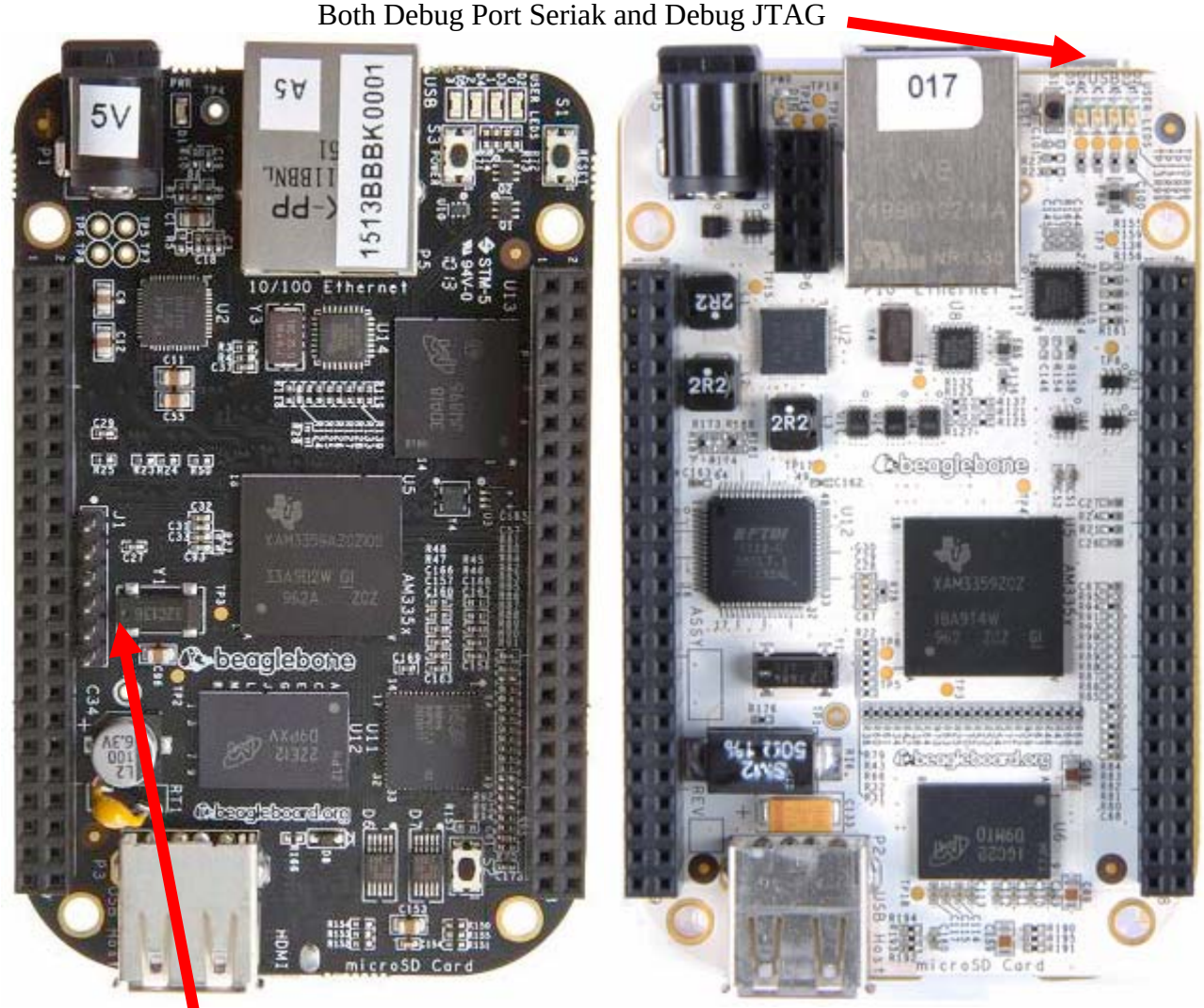
Linux for BeagleBoard – Part 2

Duy-Ky Nguyen **2015-07-04**

Both Debug Port Seriak and Debug JTAG



Debug Port requires USB-Serial adapter

	BeagleBone Black $55	BeagleBone $89
Processor	AM3358BZCZ100, 1GHZ	AM3359ZCZ72, 720MHz
Video Out	HDMI	None
DRAM	512MB DDR3L 800MHZ	256MB DDR2 400MHz
Flash	4GB eMMC, uSD	uSD
Onboard JTAG	Optional	Yes, over USB
Serial	Header	Via USB
PWR Exp Header	**No**	**Yes**
Power	210-460 mA@5V	300-500 mA@5V

