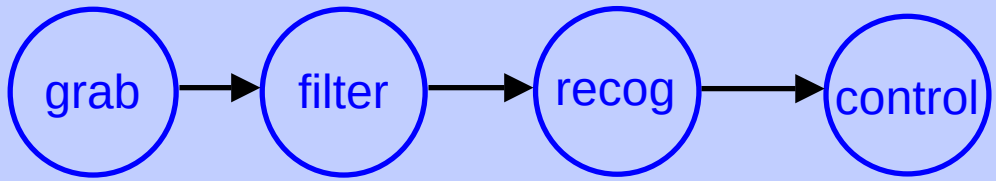

```
! Hardware description
processor root   SMT361
processor second SMT361
wire ? root[1] second[4]
wire ? root[2] second[5]

! Task declarations
task grab          ins=1 outs=1 stack=2k
task filter        ins=1 outs=1 stack=3k
task recog         ins=1 outs=1 stack=3k
task control       ins=1 outs=1 stack=2k
```



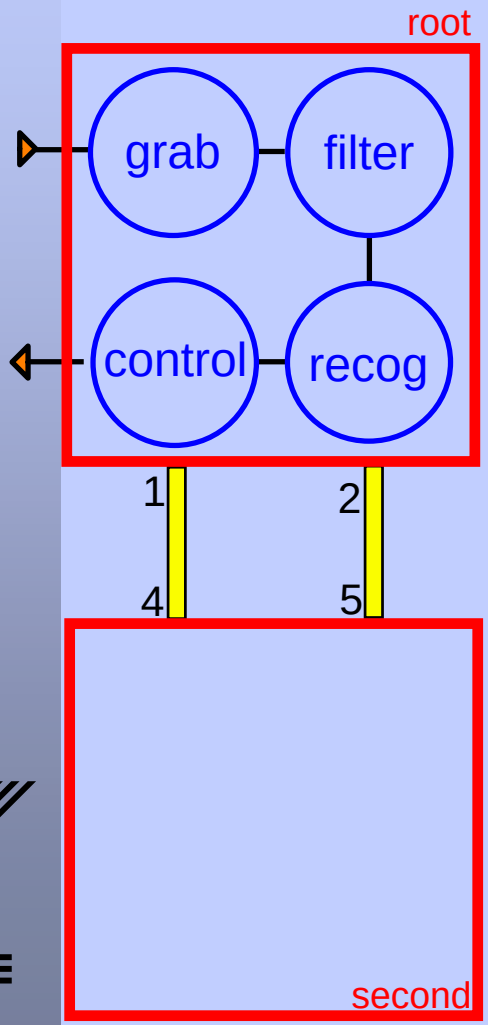
# ...connects the tasks together, ...




```
! Hardware description
processor root   SMT361
processor second SMT361
wire ? root[1] second[4]
wire ? root[2] second[5]

! Task declarations
task grab          ins=1 outs=1 stack=2k
task filter        ins=1 outs=1 stack=3k
task recog         ins=1 outs=1 stack=3k
task control       ins=1 outs=1 stack=2k

! Set up the channels between the tasks.
connect ? grab[0]   filter[0]
connect ? filter[0] recog[0]
connect ? recog[0]  control[0]
```



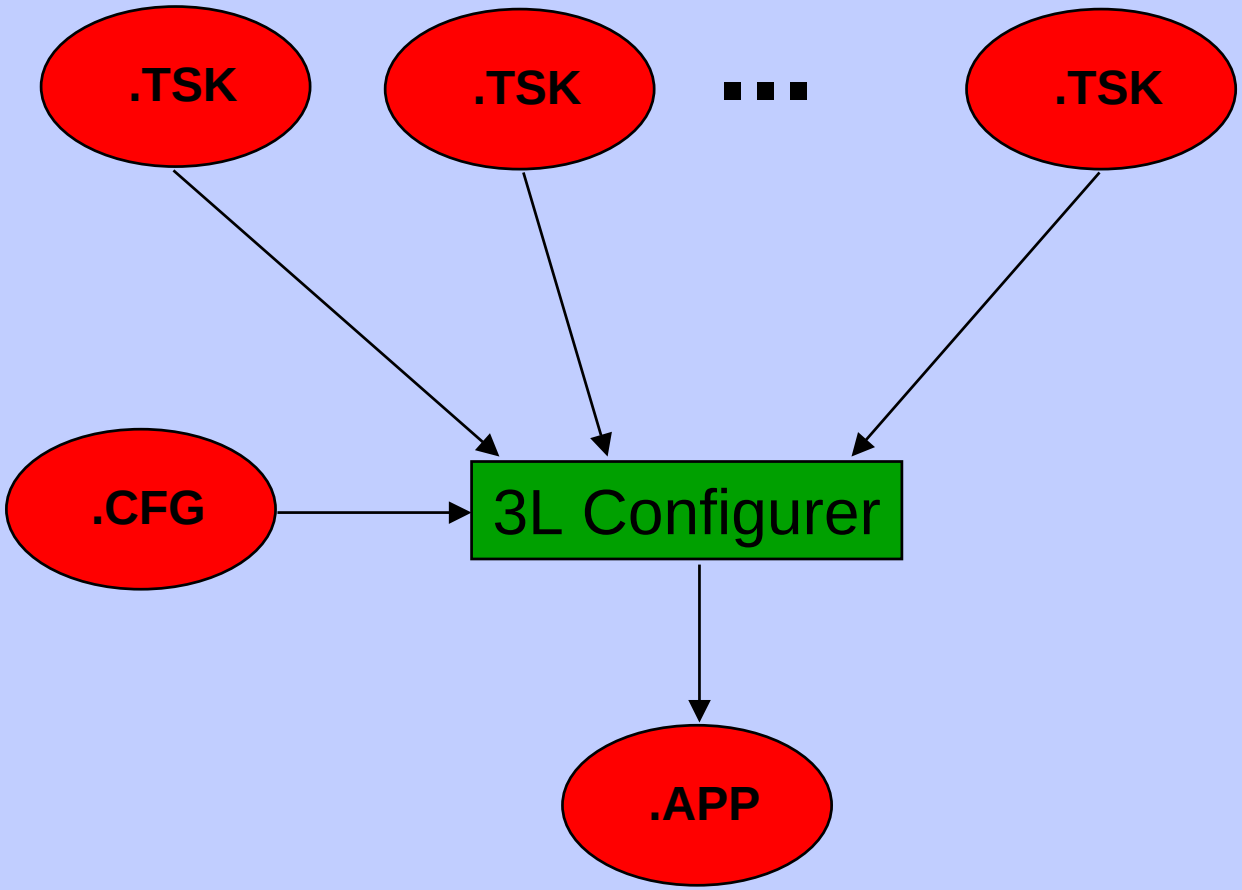
# ...and places all your tasks on one processor.




```
! Hardware description
processor root   SMT361
processor second SMT361
wire ? root[1] second[4]
wire ? root[2] second[5]

! Task declarations
task grab          ins=1 outs=1 stack=2k
task filter        ins=1 outs=1 stack=3k
task recog         ins=1 outs=1 stack=3k
task control       ins=1 outs=1 stack=2k

! Set up the channels between the tasks.
connect ? grab[0]   filter[0]
connect ? filter[0] recog[0]
connect ? recog[0]  control[0]

! Assign tasks to the available processors
place grab         root
place filter       root
place recog        root
place control      root
```

