Welcome to NCLUG

18.January.2005

Embedded Linux
without an
Embedded Target

T.Michael Turney
tmike@recipes4linux.com

What's My Line

- Contestant number 1
  - Hi, my name is T.Michael Turney and I am an Open Source Ambassador®
- Contestant number 2
  - Hi, my name is…
- Our panel has a question…
  - What is an Open Source Ambassador®?
- Contestant number 1
  - I'm glad you asked…

Open Source Ambassador®

- Believes in the philosophy of Open Source Software
- Understands that any software engineering project can benefit from being exposed to OSS
- Possesses the skills to clear most hurdles early in the project chronology
- Desires to evangelize the message

My Linux Pedigree

- October 2000
  - Hired by Monta Vista Software as an FAE to help sell HardHat Linux
- May 2002
  - Authored App-Note on kernel debugging utilizing the Abatron BDI-2000 JTAG unit
- August 2002
  - Start working on first project as an embedded Linux consultant

Current Open Source Projects

- DDD customization to form tighter integration with BDI-2000 JTAG hardware debugger (DDD-2000)
- Maintainer of www.recipes4linux.com
- X86 IPMI-enabled kernel with ipmitool on Dell hardware

Welcome to an OSS Case-Study

- Today we will go through a design roundtable, featuring the engineering requirements of the Hypothetical Part-Stamping Machine.
- As we step through the design and implementation process, we will take breaks and demonstrate the concepts utilizing Open Source tools and software.

The Story (so far…)

- Monday morning 9:00am
  - Our hero arrives at work still thinking about his girlfriend and her ultimatum : more time with me or else…
  - On his desk is a note to see his boss, urgently!
  - Behind closed doors, his boss explains the sorry state of the economy and what is coming down the pipe…

Our Story Continues…

- Monday morning in the boss' office
  - The company is losing too much money
  - Projects are being cut
  - Concentrating on core expertise
  - Layoffs are certain
  - Upper management has decided upon a contest

The Contest

- Monday morning in the boss' office
  - A new design is about to begin, taking the company in new directions
  - Each design team has one week to present their solutions to the next great Part-Stamping machine
  - The chosen design will be implemented, with all other teams receiving pink slips

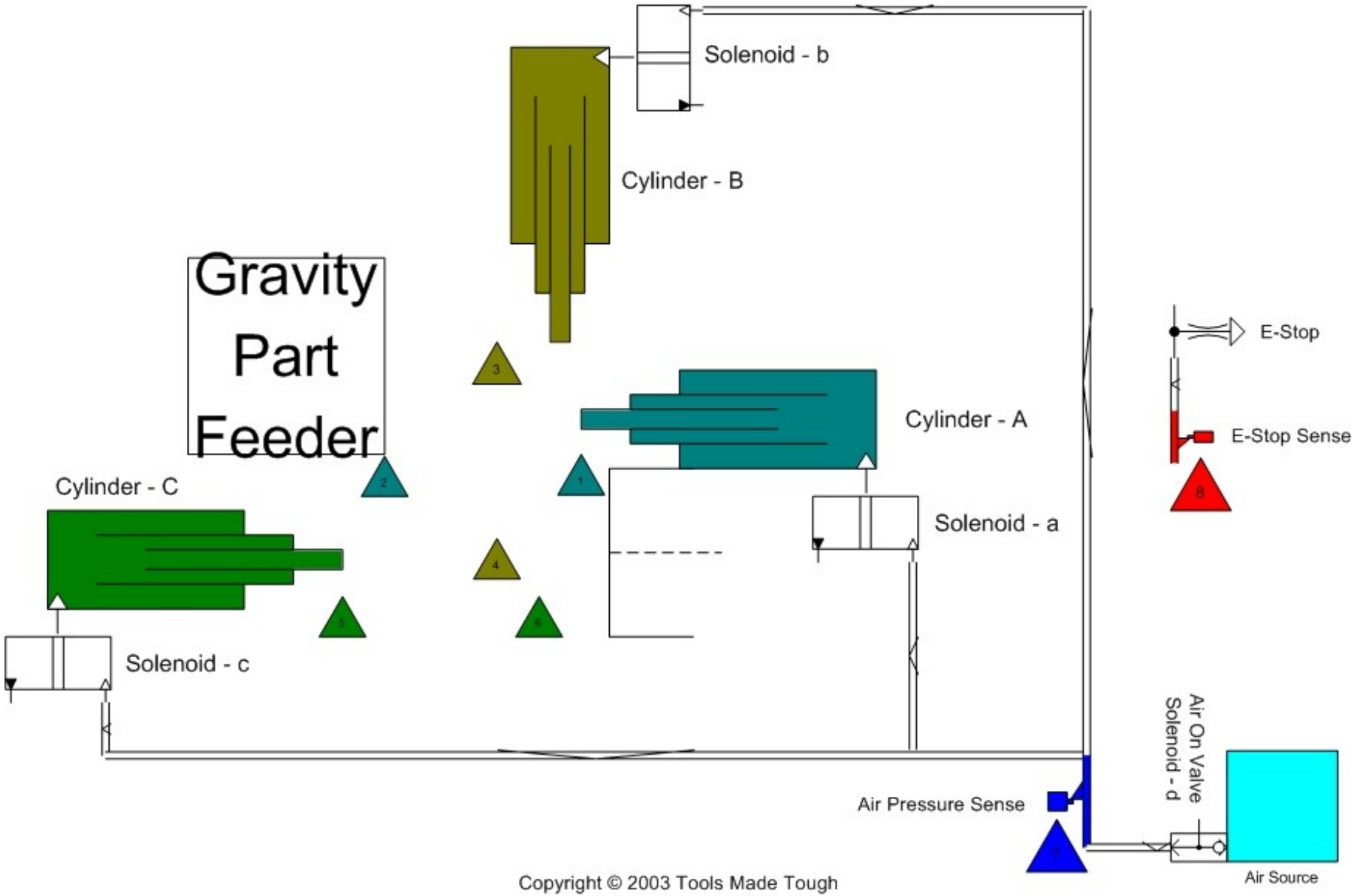
The Dilemna

- Monday at Lunch
  - Our hero has much to think about, how can he save his relationship and his job in the same week?
  - It's a good thing our hero has just returned from a week at Comdex where his head has been filled with all the latest gizmos and gadgets