1. Introduction

BB uses TI Sitara Processor ARM Cortex-A8 AM335x, Original BB White and new BB Black

Boot from SD card with uboot, otherwise from serial port with “C” waiting for the boot image from PC.
SD Card is partitioned into 2 parts (1)Boot_FAT (2)FS_Ext4

Uboot is compiled into 2 files

- (1) stage 1 raw boot name MUST be renamed to “MLO” from “boot_ti.bin”
- (2) staged 2 load boot name must be reanmed to“app” from “app_ti.bin”

	BeagleBone White	**BeagleBone Black**
Power	USB / Ext_DC (Auto-Select if avail)	Ext_DC
Debug Port	USB (Serial & JTAG &Power)	Ext USB Serial required @ J1
Boot	uSD only	On-board flash (default) uSD if BOOT btn pressed during power-up

1.1. Flashing Download Image

- Download & Install 7Zip
- Download and Install Win32DiskImager used as GUI “dd” tool for raw binary read/write
- Download the latest image from BeagleBoard.org
  BBB-eMMC-flasher-2013.09.04.img.xz

Use 7Zip to unzip the downloaded zipped image
Use Win32DiskImager to write the unzipped image to SD card

1.2. Flashing Boot Files : uboot, kernel device-tree

Partition SD into 2 parts

- FAT (c) **Bootable** (FAT16 or 32 both work)
- Ext 2

1.3. Flashing System File : bone-debian-*.img.xz

tar xz bone-debian-*.img.xz

sudo dd if=bone-debian-*.img of=/dev/sdb2

2. System Build

2.1. ToolChain : Linaro-ARM-2014-09 arm-linux-gnueabihf- (BuildRoot)

2.2. U-Boot : 2013.10 (BuildRoot)

Customized to support

- Update boofiles over network : MLO (stage 1 bootloader), uboot (stage 2 bootloader), kernel & device-tree
- NFS Boot

```
/* NOTE : BLOCK COMMENT must be used, NOT Line-Comment one // !!!*/

#define CONFIG_EXTRA_ENV_SETTINGS \
       "loadaddr=0x80200000\0" \
       "fdtaddr=0x80F80000\0" \
       "rdaddr=0x81000000\0" \
       "mloFile=BBONE/_MLO\0" \
       "ubtFile=BBONE/_UBT\0" \
       "bootfile=BBONE/_zIMG\0" \
       "fdtfile=BBONE/_DTB\0" \
       "console=ttyO0,115200n8\0" \
       "nfs.root=/home/dkn/_NFS/_RFS\0" \
       "nfs.opts=nolock\0" \
       "nfs.args=setenv bootargs console=$console root=/dev/nfs " \
             "nfsroot=${serverip}:${nfs.root},${nfs.opts} rw ip=dhcp ${morebootargs}\0" \
       "nfs.boot=tftp $loadaddr $bootfile ; tftp $fdtaddr $fdtfile ; " \
             "run nfs.args ; bootz $loadaddr - $fdtaddr \0" \
       "mmc.ubt=tftp $loadaddr $mloFile ; fatwrite mmc 0 $loadaddr MLO $filesize ; " \
              "tftp $loadaddr $ubtFile ; fatwrite mmc 0 $loadaddr u-boot.img $filesize \0" \
       "mmc.lnx=tftp $loadaddr $fdtfile ; fatwrite mmc 0 $loadaddr device.dtb $filesize ; " \
              "tftp $loadaddr $bootfile ; fatwrite mmc 0 $loadaddr zImage $filesize \0" \
       "fat.lnx=fatload mmc 0 $rdaddr zImage\0" \
       "fat.dtb=fatload mmc 0 $fdtaddr device.dtb\0" \
       "mmc.args=setenv bootargs console=$console " \
             "root=/dev/mmcblk0p2 rw rootwait \0" \
       "mmc.boot=run mmc.args; mmc rescan; run fat.lnx; run fat.dtb; bootz $rdaddr - $fdtaddr\0" \
       "bootcmd=run mmc.boot\0" \
```