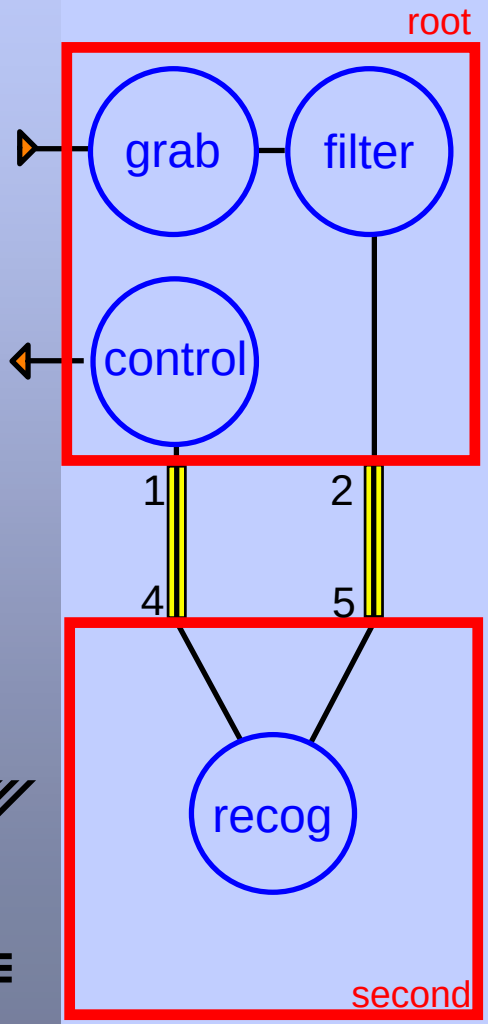
# How do I use the configuration file?
## Easy, give a command like this:

**C> config example.cfg example.app**



# How can I use the second processor?

## Just change one placement and reconfigure...




```
! Hardware description
processor root   SMT361
processor second SMT361
wire ? root[1] second[4]
wire ? root[2] second[5]

! Task declarations
task grab          ins=1 outs=1 stack=2k
task filter        ins=1 outs=1 stack=3k
task recog         ins=1 outs=1 stack=3k
task control       ins=1 outs=1 stack=2k

! Set up the channels between the tasks.
connect ? grab[0]   filter[0]
connect ? filter[0] recog[0]
connect ? recog[0]  control[0]

! Assign tasks to the available processors
place grab         root
place filter       root
place recog        second
place control      root
```

# Don’t I have to recompile or relink?
# No. Just run the configurer again.

- Experimenting with moving tasks around the network is easy: just change PLACE statements in the configuration file, reconfigure, and then run your modified application.

- Don’t worry about getting data from one task to another, even when you move tasks. Connect their channels and Diamond will handle the communication wherever the tasks are in the network.

# But I need low-level access to the DSP... **You've got it.**

- Diamond makes very efficient use of the DSP hardware: links, DMA channels, interrupts,...
- If you need to control the hardware directly (e.g., use the advanced features of the DMA engines or handle special interrupts) you can use the Diamond low-level library functions, or even write those components in assembler.

# How do I debug my application?
## Use the standard debugger.

- Diamond applications are compatible with the DSP manufacturer's debugger.
- Remember, you can also put **`printf`** statements into any task on any processor.

# What's coming next?

- Diamond is being ported to Power PCs embedded in Xilinx FPGAs.
- These processors with their Rocket I/O links fit perfectly into the Diamond model.
- You will be able to build heterogeneous networks with FPGA PPCs and other DSPs having Rocket I/O links.
- The combination of FPGA logic with CPU processing power will be formidable.

# 3L Diamond

Multiprocessor DSP

RTOS