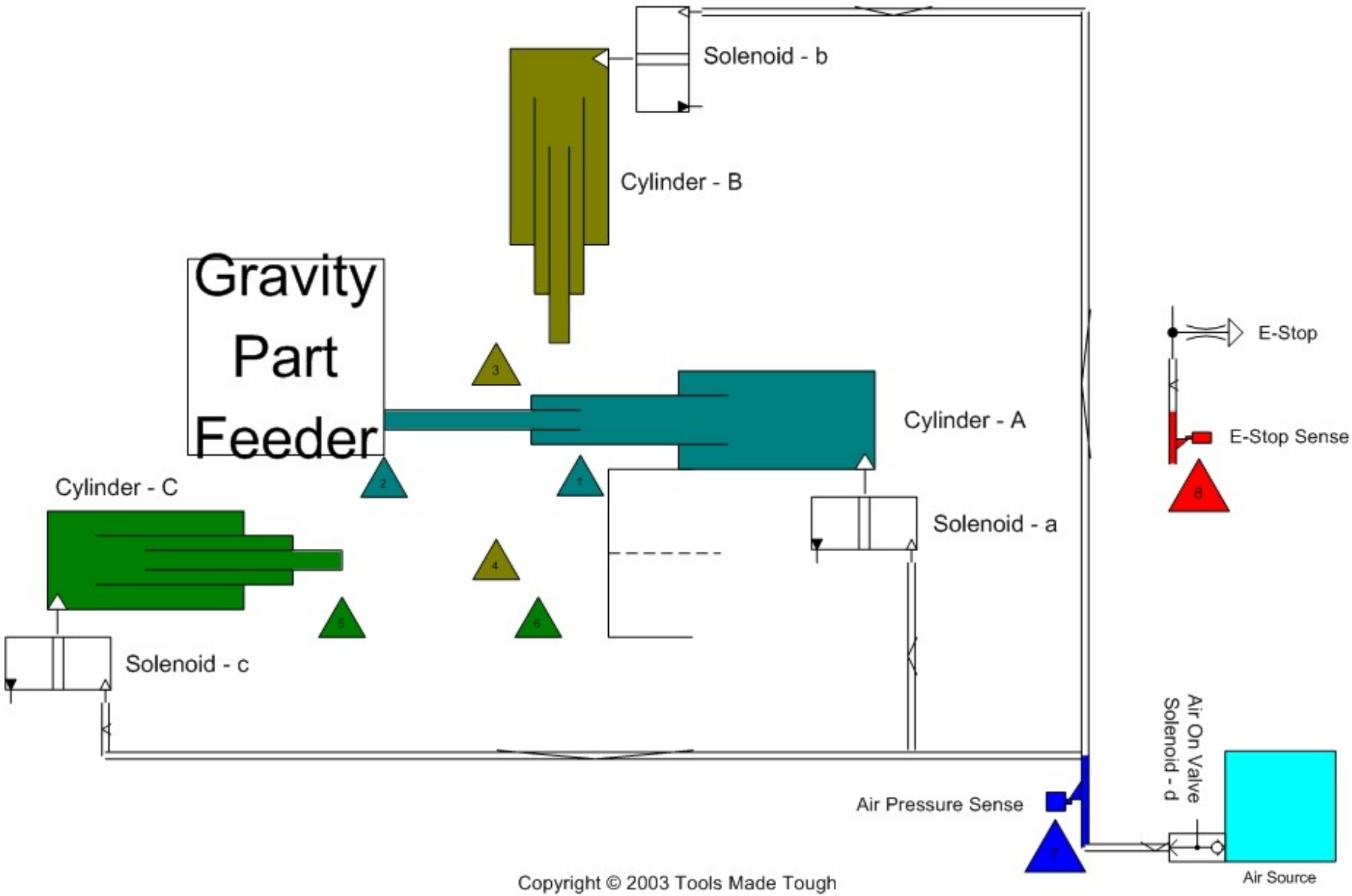
The Design

- Monday afternoon
  - Our hero sets about designing his solution to the Hypothetical Part-Stamping Machine.
  - While our hero is hard at work, what are some of your ideas on how to start this project?

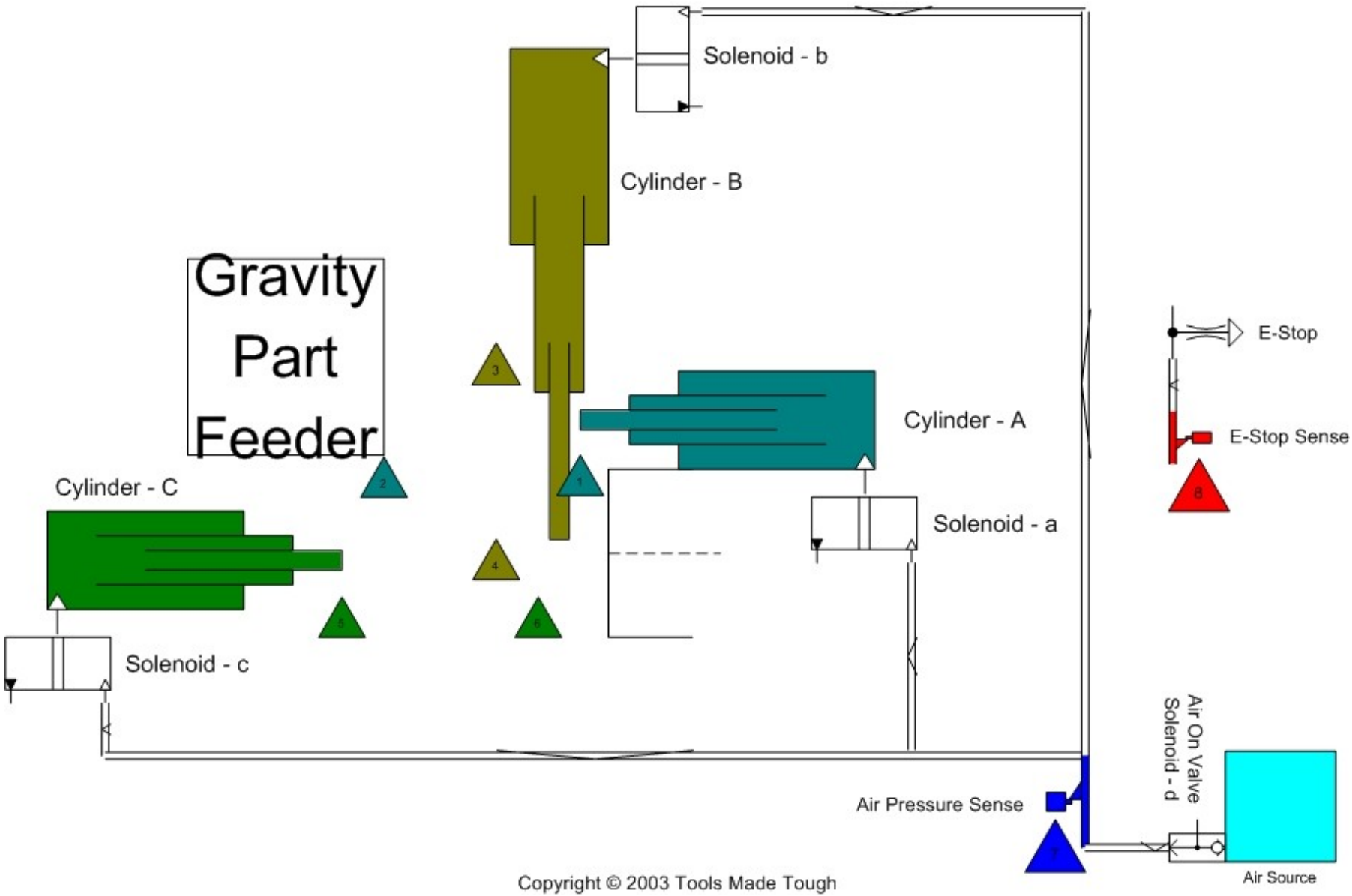
Hypothetical Part Stamping Machine
Design Diagram
Machine Control