2.3. Linux Kernel

Enable Network options wit TCP/IP, but none of DHCP, …, for NFS Boot

Network options

- TCP/IP networking
  - [ ] IP: DHCP
  - [ ]: BOOTP
  - [ ]: RARP

- Network Filesystem
  - Filesystems
    - Root filesystem on NFS

2.4. Root Filesystem : BuildRoot (BR)

The “dd” tool was used to flash the latest Debian filesystem “bone-debian-7.8-lxde-4gb-armhf-2015-03-01-4gb.img”, but I had to give up after it failed to complete after few hours, probably 3 or 4 !!!

So, a bare minimum buildroot filesystem has been used to create filesystem adding the following packages

- static full busybox : httpd, telnetd, tftp, ifconfig, … just make symbol link if so required
- ntp : ntpdate (correct clock without using onboard realtime clock)
- samba (3) for **Samba Serve**r
- php (CGI) for Server-Side Script : **Web Server**
- perl : script for pattern extraction, a lot more powerful than using awk or sed
- bash (via make)

Full Debian	Bare Minimum Debian Wheezy	Bare Minimum BuildRoot
3800 MB	724 MB	121 MB

BuildRoot-2015.02 was used to build filesystem as it's the latest one to support Samba-3. The next BR-2015.05 has only Samba-4 and it appears broken!?!?

Download and untar BR-2015.02

```
make beaglebone_defconfig
```

- Tool Chain
  - External
  - Linaro ARM 2014.09
  - To be downloaded
- Target packages
  - Busybox
    - Show packages (in order to select BASH in System Config)
  - Interpreter Language
    - Perl : instead of using awk, sed for pattern processing
    - PHP
      - CGI
- Networking app
  - NTP
    - NTP-date
  - Samba
- System Config
  - /bin/bash (default busybox shell not good enough for scripti programming)

```
make busybox-xconfig or gconfig if xconfig broken (normally xconfig OK after gconfig used)
```