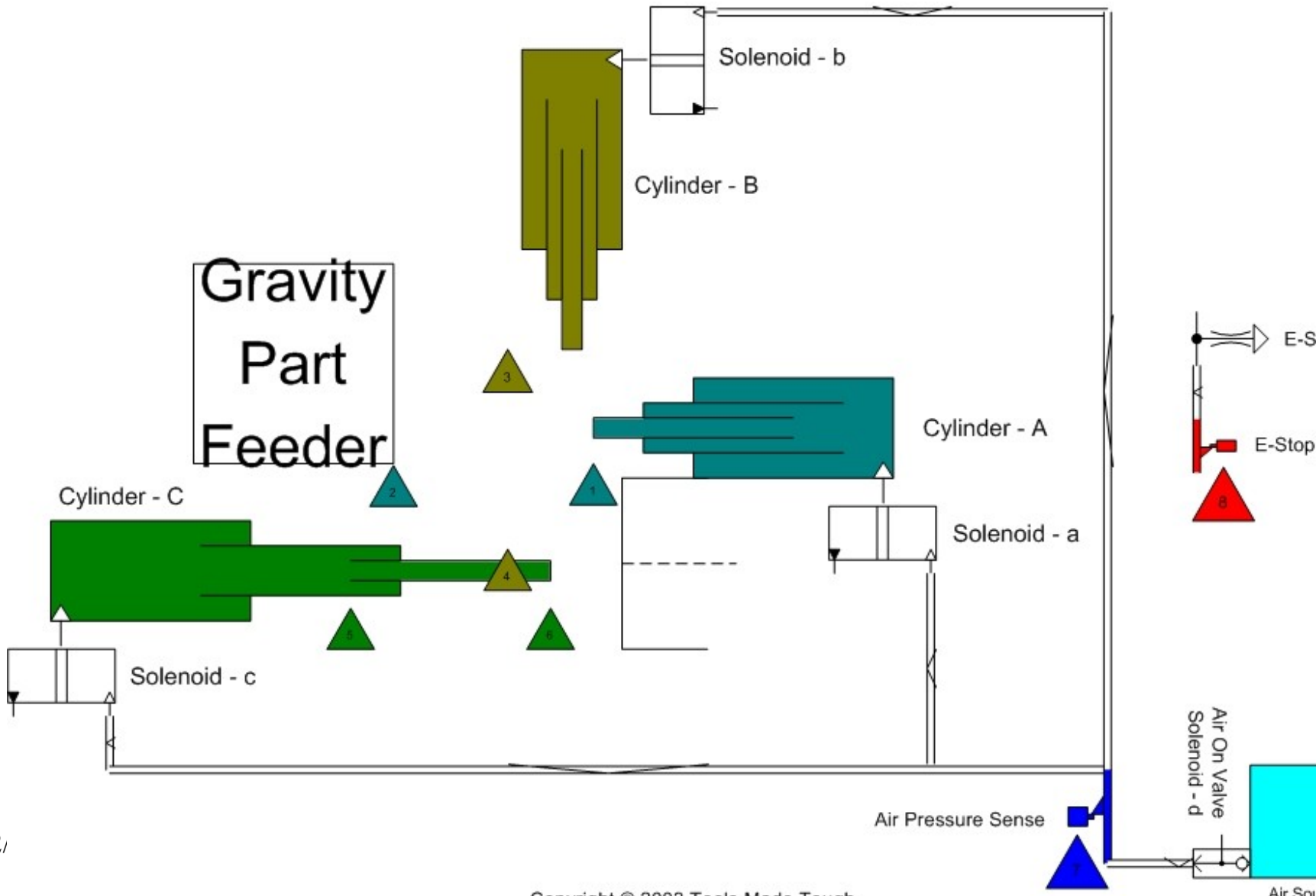
Hypothetical Part Stamping Machine
Design Diagram
Machine Control



Hypothetical Part Stamping Machine
Design Diagram
Machine Control

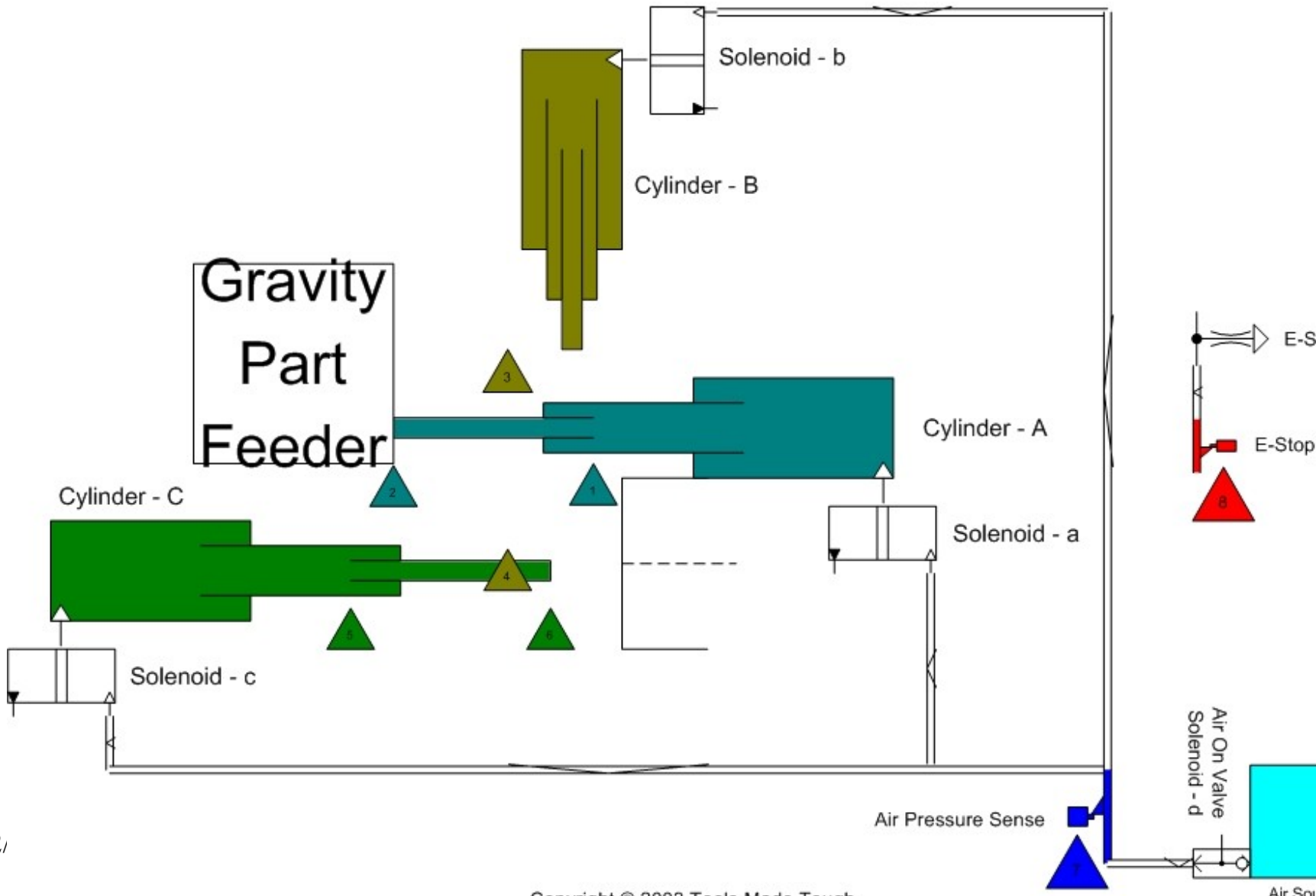


Hypothetical Part Stamping Machine
Design Diagram
Machine Control



02/

Hypothetical Part Stamping Machine
Design Diagram
Machine Control



02/

A Plan of Action

- Understand Requirements
- Analyze Available Resources
- Decide What Can Be Accomplished
- Layout 1 Week Schedule
- Start working on the schedule milestones…

Understand Requirements

- Create a software architecture that…
  - Can handle machine control of N part-stamping machines
  - Provide UI through touch-screen and UART
  - Support timer interrupt

Analyze Available Resources

- No hardware to control
- No embedded control systems to play with
- No RTOS software systems to play with
- A networked desktop workstation, running RHL 7.3
- A plan starts to formulate…

Decide What Can be Accomplished

- Time and resource constraints prevent a working model from being considered
- This implies that anything delivered within 1 week period will consist of part-simulation, part-product
- A simple “perfect world” (see physics problems) simulation at the I/O level yields big dividends

1 Week Schedule

- Day 1 : Analyze Requirements and draft design document
- Day 2 : Start implementing design on selected hardware target
- Day 3 : Design running in Simulation mode
- Day 4 : Design running with I/O control
- Day 5 : Design ported to 2nd platform

PS Design Decisions

- Run PartStamper simulator on RHL7.3
- Build as much of core system as possible
- Remove UI from 1-week horizon
- Provide tool for system stimulus for test purposes
- Implement 5 control processes with a command/reply message interface
  - State Machine
  - Part Grabber, Part Pusher, Part Stamper
  - IO

Project Management 101

- What is the first thing our hero should do now that he is ready to get his hands dirty?
  - Start writing main.c ?
  - Draw some more diagrams ?
  - Fill in some more Pert Chart milestones ?
  - Build project infrastructure ?

Project Infrastructure

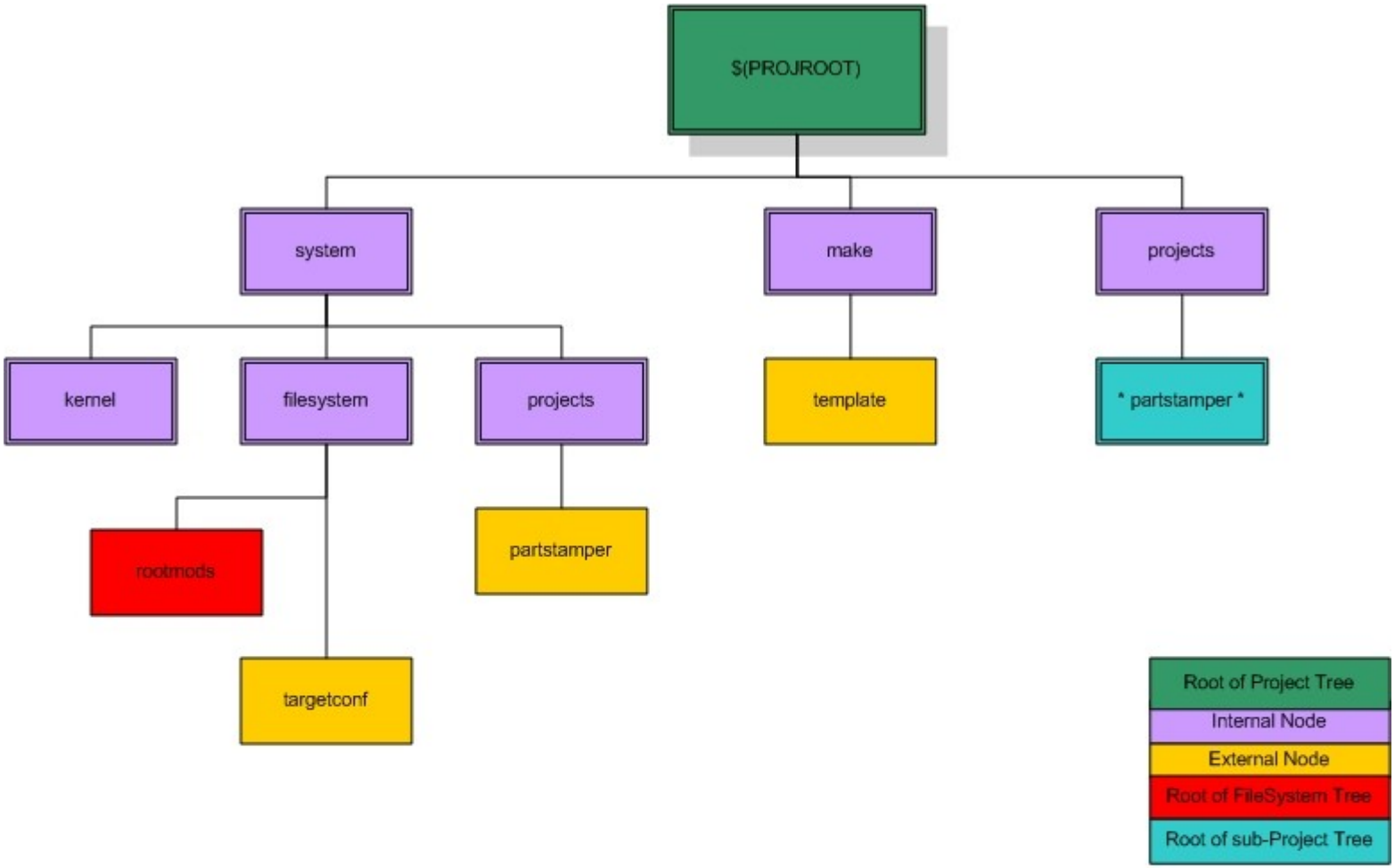
- Should be expandable
- Should support the chosen development tools
- Should support multiple users