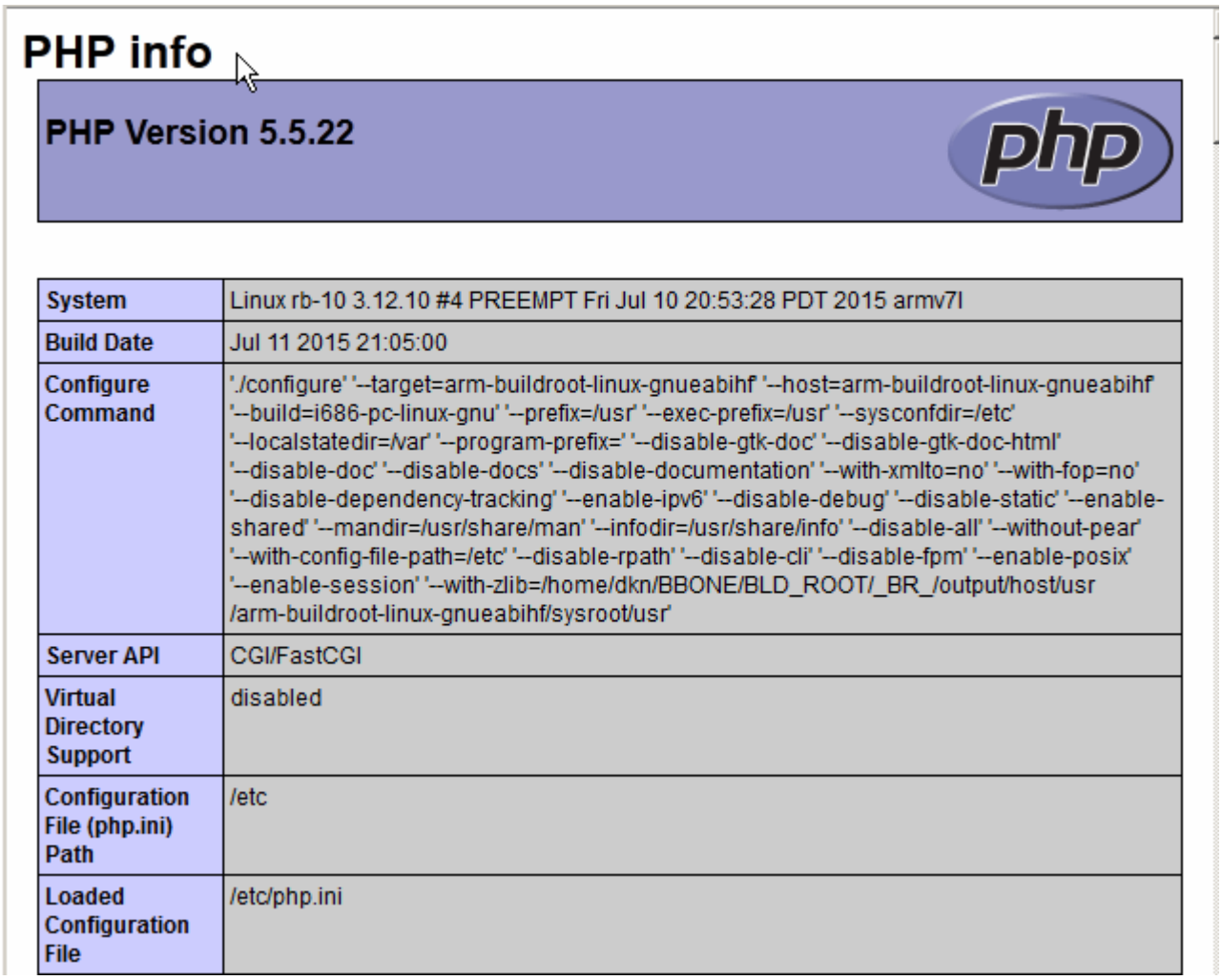
- Build Option
  - Static (not shared libs)
- Network Utilties
  - httpd
  - telnet (UnCheck)
  - telnetd

2.5. HTTPD config

```
## /etc/httpd.conf

# work with PHP file
*.php:/usr/bin/php-cgi
```

```
## /etc/ php.ini

# redirect PHP to HTM page
cgi.force_redirect = 0
```

```
// phpinfo.php

<?
        phpinfo();
?>
```



PHP info

PHP Version 5.5.22

System	Linux rb-10 3.12.10 #4 PREEMPT Fri Jul 10 20:53:28 PDT 2015 armv7l
Build Date	Jul 11 2015 21:05:00
Configure Command	'./configure' '--target=arm-buildroot-linux-gnueabihf' '--host=arm-buildroot-linux-gnueabihf' '--build=i686-pc-linux-gnu' '--prefix=/usr' '--exec-prefix=/usr' '--sysconfdir=/etc' '--localstatedir=/var' '--program-prefix=' '--disable-gtk-doc' '--disable-gtk-doc-html' '--disable-doc' '--disable-docs' '--disable-documentation' '--with-xmlto=no' '--with-fop=no' '--disable-dependency-tracking' '--enable-ipv6' '--disable-debug' '--disable-static' '--enable-shared' '--mandir=/usr/share/man' '--infodir=/usr/share/info' '--disable-all' '--without-pear' '--with-config-file-path=/etc' '--disable-rpath' '--disable-cli' '--disable-fpm' '--enable-posix' '--enable-session' '--with-zlib=/home/dkn/BBONE/BLD_ROOT/_BR_/output/host/usr /arm-buildroot-linux-gnueabihf/sysroot/usr'
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc
Loaded Configuration File	/etc/php.ini