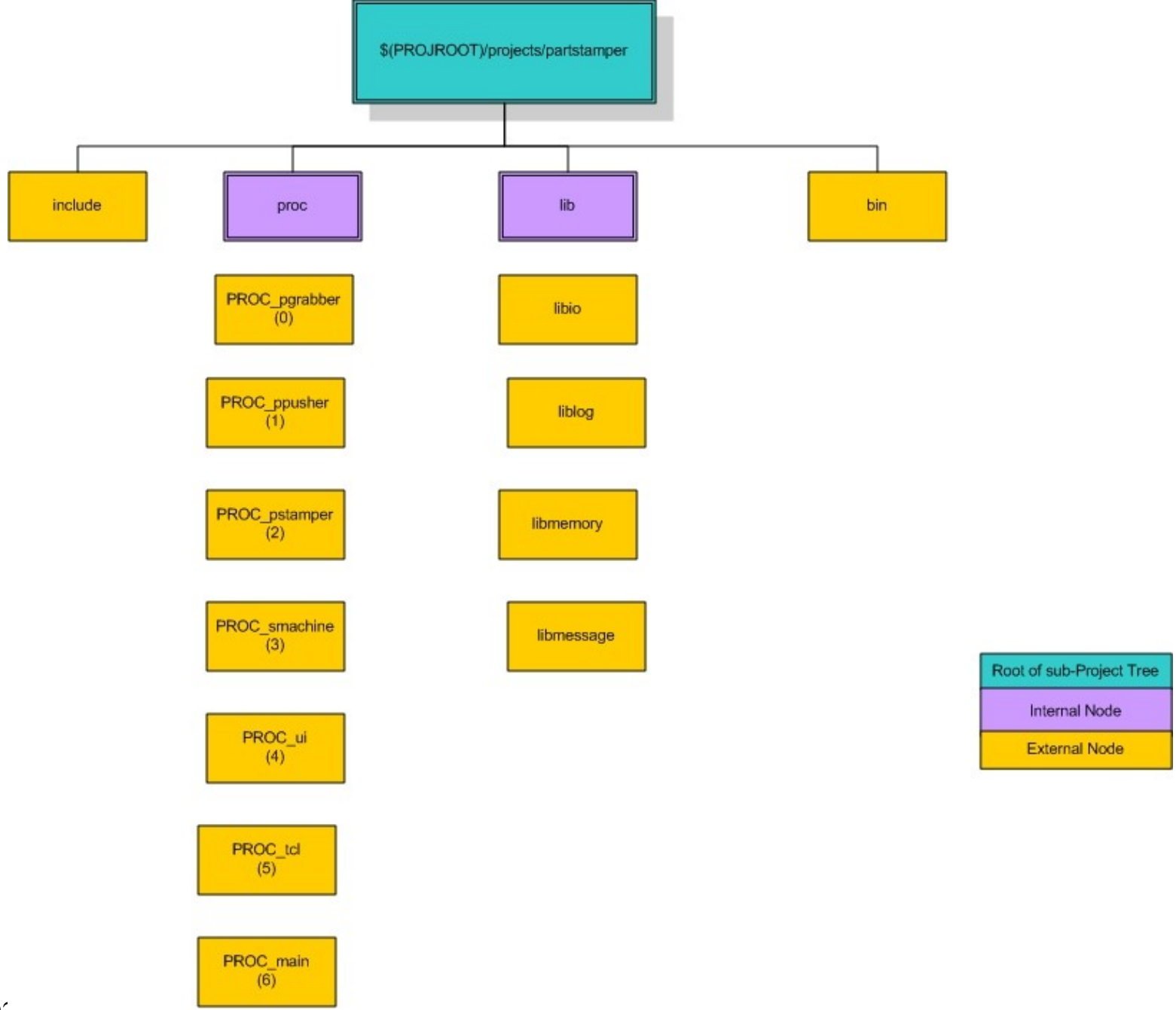
Development Tools

- CVS : Source Code Control System
- make : Project Automation
- gcc : Host Tool Chain
- Tcl : Tool Command Language
- LTT : Linux Trace Toolkit
- Shell scripts

Recursive Project Make System

- Premise
  - A project structure can be implemented that is portable and expandable
  - Builds or ‘make’ can be performed anywhere in the development tree
  - Dependency information is auto-generated
  - Supports command-line tools





1 : The Environment

- The following are candidates from the make system for setting in ~/.bashrc file
  - PROJROOT (**required**)
  - PROJECT (**optional**)
  - ARCHITECTURE **(optional**)

2 : The Make System

- Environment variable PROJROOT referenced in all Makefiles
- $(PROJROOT)/make/project.make is main include file used by make system
- All complexity is hidden in include files in $(PROJROOT)/make

3 : Sample Node Makefile

- $(PROJROOT)/projects/partstamper/Makefile
  - include $(PROJROOT)/make/project.make
  - include $(PROJROOT)/make/target.node

  - # list of sub-directories in this node,
  - # that recursive make system should descend

  DIRS = lib proc

4 : Sample Leaf Makefile

- $(PROJROOT)/projects/partstamper/lib/log/Makefile
  - include $(PROJROOT)/make/project.make
  - # directory name of this leaf
  - LEAF = log
  - # source files in this directory
  - CFILES = $(LEAF).c dump.c dump_msg.c
  - CCFILES =
  - CPPFILES =
  - include $(PROJROOT)/make/target.library
  - include $(LEAF).dpd

5 : List of make include files

- Rules
  - project.make
  - project.host
  - project.x86
  - project.opto22
- Targets
  - target.driver
  - target.library
  - target.process
  - target.node

Log file format

- Name : yyyymmdd.log
- Path : Either /var/log/partstamper or ./
- Entry format :
  - Log Type
  - Timestamp
  - Printf string
  - [TRACE]17:19:31:584: [7].ps_msg_init().
  - Last field in timestamp is milliseconds

Log file format continued

- Log Types
  - LOG_ERROR
  - LOG_DEBUG
  - LOG_TRACE
  - LOG_MESSAGE
  - LOG_TCL
  - LOG_SIMULATE

Today's Agenda

- HPS Project Design Review
  - **General Design Issues with Linux**
  - Inter-process Communication
  - Process List
  - Message Protocol
  - State Machine
  - I/O : Bits and Simulator
  - Tcl Test Harness

Realtime Design, 101

- How many tasks in the system?
- How do the tasks share data?
- Without a UI, how do we test the system?
- How do our design decisions impact the system's time budget?

Our Hero's Project Design

- Use of Linux as a runtime environment
  - Our scorecard indicates there are many options
    - Threads or processes?
    - Pipes or messages?
  - Do I have to write a device driver?
  - Will I have to modify the kernel?

Threads

- Less expensive in terms of system resources:
  - Shares memory space of parent process
  - Depending upon implementation, many thread services are handled in user-space
- Can synchronize/communicate with global memory variables (a double-edged sword)
- Newer and less well supported, in general

Processes

- More expensive in terms of system resources:
  - Memory (replicates parent process memory)
  - Time (kernel call to create/switch)
- Provide protection through unique address space
- Can be a unique program having its own ‘main’
- Are visible to many system tools (PID)

Today's Agenda

- HPS Project Design Review
  - General Design Issues with Linux
  - **Inter-process Communication**
  - Process List
  - Message Protocol
  - State Machine
  - I/O : Bits and Simulator
  - Tcl Test Harness

Inter-process Communication

- Pipes
  - Uni-directional
  - Uses file descriptors
  - Processes must be related
- Named Pipes (FIFOs)
  - Uni-directional
  - Special file in filesystem
  - Unrelated processes may read/write FIFOs
- System-wide size limit on any Pipe

SYS V IPC

- Introduced with AT&T System V.2 release of Unix
  - Semaphores (SEM)
  - Shared Memory (SHM)
  - Message Queues (MSG)
- Allows any two unrelated processes to communicate and/or synchronize

HPS Implementation

- HPS Design Decisions
  - Heavyweight process
  - SYSV IPC MSG
  - SYSV IPC SHM
- Why?
  - Leave room for performance improvement
  - Exposure to more interesting process-level programming entities
  - Push you out of your comfort level (pthreads have many similarities to traditional RTOS)

Today's Agenda

- HPS Project Design Review
  - General Design Issues with Linux
  - Inter-process Communication
  - **Process List**
  - Message Protocol
  - State Machine
  - I/O : Bits and Simulator
  - Tcl Test Harness