In-line perl script to check eth0 before doing dhcp-client

```
# /etc/init.d/rcS
if [[ ! $(ifconfig | perl -lane 'print $1, if /eth0/') ]]; then
       printf "\n>> eth0 NOT config'ed : Start uDHCPc\n"
       udhcpc -q -i eth0
else
       printf "\n>> eth0 config'ed\n"
fi
```

In-line perl script for prompt with IP-addr

```
HOSTNAME=$(ifconfig | perl -lane 'print $1, if /inet addr:(\S*)/' | perl -lane 'print, unless
/127/' | perl -lane 'printf "rb-%d", $1, if /\d*\.\d*\.\d*\.(\d*)/')

hostname $HOSTNAME

PS1="${HOSTNAME}:\w % "
```

Original Debian root filesystem screen capture is **HERE**

New BuildRoot root filesystem screen capture is **HERE**

3. Linux SysInit

There're 3 approaches in transfer control from kernel space to user space : (1) sys-V init (2) busybox init (3) new SystemD (4) Upstart init (intctl). Both (1) & (2) have 2 same files "/etc/inittab" invoking "/etc/init.d/rcS"; and the same directory "/etc/init.d".

The new SystemD is quite different using "systemclt". It has

- NO "/etc/inittab"
- "/etc/init.d" is still there for legacy jobs like "cron", "xinetd", …
- "/lib/systemd" under system control
- "/etc/systemd" under user control;

I found systemD is NOT suitable, too "bureaucracy" for embedded system, too much debugging effort

The Upstart Init used in Ubuntu (probably since v-12) is even worse, almost impossible to debug, the traditional init runs in sequence, this new init does not. It's suitable for dynamic server environment, say adding new known-working storage device to a cloud server, but problem-free startup due to stable HW. But for embedded system, there's no such problem-free startup due to "dynamic" HW environment with unknown-working HW device. I doubt if cloud server system dare to use Ubuntu for their servers!

In both cases, "/etc/init.d" stores all system run-control scripts.

It's possible to use exactly the same files and directory for both approaches, but it's just a myst to me why sysV init chose the hard way to do by creating many run-control directories, rc0.d, rc1.d, ,,,, rcS.d then just puts links in those directories and point to scripts in "int.d". In addition, most of rcN.s never get executed, it stops at default runlevel specified in inittab, say 2, so ignores the rest!

So, the sequence is : uboot – kernel – init : inittab – rcX – profile – bashrc : user

3.1. Sys-V Init

Typical implementation of Sys-V init has 8 run-control directories : rc0.d ~ rc6.d and rcS.d. It always starts with rcS.d first, then rc0.d, till rcN.d where "N" is default run-level specified in "inittab". All these rcX.d store links to run-control scripts in "/etc/init.d"

All run-control scripts is based solely on the script "/etc/init.d/rc" where its argument is "S", "0", "1", … "6". This script does jobs to start – stop – restart – status.

"Inittab" first invokes "/etc/rcS" which in turn just calls "rc" with argument "S", then "rc" with "0", "1", … finally stops at default run-level, say 2 as below

```
#! /bin/sh

# /etc/init.d/rcS

exec /etc/init.d/rc S
```

```
# sysyinit-v5
```

```
# /etc/inittab: init(8) configuration.
# $Id: inittab,v 1.91 2002/01/25 13:35:21 miquels Exp $

# The default runlevel.
id:2:initdefault:

# Boot-time system configuration/initialization script.
# This is run first except when booting in emergency (-b) mode.
si::sysinit:/etc/init.d/rcS

# What to do in single-user mode.
~~:S:wait:/sbin/sulogin

# /etc/init.d executes the S and K scripts upon change
# of runlevel.
#
# Runlevel 0 is halt.
# Runlevel 1 is single-user.
# Runlevels 2-5 are multi-user.
# Runlevel 6 is reboot.

l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
# Normally not reached, but fallthrough in case of emergency.
z6:6:respawn:/sbin/sulogin

# What to do when CTRL-ALT-DEL is pressed.
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

# Action on special keypress (ALT-UpArrow).
#kb::kbrequest:/bin/echo "Keyboard Request--edit /etc/inittab to let this work."

# What to do when the power fails/returns.
pf::powerwait:/etc/init.d/powerfail start
pn::powerfailnow:/etc/init.d/powerfail now
po::powerokwait:/etc/init.d/powerfail stop

# /sbin/getty invocations for the runlevels.
#
# The "id" field MUST be the same as the last
# characters of the device (after "tty").
#
# Format:
#  <id>:<runlevels>:<action>:<process>
#
# Note that on most Debian systems tty7 is used by the X Window System,
# so if you want to add more getty's go ahead but skip tty7 if you run X.
```

```
#
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
3:23:respawn:/sbin/getty 38400 tty3
4:23:respawn:/sbin/getty 38400 tty4
5:23:respawn:/sbin/getty 38400 tty5
6:23:respawn:/sbin/getty 38400 tty6

# Example how to put a getty on a serial line (for a terminal)
#
#T0:23:respawn:/sbin/getty -L ttyS0 9600 vt100
#T1:23:respawn:/sbin/getty -L ttyS1 9600 vt100

# Example how to put a getty on a modem line.
#
#T3:23:respawn:/sbin/mgetty -x0 -s 57600 ttyS3

## MUST
T0:23:respawn:/sbin/getty -L ttyO0 115200 vt102
```

3.2. Busybox Init

Busybox init has no rcX.d, as all rcX.d just stores links to scripts in “init.d” anyway!

“inittab” invokes “/etc/init.d/rcS”, “rcS” in turn executes all scripts in “/etc/init.d” start with “**Snn**” where “nn” are numbers used to order the execution, say “S10” before “S20”, …

```
# busybox init
# /etc/inittab
#
# Copyright (C) 2001 Erik Andersen <andersen@codepoet.org>
#
# Note: BusyBox init doesn't support runlevels.  The runlevels field is
# completely ignored by BusyBox init. If you want runlevels, use
# sysvinit.
#
# Format for each entry: <id>:<runlevels>:<action>:<process>
#
# id        == tty to run on, or empty for /dev/console
# runlevels == ignored
# action    == one of sysinit, respawn, askfirst, wait, and once
# process   == program to run

# Startup the system
null::sysinit:/bin/mount -t proc proc /proc
null::sysinit:/bin/mount -o remount,rw /
null::sysinit:/bin/mkdir -p /dev/pts
null::sysinit:/bin/mkdir -p /dev/shm
null::sysinit:/bin/mount -a
null::sysinit:/bin/hostname -F /etc/hostname
# now run any rc scripts
::sysinit:/etc/init.d/rcS

# Put a getty on the serial port
ttymxc1::respawn:/sbin/getty -L  ttymxc1 0 vt100 # GENERIC_SERIAL
```

```
# Stuff to do for the 3-finger salute
::ctrlaltdel:/sbin/reboot

# Stuff to do before rebooting
::shutdown:/etc/init.d/rcK
::shutdown:/sbin/swapoff -a
::shutdown:/bin/umount -a –r
```

```
#!/bin/sh

# Start all init scripts in /etc/init.d
# executing them in numerical order.
#
for i in /etc/init.d/S??* ;do

     # Ignore dangling symlinks (if any).
     [ ! -f "$i" ] && continue

     case "$i" in
       *.sh)
           # Source shell script for speed.
           (
              trap - INT QUIT TSTP
              set start
              . $i
           )
           ;;
       *)
           # No sh extension, so fork subprocess.
           $i start
           ;;
    esac
done
```

4. Conclusion

	Original	**New**
U-Boot	FAT	FAT
Kernel	Ext2 (filesystem	FAT
Update Boot	On PC (Copy)	On Target over network
Root Filesystem	Debian 4- GB	Build-Root 120 MB • Bash, Perl as script language • PHP as Server-Side Script • Samber server • Web server • NTP : realtime clock