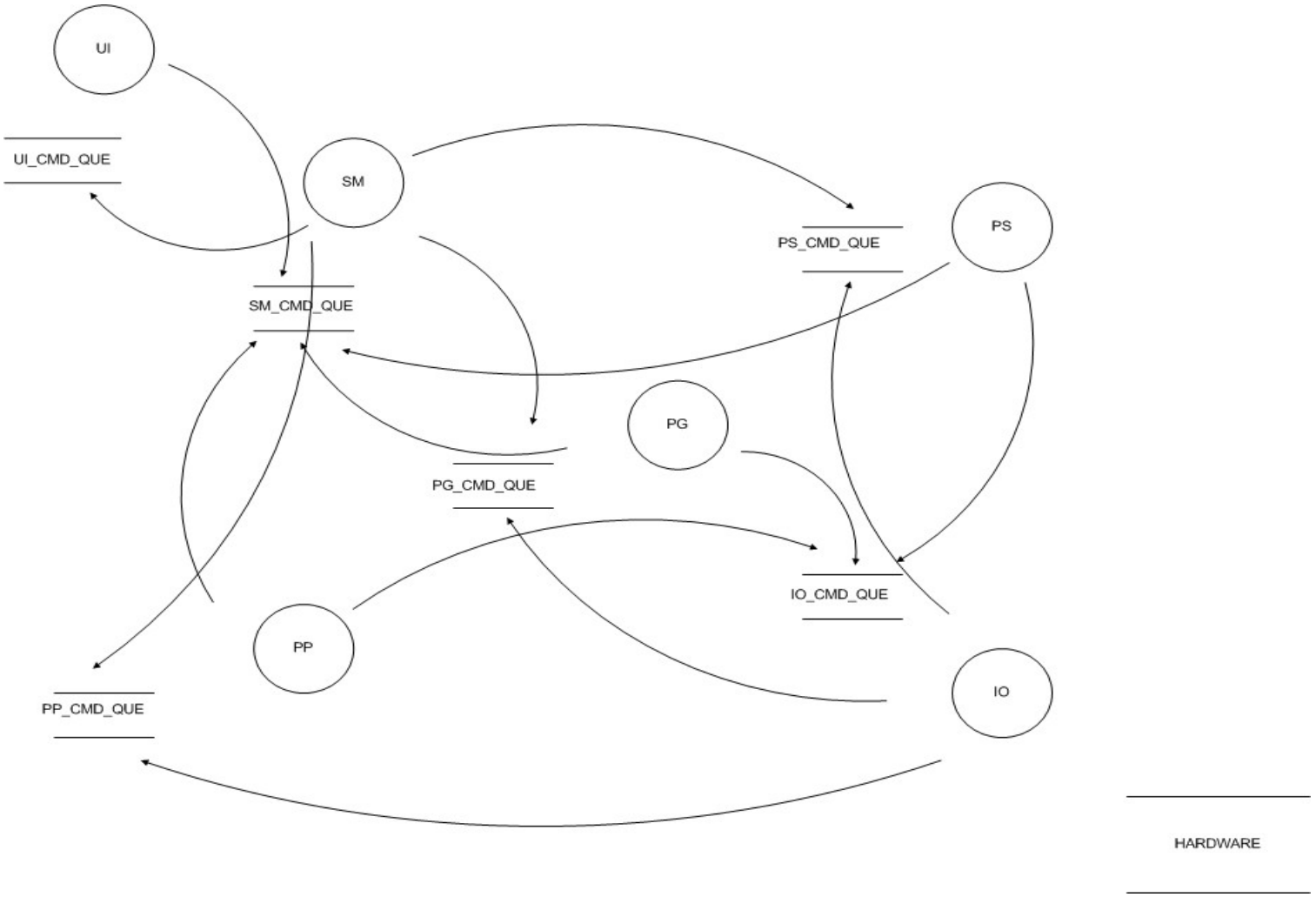
Design Aids

- Data Flow Diagram
- Process Diagram

Generic Data Flow Diagram

The Practical Guide to Structured Systems Design
Meilir Page-Jones, Yourdon Press
Page 60

HPS Data Flow Diagram



Process Diagram

Today's Agenda

- HPS Project Design Review
  - General Design Issues with Linux
  - Inter-process Communication
  - Process List
  - **Message Protocol**
  - State Machine
  - I/O : Bits and Simulator
  - Tcl Test Harness

Part Stamper Messaging

- From the start, our hero's design has revolved around command messages for communication between the system tasks
- This is a methodology that is supported by all commercial embedded RTOS products
- We will uncover the advantages and disadvantages of this approach as we dig into our hero's design and implement it

Messaging in Linux

- One of the benefits of choosing Linux as a development environment is the multitude of ways to solve a problem
- Our hero chose to stick with the SYSV IPC mechanisms because these more closely map to traditional RTOS features
- The following is an excerpt from the msgop man page…

msgop man page

- To send or receive a message, the calling process allocates a structure that looks like the following
  - struct msgbuf {
    - long mtype; /* message type, must be > 0 */
    - char mtext[1]; /* message data */
  - };
  - but with an array mtext of size msgsz, a non-negative integer value. The structure member mtype must have a strictly positive integer value that can be used by the receiving process for message selection

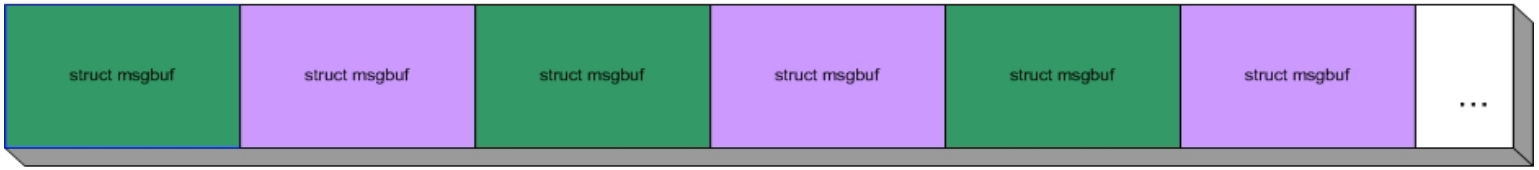
message.h

- #define MSGBUF_SIZE 256
- #define MSGBUF_PAYLOAD_SIZE (MSGBUF_SIZE-sizeof(long))
- Typedef struct {
  - MSG_LIST message;
  - PROC_LIST sender;
  - PROC_LIST receiver;
  - MP_ERROR error;
- } MESSAGE_PACKET;
- typedef struct {
  - MESSAGE_PACKET mp;
  - char data[MSGBUF_PAYLOAD_SIZE-sizeof(MESSAGE_PACKET)];
- } GENERIC_MSG;
- typedef struct {
  - long mtype;
  - GENERIC_MSG gm;

  } msgbuf;

SYSV IPC MESSAGE QUEUE

Msgget returns a handle to a kernel resource, a message queue




```
typedef struct {
   mtype   (long)
   gm      (GENERIC_MSG)
} msgbuf;
```

```
typedef struct {
  mtype          (long)
  gm.mp          (MESSAGE_PACKET)
  gm.data[236]   (char)
} msgbuf
```

```
typedef struct {
  mtype              (long)              4
  gm.mp.message      (MSG_LIST)          4
  gm.mp.sender;      (PROC_LIST)         4
  gm.mp.receiver;    (PROC_LIST)         4
  gm.mp.error;       (MP_ERROR)          4
  gm.data[2366];     (char)            236
} msgbuf                               (256)
```

Linux :	1.6 % data (memory) overhead	(4 / 256 = 0.0157)
PS Message Protocol :	6.3 % data (memory) overhead	(16/256 = 0.0625)
Total :	7.9 % data (memory) overhead	(20 / 256 = 0.078125)

Semantics of the PS Message Protocol

- Each task in the system has an input command message queue
- All message traffic to a task is routed to the input command message queue
- Every outbound command message generates an inbound reply packet, with fields from original message copied/filled in, indicating the command has completed, either successfully or in error

The Message Library

- void generic_msg_free(GENERIC_MSG *);
- GENERIC_MSG * generic_msg_init(MSG_LIST);

- GENERIC_MSG * msg_receive(MSG_LIST);
- void msg_send(GENERIC_MSG *);
- void msg_reply(GENERIC_MSG *);

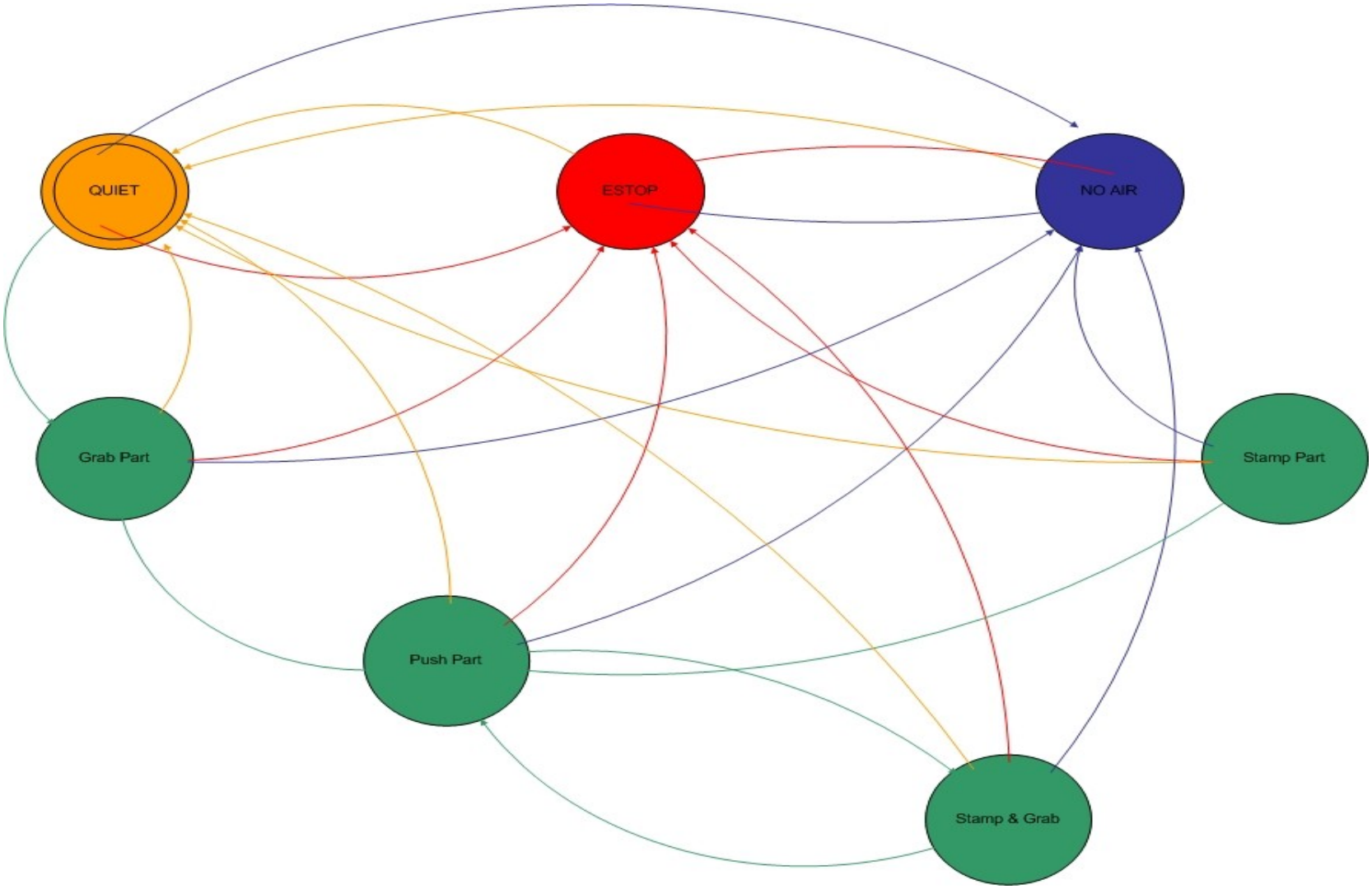
Today's Agenda

- HPS Project Design Review
  - General Design Issues with Linux
  - Inter-process Communication
  - Process List
  - Message Protocol
  - **State Machine**
  - I/O : Bits and Simulator
  - Tcl Test Harness

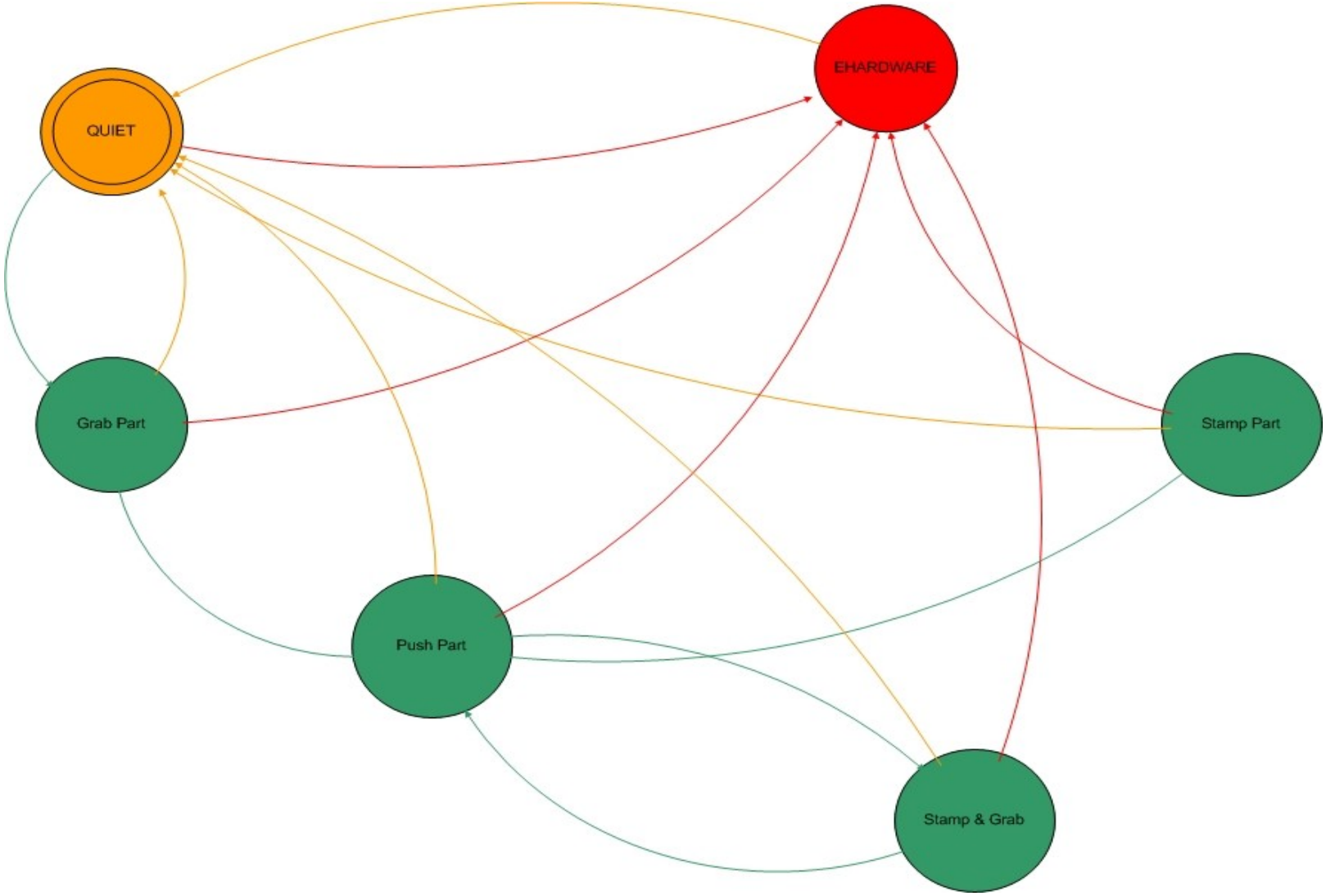
State Machine Basics

- A state machine consists of:
  - An initial state
  - A collection of states
  - A collection of inputs
  - A state transition table (the rules)
  - A terminal state
- An embedded system typically will not have a terminal state and will have a mechanism to return to the initial state (system reset)

HPS State Machine Diagram



HPS Reduced State Machine



StateMachine Cheat Sheet

```
typedef enum {
        sm_start,                // 00
        sm_stop,                 // 01
        sm_return_state,         // 02
        sm_reset,                // 03
        sm_io_value_change,      // 04
        sm_cmd_last
} SM_CMD_LIST;

typedef struct {
        SM_CMD_LIST  command;
        int          continuous;
        int          state;
        IO_BIT_LIST   bit;
        int          value;
} SMCM;
```

```
typedef enum {
        state_quiet,             // 00
        state_ehardware,         // 01
        state_grab,              // 02
        state_push,              // 03
        state_stamp,             // 04
        state_stamp_grab,        // 05
        state_last,
        state_nostate = -1
} SM_STATE;
```

Today's Agenda

- HPS Project Design Review
  - General Design Issues with Linux
  - Inter-process Communication
  - Process List
  - Message Protocol
  - State Machine
  - **I/O : Bits and Simulator**
  - Tcl Test Harness

The I/O Library

- int turn_off(IO_BIT_LIST);
- int turn_on(IO_BIT_LIST);
- int value_of(IO_BIT_LIST);
- int wait_until(IO_BIT_LIST);

I/O Monitor Cheat Sheet

```
typedef enum {
        io_turn_off,   // 00
        io_turn_on,    // 01
        io_value_of,   // 02
        io_report,     // 03
        io_reset,      // 04
        io_cmd_last
} IO_CMD_LIST;
```

```
typedef enum {
        cylAretracted,    // 00
        cylAextended,     // 01
        cylAsolenoid,     // 02
        cylBretracted,    // 03
        cylBextended,     // 04
        cylBsolenoid,     // 05
        cylCretracted,    // 06
        cylCextended,     // 07
        cylCsolenoid,     // 08
        airpressure,      // 09
        Estop,            // 10
        IO_LAST
} IO_BIT_LIST;
```

I/O Simulator

- By using the same technique that all High School and College Physics problems are based on, a very simple and effective solution is found
- When applying all of the forces for an equation, the Physics student is always cautioned to assume the force from friction is neglible and set it to 0
- In the Part Stamper Simulator, I/O activation times are negible, as soon as a control line is toggled, the input values are adjusted.

Today's Agenda

- HPS Project Design Review
  - General Design Issues with Linux
  - Inter-process Communication
  - Process List
  - Message Protocol
  - State Machine
  - I/O : Bits and Simulator
  - **Tcl Test Harness**

Tcl Discussion Agenda

- Tcl
  - What is it?
  - What can you do with it?
  - Why do we care?
- Tcl Meets Realtime Scheduler
  - rt and showsched
  - pslist, pssort, appsched, syssched
- Tcl Extends PartStamper

Tcl : What is it?

- Tool Command Language
  - Similar to other shell languages, e.g.
    - Perl
    - Bash
    - C shell
  - With some important differences…
    - Tk widget (easy window programming)
    - Embeddable
    - Extensible

Tcl : What can you do with it?

- Build product with it
  - HNC Software : SelectResponse
  - WRS : Tornado
- Integrate into product
  - Ampex : System debugger
  - Linux : Kernel Configuration

The C / Tcl Philosophy

- As espoused by John K.Ousterhout…
  - To make a Tcl application as flexible and powerful as possible, you should organize its C code as a set of new Tcl commands that provide a clean set of primitive operations…The purpose of the C code is to provide basic operations that make it easy to implement a wide variety of useful scripts.

Tcl and the Tk Toolkit
John K. Ousterhout, Addison-Wesley
Page 281

Tcl Meets Realtime Scheduler

- rt module includes following commands:
  - rt : change scheduling parameters
  - showsched : show scheduling parameters
  - pslist : ps clone that builds internal arrays
  - pssort : sorts ps array into system and app
  - appsched : show all application processes
  - syssched : show all system processes

Tcl Features used by rt Module

- Multidimensional Arrays
  - Array of structures
  - Set : set psList($index,ppid) $ppid
  - Get : if { $psList($I,$ppid) == 0 } { … }
- Default Parameter Values
  - Proc pslist { {verbose 1} } { … }
- Lists, File I/O
- C interface

Tcl Meets Part Stamper

- pstcl adds following commands :
  - sendcmd : send ps command message
- pstcl adds following linked variables :
  - Log_trace
  - Log_message
  - Log_debug
  - Log_tcl
  - Log_simulate

New Tcl Features Used

- Tcl_LinkVar(Tcl_Interp *interp, …)
  - Links Tcl variable to C variable
  - Limited data type support
    - TCL_LINK_INT
    - TCL_LINK_DOUBLE
    - TCL_LINK_BOOLEAN
    - TCL_LINK_STRING
  - Changes in either world are mapped to otherside, automatically

pstcl : sendcmd

- New Tcl command : sendcmd
- Connects pstcl to rest of PartStamper application
- Allows any command message to be generated and sent in the system
- With a message passing software architecture, this single test command allows virtually all aspects of the system to be tested. With a script-capable test harness, regression tests can be created from the engineer's interactive test tool.

sendcmd Cheat Sheet

```
typedef enum {
        PROC_NONE = -1,
        PROC_pgrabber = 0,
        PROC_ppusher = 1,
        PROC_pstamper = 2,
        PROC_smachine = 3,
        PROC_ui = 4,
        PROC_tcl = 5,
        PROC_main = 6,
        LAST_PROC
} PROC_LIST;
```

```
typedef enum {
        MP_NOERROR,             // 00
        MP_NOMESSAGE,           // 01
        MP_NOSENDER,            // 02
        MP_NORECEIVER,          // 03
        MP_BADCOMMAND,          // 04
        MP_NULLRETURN,          // 05
        MP_UNHANDLEDMSG,        // 06
        MP_BADSTATE,            // 07
        MP_LAST
} MP_ERROR;
```

```
typedef enum {
        PS_COMMAND_MSG   // 00
        SM_COMMAND_MSG   // 01
        UI_COMMAND_MSG   // 02
        PS_GENERIC_MSG   // 03
        LAST_MSG
} MSG_LIST;
```

Part-Stamper Cheat Sheet

```
typedef enum {
        ps_grab_part,     // 00
        ps_push_part,     // 01
        ps_stamp_part,    // 02
        ps_return_state,  // 03
        ps_reset,         // 04
        ps_cmd_last
} PS_CMD_LIST;

 typedef struct {
        PS_CMD_LIST  command;
        int                  state;
 } PSCM;
```